Matteo Baldoni
Jamal Bentahar
M. Birna van Riemsdijk
John Lloyd (Eds.)

LNAI 5948

Declarative Agent Languages and Technologies VII

7th International Workshop, DALT 2009
Budapest, Hungary, May 2009
Revised Selected and Invited Papers

Lecture Notes in Artificial Intelligence 5948

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

Matteo Baldoni Jamal Bentahar
M. Birna van Riemsdijk John Lloyd (Eds.)

Declarative Agent Languages and Technologies VII

7th International Workshop, DALT 2009
Budapest, Hungary, May 11, 2009
Revised Selected and Invited Papers

Springer

Volume Editors

Matteo Baldoni
University of Turin, Italy
E-mail: baldoni@di.unito.it

Jamal Bentahar
Concordia University, Montreal, Canada
E-mail: bentahar@ciise.concordia.ca

M. Birna van Riemsdijk
Delft University of Technology, Delft, The Netherlands
E-mail: m.b.vanriemsdijk@tudelft.nl

John Lloyd
Australian National University, Canberra ACT, Australia
E-mail: john.lloyd@anu.edu.au

Library of Congress Control Number: 2009943839

CR Subject Classification (1998): I.2.11, C.2.4, D.2.4, D.2, D.3, F.3.1

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN	0302-9743
ISBN-10	3-642-11354-0 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-11354-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12832613 06/3180 5 4 3 2 1 0

Preface

In the multi-agent systems area, linking theory to practical applications is still a fertile research topic. The aim of the workshop on Declarative Agent Languages and Technologies (DALT 2009), in its seventh edition this year, is to achieve this goal, which needs developing and using advanced declarative technologies and languages, particularly agent programming, communication languages, and reasoning and decision-making mechanisms. Developing these technologies is a particularly challenging issue from many perspectives: formal foundations, practical feasibility, degree of flexibility, etc. In this context, the declarative paradigm is arguably the most appropriate as unlike imperative approaches, the focus is on what the solution should accomplish rather than on describing how to accomplish it. This is because agent computing, as a paradigm, is about describing the logic of computation instead of describing how to accomplish it. DALT is about investigating, studying, and using the declarative paradigm as well as combining declarative and formal approaches with engineering and technology aspects of agents and multi-agent systems.

This volume presents the latest developments in the area of declarative languages and technologies, which aim to provide rigorous frameworks for designing, specifying, implementing and verifying autonomous interacting agents. These frameworks are based on computational logics and other formal methods such as mathematical models and game theoretical approaches. Using such models and approaches facilitates the development of agents that reason and act rationally while at the same time being able to verify the behavior of these agents against their specification. The main theme of DALT 2009 was the further advancement of relevant specification and verification techniques, such as, for instance, modal and epistemic logics, model checking, constraint logic programming, and distributed constraint satisfaction.

As one of the well-established workshops in the multi-agent systems area, DALT 2009 was held as a satellite workshop of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), in Budapest, Hungary. Following the success of DALT 2003 in Melbourne (LNAI 2990), DALT 2004 in New York (LNAI 3476), DALT 2005 in Utrecht (LNAI 3904), DALT 2006 in Hakodate (LNAI 4327), DALT 2007 in Honolulu (LNAI 4897), and DALT 2008 in Estoril (LNAI 5397), DALT 2009 aimed at providing a discussion forum to both (a) support the transfer of declarative paradigms and techniques to the broader community of agent researchers and practitioners; and (b) bring the issue of designing complex agent systems to the attention of researchers working on declarative languages and technologies. DALT has traditionally fostered the development of declarative approaches to engineering agent-based systems and applications in different areas such as the Semantic Web, service-oriented computing, Web services, security, and electronic contracting.

The book includes 15 chapters: one by the invited speaker; 3 from AAMAS 2009 short papers (revised and augmented versions); and 11 from DALT 2009. All the papers were carefully reviewed to check the originality, quality, and technical soundness. The DALT 2009 workshop received 17 papers. After a rigorous reviewing process by at least 3 reviewers, 11 papers were selected by the Program Committee to be published in this volume. Chapter 1: "Playing with Rules" by João Leite is about discussing agent-oriented programming languages by focussing on two languages that use logic rules and rule updates, namely, answer-set programming and evolving logic programming. Chapter 2: "The Refinement of Choreographed Multi-Agent Systems" by Lăcrămioara Aştefănoaei, Mehdi Dastani and Frank S. de Boer is about generalizing a theory of agent refinement to multi-agent systems, where coordination mechanisms and real time are key issues. The proposed refinement is compositional, which reduces the verification process. Chapter 3: "Goal Generation from Possibilistic Beliefs Based on Trust and Distrust" by Célia da Costa Pereira and Andrea Tettamanzi discusses some agents belief behaviors used in a goal generation and adoption framework by focussing on the trustworthiness of the source of information that depends not only on the degree of trust but also on an independent degree of distrust. Chapter 4: "Monitoring Directed Obligations with Flexible Deadlines: a Rule-Based Approach" by Henrique Lopes Cardoso and Eugénio Oliveira introduces, in a B2B cooperation setting, an approach to model contractual commitments through directed obligations with time windows, where authorizations granted at specific states of an obligation life cycle model are considered.

Chapter 5: "Unifying the Intentional and Institutional Semantics of Speech Acts" by Carole Adam, Andreas Herzig, Dominique Longin and Vincent Louis addresses the semantic issue of agent communication languages mixing the mentalist and social approaches. This semantics extends FIPA-ACL with new speech acts along with new institutional features. Chapter 6: "Tableaux for Acceptance Logic" by Mathijs de Boer, Andreas Herzig, Tiago de Lima and Emiliano Lorini presents a modal logic for modeling individual and collective acceptances called acceptance logic and a sound and complete tableau method that automatically decides whether a formula of the logic is satisfiable. Chapter 7: "Ontology and Time Evolution of Obligations and Prohibitions Using Semantic Web Technology" by Nicoletta Fornara and Marco Colombetti formalizes conditional obligations and prohibitions with starting times and deadlines using social commitments and models them in OWL, the logical language used to specify semantic web applications. Chapter 8: "Prioritized Goals and Subgoals in a Logical Account of Goal Change – A Preliminary Report" by Shakil Khan and Yves Lesperance develops a logical framework for goal change considering the dynamics of prioritized goals and subgoals. Lower priority goals are not drop permanently, but they are considered inactive and can become active in the future.

Chapter 9: "Declarative and Numerical Analysis of Edge Creation Process in Trust-Based Social Networks" by Babak Khosravifar, Jamal Bentahar and Maziar Gomrokchi addresses the efficiency issue of the interactions among agents

in a social network by focussing on some trust-based factors. It presents declarative and numerical analysis of the proposed model and its assessment along with empirical evaluation. Chapter 10: "Computing Utility from Weighted Description Logic Preference Formulas" by Azzurra Ragone, Tommaso Di Noia, Francesco M. Donini, Eugenio Di Sciascio and Michael Wellman proposes a framework to compute the utility of a proposal considering a preference set in a negotiation process, where preferences are dealt with as weighted formulas in a decidable fragment of first order logic. Chapter 11: "Explaining and Predicting the Behavior of BDI-Based Agents in Role-Playing Games" by Michal Sindlar, Mehdi Dastani, Frank Dignum and John-Jules Meyer discusses the use of BDI agents to model virtual characters in games. It illustrates how these agents can infer the mental state of other virtual characters by observing others' actions in a role-playing game. Chapter 12: "Correctness Properties for Multiagent Systems" by Munindar Singh and Amit Chopra discusses the characteristics of some correctness properties for interacting agent-based systems, which are commitment-centered. Examples of these properties are interoperability, which is mapped to commitment alignment and compliance expressed as commitment discharge.

Chapter 13: "Reasoning and Planning with Cooperative Actions for Multi-agents Using Answer Set Programming" by Tran Cao Son and Chiaki Sakama introduces a framework to represent and reason about plans with cooperative actions of an agent operating in a multi-agent system. An extended action language (the action language $\mathcal{A}$) has been used to formalize the multi-agent planning problem and the notion of joint plans that are computed using answer set programming. Chapter 14: "Social Commitments in Time: Satisfied or Compensated" by Paolo Torroni, Federico Chesani, Paola Mello and Marco Montali formalizes the time evolution of commitments within a framework based on computational logic and on a reactive axiomatization of the event calculus. The framework proposes a new characterization of commitments with time that enables run-time and static verification. Chapter 15: "Verifying Dribble Agents" by Doan Thu Trang, Brian Logan and Natasha Alechina addresses the model-checking problem of programs written in the agent programming language Dribble. An extension of the computation tree logic CTL, which describes transition systems corresponding to a Dribble program, has been proposed and the MOCHA model checker has been used for simulation.

We would like to thank all the authors for their enthusiasm to submit papers to the workshop and revise them for inclusion in this book, the members of the Steering Committee for their valuable suggestions and support, and the members of the Program Committee for their excellent work during the reviewing phase.

November 2009

Matteo Baldoni
Jamal Bentahar
John Lloyd
M. Birna van Riemsdijk

Organization

Workshop Organizers

Matteo Baldoni	University of Turin, Italy
Jamal Bentahar	Concordia University, Canada
John Lloyd	Australian National University, Australia
M. Birna van Riemsdijk	Delft University of Technology, The Netherlands

Program Committee

Thomas Ågotnes	Bergen University College, Norway
Marco Alberti	University of Ferrara, Italy
Natasha Alechina	University of Nottingham, UK
Cristina Baroglio	University of Turin, Italy
Rafael Bordini	University of Durham, UK
Jan Broersen	University of Utrecht, The Netherlands
Federico Chesani	University of Bologna, Italy
Amit Chopra	North Carolina State University, USA
Keith Clark	Imperial College London, UK
James Harland	RMIT University, Australia
Andreas Herzig	Paul Sabatier University, France
Koen Hindriks	Delt University of Technology, The Netherlands
Shinichi Honiden	National Institute of Informatics, Japan
Yves Lespérance	York University, Canada
Alessio Lomuscio	Imperial College London, UK
Viviana Mascardi	University of Genoa, Italy
Nicolas Maudet	University of Paris-Dauphine, France
John-Jules Ch. Meyer	Utrecht University, The Netherlands
Peter Novak	Clausthal University of Technology, Germany
Enrico Pontelli	New Mexico State University, USA
Azzurra Ragone	Polytechnic of Bari, Italy
Chiaki Sakama	Wakayama University, Japan
Guillermo Simari	Universidad Nacional del Sur, Argentina
Tran Cao Son	New Mexico State University, USA
Eugenia Ternovska	Simon Fraser University, Canada
Wamberto Vasconcelos	University of Aberdeen, UK
Marina De Vos	University of Bath, UK
Michael Winikoff	University of Otago, New Zealand

Steering Committee

Matteo Baldoni	University of Turin, Italy
Andrea Omicini	University of Bologna-Cesena, Italy
M. Birna van Riemsdijk	Delft University of Technology, The Netherlands
Tran Cao Son	New Mexico State University, USA
Paolo Torroni	University of Bologna, Italy
Pınar Yolum	Bogazici University, Turkey
Michael Winikoff	University of Otago, New Zealand

Additional Reviewers

Aaron Hunter
Tommaso Di Noia

Table of Contents

Invited Talk

Invited Papers

Contributed Papers

Playing with Rules

João Leite

CENTRIA, Universidade Nova de Lisboa, Portugal
jleite@di.fct.unl.pt

Abstract. In this paper we revisit Logic Programming under the answer-set semantics - or *Answer-Set Programming* - and its extension *Evolving Logic Programming*, two languages that use logic rules and rule updates and exhibit characteristics that make them suitable to be used for knowledge representation and reasoning within Agent Oriented Programming Languages. We illustrate the power of these rule based languages by means of examples showing how several of its features can be used to model situations faced by Agents.

1 Introduction

Recently, there has been considerable amount of research in designing Agent Oriented Programming Languages (AOPL). From the definition of the syntax and semantics to the development of the necessary tools and infrastructure (editors, debuggers, environments, etc.), the results of such efforts are key in turning these AOPLs into practical tools which exhibit the necessary level of maturity to make them compete with more established languages, and help widespread the Agent Oriented Programming paradigm. The collections of papers [8,9] reflect the state of the art on this subject.

In most cases, the designers of Agent Oriented Programming Languages have focused on developing an appropriate syntax which promotes to first class citizens notions such as beliefs, goals, intentions, plans, events, messages, etc., and defining a semantics which, for most cases, concentrates on the interaction between these notions and the definition of more or less complex transition rules which define the behaviour of the agent. These two tasks are not trivial ones. On the one hand, defining and fixing a particular syntax faces the big tension between flexibility and ease of use: fewer prescribed syntactical constructs, as often seen in declarative languages, usually provide greater flexibility but are harder to use, while more prescribed syntactical constructs are usually easier to use, at the cost of flexibility. On the other hand, defining a semantics that properly deals with these high level notions (beliefs, intentions, plans, goals, etc...), is efficient, to some extent declarative, and keeps some desirable (theoretical) principles is a very hard task.

When looking at existing AOPLs, one often sees that in order to achieve acceptable results in what concerns the appropriate balance between the definition of a syntax with high level agent oriented notions and an appropriate semantics, their developers need to make some compromises, often in the form of simplifications in the expressiveness of the languages used for specifying the underlying high level notions, with consequences in what can be expressed and how the agent can behave. For example, if

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 1–19, 2010.

we consider the representation of beliefs, it is often the case that the knowledge representation languages used are not more expressive than a set of horn clauses, and belief revision/update mechanisms amount to the simple addition/retraction of facts. In practice, it is often the case that PROLOG is used to represent and reason about knowledge, and implementations diverge from the theoretical definitions, often with unforeseen, unintended and unintuitive results.

There is substantial literature illustrating the need for knowledge representation languages that are richer than those limited to horn clauses, e.g. exhibiting non-monotonicity [10,23], and richer belief change operators that permit not only the update of the extensional part of existing knowledge (the facts), but also the intensional part (the rules) [19,5]. And these should be accompanied by implementations that accurately implement the theory, which should preferably be (at least) decidable. The incorporation of richer forms of knowledge representation in current AOPLs should be investigated, not only to allow agent's beliefs to be more expressive, but also to specify other agent high level notions such as their goals (e.g. to represent conditional goals and their updates), richer forms of intentions, etc.

In this paper we revisit Logic Programming under the answer-set semantics [15] - or *Answer-Set Programming* - and its extension *Evolving Logic Programming*, [3], two languages with characteristics that make them suitable to be used for knowledge representation and reasoning within AOPLs, and illustrate some of their characteristics by means of examples.

Answer-Set Programming (ASP) is a form of declarative programming that is similar in syntax to traditional logic programming and close in semantics to non-monotonic logic, that is particularly suited for knowledge representation. Some of the most important characteristics of ASP include: the use of default negation to allow for reasoning about incomplete knowledge; a very intuitive semantics based on multiple answer-sets for reasoning about several possible consistent worlds; possibility to compactly represent all NP and coNP problems if non-disjunctive logic programs are used, while disjunctive logic programs under answer sets semantics capture the complexity class Σ_2^P [14]; fully declarative character in the sense that the program specification resembles the problem specification; the existence of a number of well studied extensions such as preferences, revision, abduction, etc. Enormous progress concerning the theoretical foundations of ASP (c.f. [7] for more) have been made in recent years, and the existence of very efficient ASP solvers (e.g. DLV and SMODELS) has made it possible to use it in real applications such as Decision Support for the Space Shuttle [24], Automated Product Configuration [26], Heterogeneous Data Integration [20], Inferring Phylogenetic Trees [11], Resource Allocation [17], as well as Reasoning about actions, Legal Reasoning, Games, Planning, Scheduling, Diagnosis, etc.

While ASP can be seen as a good representation language for static knowledge, if we are to move to a more open and dynamic environment, typical of the agency paradigm, we must consider ways and means of representing and integrating knowledge updates from external as well as internal sources. In fact, an agent should not only comprise knowledge about each state, but also knowledge about the transitions between states. The latter may represent the agent's knowledge about the environment's evolution, coupled to its own behaviour and evolution. The lack of rich mechanisms to

represent and reason about dynamic knowledge and agents i.e. represent and reason about environments where not only some facts about it change, but also the rules that govern it, and where the behaviours of agents also change, is common to most existing AOPLs.

To address this issue the paradigm of *Evolving Logic Programming* (*EVOLP*) was introduced in [3][1]. In a nutshell, *EVOLP* is a simple though quite powerful extension of ASP [15] that allows for the specification of a program's evolution, in a single unified way, by permitting rules to indicate assertive conclusions in the form of program rules. Syntactically, evolving logic programs are just generalized logic programs[2]. But semantically, they permit to reason about updates of the program itself. The language of *EVOLP* contains a special predicate $assert/1$ whose sole argument is a full-blown rule. Whenever an assertion $assert\,(r)$ is true in a model, the program is updated with rule r. The process is then further iterated with the new program. These assertions arise both from self (i.e. internal to the program) updating, and from external updating originating in the environment. EVOLP can adequately express the semantics resulting from successive updates to logic programs, considered as incremental specifications of agents, and whose effect can be contextual. Whenever the program semantics allows for several possible program models, evolution branching occurs, and several evolution sequences are made possible. This branching can be used to specify the evolution of a situation in the presence of incomplete information. Moreover, the ability of *EVOLP* to nest rule assertions within assertions allows rule updates to be themselves updated down the line. Furthermore, the *EVOLP* language can express self-modifications triggered by the evolution context itself, present or future – $assert$ literals included in rule bodies allow for looking ahead on some program changes and acting on that knowledge before the changes occur. In contradistinction to other approaches, *EVOLP* also automatically and appropriately deals with the possible contradictions arising from successive specification changes and refinements (via Dynamic Logic Programming[3] [19,5,2]).

In this paper we start by revisiting EVOLP and illustrate how its features, many of which inherited from ASP, seem appropriate to represent and reason about several aspects related to Agents and Agent Oriented Programming.

The remainder of the paper is structured as follows: In Sect. 2 we present a very simple agent architecture that will be used in the remainder of the paper. In Sect. 3 we illustrate many features of EVOLP and ASP in this context, and we draw some conclusions and pointers to future work in Sect. 4. In Appendix A we recap the syntax and semantics of EVOLP.

[1] There are implementations of EVOLP [25] available at http://centria.di.fct.unl.pt/~jja/updates.

[2] Logic programs that allow for rules with default negated literals in their heads.

[3] *Dynamic Logic Programming* determines the semantics of sequences of generalized logic programs representing states of the world at different time periods, i.e. knowledge undergoing successive updates. As individual theories may comprise mutually contradictory as well as overlapping information, the role of *DLP* is to employ the mutual relationships among different states to determine the declarative semantics, at each state, for the combined theory comprised of all individual theories. Intuitively, one can add newer rules at the end of the sequence, leaving to *DLP* the task of ensuring that these rules are in force, and that previous ones are valid (by inertia) only so far as possible, i.e. they are kept for as long as they are not in conflict with newly added ones, these always prevailing.

2 Simple Agent Architecture

In this Section we define a very simple agent architecture with the purpose of facilitating the illustration of EVOLP as a language for knowledge representation and reasoning to be used in the context of Agent Oriented Programming. Accordingly, we will only be concerned with the mental part of the agent, namely the representation of beliefs, agent behaviour and epistemic effects of actions.

To this purpose, we assume that an agent's initial specification will be given by an EVOLP program P over some language $\mathcal{A}$, which will be used to concurrently represent the beliefs, behaviour and epistemic effects of actions. To simplify our presentation, we assume the existence of a set $\mathcal{A}_{\mathcal{C}} \subseteq \mathcal{A}$ of propositions, each representing an action that the agent is capable of preforming, and a set $\mathcal{A}_{do(\mathcal{C})} \subseteq \mathcal{A}$ such that $\mathcal{A}_{do(\mathcal{C})} = \{do(\alpha) : \alpha \in \mathcal{A}_{\mathcal{C}}\}$ and $\mathcal{A}_{do(\mathcal{C})} \cap \mathcal{A}_{\mathcal{C}} = \emptyset$. Propositions of the form $do(\alpha)$ will be used to indicate that some current belief state prescribes (or permits) the execution of action α, somehow encoding the agent's behaviour. Propositions of the form $\alpha \in \mathcal{A}_{\mathcal{C}}$ will indicate the actual execution of action α, possibly causing some update representing the epistemic effects of executing it. Furthermore we assume that at each state the agent perceives a set of events which, besides any observations from the environment or incoming messages, will also include a sub-set of $\mathcal{A}_{\mathcal{C}}$ representing the actions that were just performed, effectively allowing the agent to update its beliefs accordingly. Finally, we assume some selection function ($Sel()$) that, given a set of stable models (at some state), selects the set of actions to be executed[4]. Accordingly, at each state, and agent mental state is characterised by its initial specification and the sequence of events it has perceived so far.

Definition 1. *An agent state is a pair $\langle P, \mathcal{E}\rangle$ where P is an evolving logic program and $\mathcal{E}$ an event sequence, both over language $\mathcal{A}$.*

An agent evolves into the next state as per the following observe-think-act cycle and definition:

$cycle(\langle P, \mathcal{E}\rangle)$
  $observe$ (percieve E from inbox)
  $think$ (determine $SM(\langle P, (\mathcal{E}, E)\rangle)$)
  act (execute actions $Sel(SM(\langle P, (\mathcal{E}, E)\rangle))$)
  $cycle(\langle P, (\mathcal{E}, E)\rangle)$

---

3 Playing with Rules

In this section we illustrate how some of the features of EVOLP, many of which inherited from ASP, can be used to represent and reason about several aspects related to

[4] Throughout this paper, we assume that the selection function returns all actions belonging to one of the stable models, non-deterministically chosen from the set of stable models provided as its input. Other possible selection functions include returning the actions that belong to all stable models, those that belong to at least one stable model, or some more complex selection procedure e.g. based on priorities between actions/models.

Agents and Agent Oriented Programming. This illustration will be done incrementally through the elaboration of the specification of an agent whose purpose is to control the access to a building of some company with several floors. Some basic facts about this scenario include:

– the existence of 4 floors, numbered 0 through 3, represented by the following facts in P:

$$floor\,(0)\,.\quad floor\,(1)\,.\quad floor\,(2)\,.\quad floor\,(3)\,. \tag{1}$$

– the existence of 4 people, named Birna, John, Jamal and Matteo, represented by the following facts in P:

$$person\,(birna)\,.\quad person\,(john)\,.\quad person\,(jamal)\,.\quad person\,(matteo)\,. \tag{2}$$

– John, Jamal and Matteo are employees, represented by the following facts in P:

$$employee\,(john)\,.\quad employee\,(jamal)\,.\quad employee\,(matteo)\,. \tag{3}$$

– John is the company's director, represented by the following fact in P:

$$director\,(john)\,. \tag{4}$$

Deductive Reasoning. Rules in ASP provide an easy way to represent deductive knowledge. Suppose, for example, that according to the initial access policy of the building, any person who is employed by the company has permission to access any floor. This can be represented in ASP through the following deductive rule in P:

$$permit(P, F) \leftarrow person(P), floor(F), employee(P). \tag{5}$$

It is trivial to see that P has the following stable model (given an empty event sequence)[5]:

$$\begin{array}{l}\{f\,(0)\,, f\,(1)\,, f\,(2)\,, f\,(3)\,, p\,(birna)\,, p\,(john)\,, p\,(jamal)\,, p\,(matteo)\,,\\ e\,(john)\,, e\,(jamal)\,, e\,(matteo)\,, d\,(john)\,,\\ permit\,(0, john)\,, permit\,(1, john)\,, permit\,(2, john)\,, permit\,(3, john)\,,\\ permit\,(0, jamal)\,, permit\,(1, jamal)\,, permit\,(2, jamal)\,, permit\,(3, jamal)\,,\\ permit\,(0, matteo)\,, permit\,(1, matteo)\,, permit\,(2, matteo)\,, permit\,(3, matteo)\}\end{array}$$

It is interesting to note how rules in ASP can be seen as a query language (or a way to define views). If we interpret predicates as relations (e.g. $person/1$ as representing the relation $person$ with one attribute and four tuples: $birna$, $john$, $jamal$ and $matteo$) then, intuitively, the head of a rule represents the relation (view) obtained from the natural inner join of the relations in its body (when no negation is present).

[5] With the obvious abbreviations.

Reactive Behaviour. The same kind of deductive rules, when coupled with the previously presented agent cycle, can be used to express the reactive behaviour of an agent. For example, to express that any request to access some floor should result in the action of opening the corresponding door if the requester has a permit for that floor, we can include the following rule in P:

$$do\left(open\_door\left(F\right)\right) \leftarrow person\left(P\right), floor\left(F\right), permit\left(P,F\right), request\left(P,F\right). \tag{6}$$

If the agent perceives the event sequence $\mathcal{E} = (E_1, E_2)$ with:

$$\begin{array}{l} E_1 = \{request\left(matteo,3\right). \qquad request\left(birna,1\right)\} \\ E_2 = \{request\left(john,2\right).\} \end{array}$$

Then, $do\left(open\_door\left(3\right)\right)$ will belong to the only stable model at time 1, resulting in the execution of action $open\_door\left(3\right)$, while $do\left(open\_door\left(2\right)\right)$ will belong to the only stable model at time 2, resulting in the execution of action $open\_door\left(2\right)$. Note that $request\left(birna,1\right)$ will not cause the execution of any action since $permit\left(birna,1\right)$ is not true. Furthermore, since events do not persist by inertia (i.e. they are only used to determine the stable models at the state they were perceived and are ignored at subsequent states) each only triggers the execution of the actions once.

Effects of Actions. According to the simple agent architecture being used, we assume that the agent is aware of the actions it (actually) executes, inserting them at the subsequent set of events[6]. In the previous example, the action $open\_door\left(3\right)$ would actually belong to E_2, while action $open\_door\left(2\right)$ would belong to E_3, together with other observed events. We can take advantage of the inclusion of these actions in the events to express their epistemic effects, through ASP rules. For example, if we want to express that while we open the door, the door is open, we can include the following rule in P:

$$open\left(F\right) \leftarrow open\_door\left(F\right). \tag{7}$$

It is easy to see that $open\left(3\right)$ is true at state 2 while $open\left(2\right)$ is true at state 3.

Persistent Effects of Actions. In the previous example, the effect of $open\_door\left(F\right)$ does not persist i.e. $open\left(F\right)$ is only true while the action $open\_door\left(F\right)$ is true. Often we need to make the effects of actions persist. This can be done using the $assert$ predicate provided by EVOLP. If we want $open\_door\left(F\right)$ to cause the door to be open and stay open, we include the following rule in P:

$$assert\left(open\left(F\right)\right) \leftarrow open\_door\left(F\right). \tag{8}$$

[6] Since the actions actually executed depend on the selection function, the presence of $do\left(\alpha\right)$ in some (even every) model is not a guarantee that the action is going to be executed, e.g. because the selection function only selects a sub-set of those actions. This can also be used to filter those actions whose execution failed (e.g. a hardware error when moving the arm of the robot) by not inserting them in the events.

Now, $open\,(3)$ is not only true at state 2, but also at subsequent states. If we wish to close the door when someone enters the open door, provided there is no other action to open it, we can add the rule:

$$do\,(close\_door\,(F)) \leftarrow open\,(F)\,, enters\,(P, F)\,, not\, open\_door\,(F)\,. \tag{9}$$

together with the effects of closing the door:

$$assert\,(not\, open\,(F)) \leftarrow close\_door\,(F)\,. \tag{10}$$

Considering also these new rules, if $enters\,(matteo, 3)$ is observed at state 5, then $do\,(close\_door\,(3))$ also belongs to the stable model at that state, causing the action $close\_door\,(3)$ to be performed with the consequent update with $not\, open\,(3)$ at state 6, so that $open\,(3)$ will not be true from state 7, until another action opens the door again.

Default Reasoning. One of the main characteristics of ASP is its use of default negation (not) permitting, among other things, to non-monotonically reason with the Closed World Assumption. This is useful to represent exceptions. For example, if we wish to permit access to the ground floor to all people, except those on some black list, we can add the following rule to P:

$$permit(P, 0) \leftarrow person(P), not\, black\_list(P). \tag{11}$$

The default $not\, black\_list(P)$ evaluates to true whenever $black\_list(P)$ is not in the stable model. For example, if $request\,(birna, 0)$ belongs to some event at state n, then $do\,(open\_door\,(0))$ will be true at that state causing the door to be open, because $black\_list(birna)$ is not true (by default).

Open and Closed World Assumptions. ASP permits, besides default negation (not), strong negation ($\neg$). Strong negation behaves just as positive atoms i.e. in order for some $\neg A$ to evaluate to true, there must be some rule with head $\neg A$ and whose body is true, unlike $not\ A$ whose only requirement is that A is not true in the model[7]. For example, if we wish to implement an access policy where anyone not known to be a terrorist could access the first floor, we could add the following rule to P:

$$permit(P, 1) \leftarrow person(P), not\, terrorist(P). \tag{12}$$

With this rule, Birna, not known to be a terrorist in our example, would have permission to access the first floor i.e. $permit(birna, 1)$ is true. If we wish to impose a stricter policy for granting access to the second floor, where one is only allowed if (s)he is known to not be a terrorist, then we could add the following rule to P:

$$permit(P, 2) \leftarrow person(P), \neg terrorist(P). \tag{13}$$

[7] Recall that for a language $\mathcal{A}$ with one propositional atom A, there are three possible interpretations: $\{A\}\,, \{\neg A\}$ and $\{\}$, indicating, respectively, that A is true, A is (strongly) false, and A is neither true nor (strongly) false. $\neg A$ evaluates to true in the second interpretation while $not\ A$ evaluates to true in both the second and the third interpretations.

With this rule, Birna, would not have permission to access the second floor i.e. $permit\ (birna, 2)$ is false since $\neg terrorist(birna)$ is not true. in order to gain access to the second floor, the agent would have to know that $\neg terrorist(birna)$ is true (e.g. by conducting an investigation to clear Birna, leading to the assertion of $\neg terrorist(birna)$).

It is possible to turn strong negation into default negation. In our example, if we wish to represent that we should conclude $\neg terrorist(P)$ whenever $not\ terrorist(P)$ is true, we simply add the rule:

$$\neg terrorist(P) \leftarrow person(P), not\ terrorist(P). \tag{14}$$

This would turn the policies for both the first and second floors equivalent.

Non-deterministic Choice of Actions. In ASP, default negation, together with a special pair of rules known as *even loop through default negation*, allow for the generation of two (or more) stable models, which can be exploited to generate possible alternative behaviours. For example, if, for every person entering the ground floor, we wish to either perform a body search or simply ask for an id, we can add the following pair of rules to P:

$$\begin{array}{l} do\,(body\_search(P)) \leftarrow enters\,(P, 0)\,, not\ do\,(ask\_id(P))\,. \\ do\,(ask\_id(P)) \leftarrow enters\,(P, 0)\,, not\ do\,(body\_search(P))\,. \end{array} \tag{15}$$

If $enters\,(jamal, 0)$ is observed at some state, then, at that state, there will be two stable models:

$$\begin{array}{c} \{enters\,(jamal, 0)\,, do\,(body\_search(jamal))\,, ...\} \\ \{enters\,(jamal, 0)\,, do\,(ask\_id(jamal))\,, ...\} \end{array}$$

encoding both possible agent behaviours.

Uncertainty. The *even loop through default negation* and the generation of several models can also be used to represent uncertainty. For example, if we wish to state that everyone is either a friend or a terrorist, but we do not know which, we can add the following two rules to P:

$$\begin{array}{l} terrorist(P) \leftarrow person(P), not\ friend(P). \\ friend(P) \leftarrow person(P), not\ terrorist(P). \end{array} \tag{16}$$

These rules will generate many stable models, with all possible combinations where each person is either a friend or a terrorist. Some od these models include:

$$\begin{array}{c} \{friend(jamal), friend(birna), friend(matteo), friend(john), ...\} \\ \{friend(jamal), terrorist(birna), friend(matteo), friend(john), ...\} \\ \{friend(jamal), friend(birna), terrorist(matteo), terrorist(john), ...\} \\ \{terrorist(jamal), friend(birna), friend(matteo), friend(john), ...\} \\ ... \end{array}$$

Each of these models can be seen as a possible state of the world. Each of these worlds could prescribe some particular behaviour i.e. have some predicates of the form $do\,(\alpha)$

true them. However, this use of ASP would require a different selection function since different models would no longer encode different possible courses of action, as before, but rather different possible worlds in which different courses of action would be appropriate. Whereas in the former a selection function that chooses one model would be appropriate (as each can be seen as one possible course of action), in the latter the agent is unsure about which model represents the real world and simply flipping a coin to choose one of them may yield the incorrect choice (we may choose the second model above and treat Birna as a terrorist when in fact it turns out not to be the case). A selection function that returns the actions that are true in all models is probably a more appropriate choice, with natural consequences in the way knowledge is represented.

Integrity Constraints. When using the *even loop through default negation* to generate several models, we may want to eliminate some of them given the knowledge we have. This is easily done through the use of integrity constraints, which are rules of the form

$$\leftarrow L_1, \ldots, L_n. \tag{17}$$

and prevent any interpretation in which $L_1, \ldots, L_n$ is true to be a stable model[8]. For example, if we wish to state that John is not a terrorist, we simply add the following integrity constraint to P:

$$\leftarrow terrorist\,(john)\,. \tag{18}$$

This would eliminate the third model above, as well as any other models in which $terrorist\,(john)$ was true.

Uncertainty, as encoded through the even loop through default negation, can also be eliminated through the existence of factual or deductive knowledge. For example, if we know that all employees are friends, then we add the following rule to P:

$$friend\,(P) \leftarrow employee(P). \tag{19}$$

This will make $friend(jamal)$,$friend(matteo)$ and $friend(john)$ true, eliminating some of the uncertainty generated by rules (16). The remanning models would be:

$$\{friend(jamal),friend(birna),friend(matteo),friend(john),...\}$$
$$\{friend(jamal),terrorist(birna),friend(matteo),friend(john),...\}$$

Problem Solving. The combination of the *even loop through default negation* and integrity constraints has been successfully employed as a general technique to use ASP to solve many problems (e.g. hamiltonian cycles, graph coloring, large clique, planning, resource allocation, etc...). It is possible to use such ASP programs within this agent architecture, providing agents with the ability to solve complex problems through simple declarative specifications. Given that ASP can encode planning problems, it is possible to encode, without requiring changes in the agent architecture, the implementation of a planning agent where each model would encode a plan and the agent would select one of them for execution. More about using ASP for planning and problem solving in general can be found in [21,7,12].

[8] The semantics of an Integrity Constraint of the form $\leftarrow L_1, \ldots, L_n$.is given first translating it into the (normal) rule $a \leftarrow not\ a, L_1, \ldots, L_n.$, where a is a reserved atom.

Rule Updates. So far, with the exception of persistent effects of actions, we have only focused on features of ASP. We now turn our attention to those features introduced by EVOLP that allow for updating the agent specification. The semantics of EVOLP provides an easy mechanism to update the knowledge of some agent. This is due, on the one hand, to the fact that events do not persist by inertia and, on the other hand, that rules inside assert predicates that are true at some state are used to update the agent's knowledge. Accordingly, if we wish to update some agent's program with some rule r, all we have to do is include, in the events, the predicate $assert(r)$. If we further want to make such update conditional on the current state of events e.g. on some condition $L_1, \ldots, L_n$ being true, we add, instead, the rule $assert(r) \leftarrow L_1, \ldots, L_n$. In our scenario, imagine that it has been decided that, from now on, unless for the director(s), no one should have permission to access the third floor. This can be achieved by adding the fact to the next set of external events[9]:

$$assert\,(not\,permit(P,3) \leftarrow person(P), not\,director(P))\,. \tag{20}$$

Note that it is substantially different to update the agent and to simply add the rule inside the assert predicate to P. Whereas if we simply add this rule to P we obtain a contradiction with rule (5), if we update P with this rule such contradiction is automatically detected and avoided by assigning preference to the rule for the director (the newer one) than to the rule for employees (the original one)[10]. After this update, no one but John will be allowed on the third floor, and permissions regarding other floors will still be governed by the previously stated policy.

To illustrate an update conditioned on the current state of events, let's suppose that, later on, we want to grant permission to all current directors to access the third floor, even after they cease to be directors. This can be done by including the following rule in the events:

$$assert\,(permit(P,3)) \leftarrow director(P). \tag{21}$$

With this rule, the knowledge base will be updated with $permit(P,3)$ for all people (P) that are currently directors of the company, in this case, John. note however that the verification of $director(P)$ is only performed at the current state, and the update is done with $permit(john,3)$. When John ceases to be a director, he will still be allowed in the third floor, despite the effect of rule (20) which would, before this latter update, mean that John would not be allowed in the third floor.

[9] Some security mechanisms should be added to prevent anyone from being able to update some agent from the outside. This can also be expressed in EVOLP, although outside the scope of this presentation.

[10] The reader may be thinking that the same effect could be achieved by replacing the original rule with one where there was an exception for the director. Despite being true in this case, it is arguably simpler, on the one hand, to state what we want to be the case from now on (i.e. no one except for the director should access the third floor), and let the semantics of EVOLP deal with the resulting conflicts, while on the other hand coming up with such refined rules may be, for more elaborate cases, a very difficult task requiring knowledge of all rules in the agent program.

External Agent Programming. Rule updates, as exemplified above, be them simple assertions (updates) of rules, or updates conditioned on the current state of affairs, provide a flexible mechanism to incrementally program agents after they have been deployed. All aspects of the agent that are programmed in EVOLP can be incrementally specified, and the agent specification completely changed as the agent progresses. Not only can we change the knowledge of the agent, but we can also update the agent behaviour (providing new, better, ways to react), the effects of the actions (even allowing the agent to learn new actions e.g. a robot was serviced with a new arm), and any other aspect specified in EVOLP. For example, we can update the behaviour of the agent exhibited when someone enters the ground floor (specified by rules 15) in a way such that directors are neither body searched nor asked for an id, by updating with the rules:

$$\begin{array}{l} not\, do\,(body\_search(P)) \leftarrow enters\,(P,0)\,, director(P). \\ not\, do\,(ask\_id(P)) \leftarrow enters\,(P,0)\,, director(P). \end{array} \tag{22}$$

We can update the effects of action $open\_door$ so that it does not open in case of malfunction, by updating with the rule (in the case where it's effect is persistent):

$$not\, assert\,(open\,(F)) \leftarrow open\_door\,(F)\,, malfunction(F). \tag{23}$$

Evolution. Assert predicates can be present in the agent program, specifying the way in which the agent should evolve (its knowledge, behaviour, etc...). Furthermore, the fact that assertions can be nested allows for expressing more complex evolutions. Imagine that we decide to implement a policy according to which, after the agent observes a (terrorist) attack, it starts behaving in a way that, after asking someone for an ID, it will treat that person as a terrorist if the person does not have an ID. Recall that $ask\_id(P)$ is an action (hence non-inertial), and $attack$ is an observation. this policy could be implemented with the following rule:

$$assert\,(assert\,(terrorist\,(P) \leftarrow not\, id(P)) \leftarrow ask\_id(P)) \leftarrow attack. \tag{24}$$

In the event of an attack (i.e. $attack$ belongs to some set of events), then the agent evolves to a new specification resulting from the update with the rule

$$assert\,(terrorist\,(P) \leftarrow not\, id(P)) \leftarrow ask\_id(P). \tag{25}$$

If, for example, Birna is asked for an ID, e.g. due to the non-deterministic choice of actions specified in rules (15), the agent is updated with the rule

$$terrorist\,(birna) \leftarrow not\, id(birna). \tag{26}$$

and, from then on, until revoked by some other update, Birna will be considered a terrorist if she doesn't have an ID.

Complex Behaviour. The possibility provided by EVOLP to include assert atoms in the body of rules, as well as actions, allows for the specification of more complex behaviours in a simple way. For example, if we wish to specify that some action should be

followed by some other action (e.g. that a body search should be reported), we simply include the rule:

$$do\left(report\left(P\right)\right) \leftarrow body\_search(P). \tag{27}$$

To specify that some action should always be performed together with some other action (e.g. arresting someone should be done together with informing the person of his rights), can be done with the rule:

$$do\left(read\_rights\left(P\right)\right) \leftarrow do\left(arrest\left(P\right)\right). \tag{28}$$

The ability to include assertive literals in rule bodies allows for looking ahead on some program changes and acting on that knowledge before the changes occur. For example, to inform someone that (s)he will be added to the company's black list, we use the rule:

$$do\left(inform\left(P\right)\right) \leftarrow assert\left(black\_list\left(P\right)\right). \tag{29}$$

It could be important to have a timer to allow for updates to depend on it. Such timer can be encoded in EVOLP with the rules

$$\begin{array}{l} assert\left(time\left(T+1\right)\right) \leftarrow time\left(T\right). \\ assert\left(not\, time\left(T\right)\right) \leftarrow time\left(T\right). \end{array} \tag{30}$$

together with the fact $time\left(0\right)$ in P.

With this clock, we can *easily* encode effects of actions (or reactive behaviours) that span over time, e.g. delayed effect of actions or behaviours, i.e. when the effect of some action (or observation) only occurs after a certain amount of time. For example, if we wish to close the door, ten time steps after it is opened (even if no one enters it as specified in rule (9)), we can add the rule:

$$assert\left(do\left(close\_door\left(F\right)\right) \leftarrow time\left(T+10\right)\right) \leftarrow open\_door\left(F\right), time\left(T\right). \tag{31}$$

Complex Effects of Actions. Many Agent Oriented Programming Languages are very limited in what concerns representing the effects of actions. It was shown that Logic Programming Update Languages, including EVOLP, are able to capture Action Languages $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, making them suitable for describing domains of actions [1]. This includes, for example, representing effects of executing parallel actions, non-deterministic effects of actions, etc.

Communication. Messages can be treated in a straight forward manner. On the one hand, sending a message is treated just as any other action i.e. through some predicate of the form $do\left(send\_msg\left(To, Type, Content\right)\right)$, possibly with some internal effect (e.g. recording the sent message). Similarly, incoming messages are just events (observations) of some agreed form, e.g. a predicate of the form $msg\left(From, Type, Content\right)$, and can be treated by the agent just as any other event. For example, to store (with a timestamp) all incoming messages sent by employees, the following rule could be used:

$$assert\left(msg\left(F, Ty, C, T\right)\right) \leftarrow msg\left(F, Ty, C\right), time\left(T\right), employee\left(F\right). \tag{32}$$

4 Conclusions, Current and Future Work

In this paper we revisited Logic Programming under the answer-set semantics - or *Answer-Set Programming* - and its extension *Evolving Logic Programming*, two languages that use logic rules and rule updates, and exhibit characteristics that make them suitable to be used for knowledge representation and reasoning within Agent Oriented Programming Languages. We illustrated the power of these rule based languages by means of examples showing how several of its features can be used to model situations faced by Agents.

The use of EVOLP has previously been illustrated in Role Playing Games to represent the dynamic behaviour of Non-Playing Characters [18] and Knowledge Bases to describe their update policies [13]. Furthermore, it was shown that Logic Programming Update Languages are able to capture Action Languages $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, making them suitable for describing domains of actions [1].

Even though we only presented EVOLP with its stable model based semantics, which means that it inherits the complexity of ASP i.e. it is in the NP class when only non-disjunctive programs are used, EVOLP also allows for the use of a well founded three valued semantics which is less expressive but permits more efficient top down polynomial proof procedures [27,6].

Extensions of EVOLP include the introduction of LTL-like temporal operators that allow more flexibility in referring to the history of the evolving knowledge base [4].

It was not the purpose of this paper to present yet another agent architecture and language to populate the already dense field of AOPLs. Instead, the purpose was to present several features of a well known language (ASP) and a more recent extension (EVOLP) in a way that brings forward the possible advantages of their usage within existing AOPLs. This is precisely the subject of ongoing research where ASP and rule updates are being used in conjunction with the AOPL GOAL [16].

References

1. Alferes, J.J., Banti, F., Brogi, A.: From logic programs updates to action description updates. In: Leite, J., Torroni, P. (eds.) CLIMA 2004. LNCS (LNAI), vol. 3487, pp. 52–77. Springer, Heidelberg (2005)
2. Alferes, J.J., Banti, F., Brogi, A., Leite, J.A.: The refined extension principle for semantics of dynamic logic programming. Studia Logica 79(1), 7–32 (2005)
3. Alferes, J.J., Brogi, A., Leite, J.A., Pereira, L.M.: Evolving logic programs. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, pp. 50–61. Springer, Heidelberg (2002)
4. Alferes, J.J., Gabaldon, A., Leite, J.: Evolving logic programming based agents with temporal operators. In: IAT, pp. 238–244. IEEE, Los Alamitos (2008)
5. Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., Przymusinski, T.: Dynamic updates of non-monotonic knowledge bases. Journal of Logic Programming 45(1-3), 43–70 (2000)
6. Banti, F., Alferes, J.J., Brogi, A.: Well founded semantics for logic program updates. In: Lemaître, C., Reyes, C.A., González, J.A. (eds.) IBERAMIA 2004. LNCS (LNAI), vol. 3315, pp. 397–407. Springer, Heidelberg (2004)

7. Baral, C.: Knowledge Representation, Reasoning, and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
8. Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E. (eds.): Multi-Agent Programming: Languages, Platforms and Applications. Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 15. Springer, Heidelberg (2005)
9. Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E. (eds.): Multi-Agent Programming: Languages, Tools and Applications. Springer, Heidelberg (2009)
10. Brewka, G.: Nonmonotonic Reasoning: Logical Foundations of Commonsense. Cambridge University Press, Cambridge (1991)
11. Brooks, D.R., Erdem, E., Erdogan, S.T., Minett, J.W., Ringe, D.: Inferring phylogenetic trees using answer set programming. J. Autom. Reasoning 39(4), 471–511 (2007)
12. Garcia de la Banda, M., Pontelli, E. (eds.): ICLP 2008. LNCS, vol. 5366. Springer, Heidelberg (2008)
13. Eiter, T., Fink, M., Sabbatini, G., Tompits, H.: A framework for declarative update specifications in logic programs. In: Nebel, B. (ed.) IJCAI, pp. 649–654. Morgan Kaufmann, San Francisco (2001)
14. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive datalog. ACM Trans. Database Syst. 22(3), 364–418 (1997)
15. Gelfond, M., Lifschitz, V.: Logic programs with classical negation. In: Warren, D., Szeredi, P. (eds.) Proceedings of the 7th international conference on logic programming, pp. 579–597. MIT Press, Cambridge (1990)
16. Hindriks, K.V.: Programming rational agents in goal. In: Bordini, et al. (eds.) [9], ch. 3
17. Leite, J., Alferes, J.J., Mito, B.: Resource allocation with answer-set programming. In: Sierra, C., Castelfranchi, C., Decker, K.S., Sichman, J.S. (eds.) AAMAS (1), IFAAMAS, pp. 649–656 (2009)
18. Leite, J., Soares, L.: Evolving characters in role playing games. In: Trappl, R. (ed.) Cybernetics and Systems, Proceedings of the 18th European Meeting on Cybernetics and Systems Research (EMCSR 2006), vol. 2, pp. 515–520. Austrian Society for Cybernetic Studies (2006)
19. Leite, J.A.: Evolving Knowledge Bases. IOS Press, Amsterdam (2003)
20. Leone, N., Greco, G., Ianni, G., Lio, V., Terracina, G., Eiter, T., Faber, W., Fink, M., Gottlob, G., Rosati, R., Lembo, D., Lenzerini, M., Ruzzi, M., Kalka, E., Nowicki, B., Staniszkis, W.: The infomix system for advanced integration of incomplete and inconsistent data. In: Özcan, F. (ed.) SIGMOD Conference, pp. 915–917. ACM, New York (2005)
21. Lifschitz, V.: Answer set programming and plan generation. Artificial Intelligence 138(1-2), 39–54 (2002)
22. Lifschitz, V., Woo, T.Y.C.: Answer sets in general non-monotonic reasoning (preliminary report). In: Nebel, B., Rich, C., Swartout, W. (eds.) Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR 1992), pp. 603–614. Morgan Kaufmann, San Francisco (1992)
23. Makinson, D.: Bridges from Classical to Nonmonotonic Logic. College Publications (2005)
24. Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., Barry, M.: An a-prolog decision support system for the space shuttle. In: Ramakrishnan, I.V. (ed.) PADL 2001. LNCS, vol. 1990, pp. 169–183. Springer, Heidelberg (2001)
25. Slota, M., Leite, J.: Evolp: An implementation. In: Sadri, F., Satoh, K. (eds.) CLIMA VIII 2007. LNCS (LNAI), vol. 5056, pp. 288–298. Springer, Heidelberg (2008)
26. Tiihonen, J., Soininen, T., Niemelä, I., Sulonen, R.: A practical tool for mass-customising configurable products. In: Proceedings of the 14th International Conference on Engineering Design, pp. 1290–1299 (2003)
27. van Gelder, A., Ross, K.A., Schlipf, J.S.: Unfounded sets and well-founded semantics for general logic programs. Journal of the ACM 38(3), 620–650 (1991)

A Evolving Logic Programming

A.1 Language

We start with the usual preliminaries. Let $\mathcal{A}$ be a set of propositional atoms. An objective literal is either an atom A or a strongly negated atom $\neg A$. A default literal is an objective literal preceded by not. A literal is either an objective literal or a default literal. A rule r is an ordered pair $H(r) \leftarrow B(r)$ where $H(r)$ (dubbed the head of the rule) is a literal and $B(r)$ (dubbed the body of the rule) is a finite set of literals. A rule with $H(r) = L_0$ and $B(r) = \{L_1, \ldots, L_n\}$ will simply be written as $L_0 \leftarrow L_1, \ldots, L_n$. A generalized logic program (*GLP*) P, in $\mathcal{A}$, is a finite or infinite set of rules. If $H(r) = A$ (resp. $H(r) = not\ A$) then $not\ H(r) = not\ A$ (resp. $not\ H(r) = A$). If $H(r) = \neg A$, then $\neg H(r) = A$. By the expanded generalized logic program corresponding to the GLP P, denoted by $\mathbf{P}$, we mean the GLP obtained by augmenting P with a rule of the form $not\ \neg H(r) \leftarrow B(r)$ for every rule, in P, of the form $H(r) \leftarrow B(r)$, where $H(r)$ is an objective literal. Two rules r and r' are conflicting, denoted by $r \bowtie r'$, iff $H(r) = not\ H(r')$.

An interpretation M of $\mathcal{A}$ is a set of objective literals that is consistent i.e., M does not contain both A and $\neg A$. An objective literal L is true in M, denoted by $M \vDash L$, iff $L \in M$, and false otherwise. A default literal $not\ L$ is true in M, denoted by $M \vDash not\ L$, iff $L \notin M$, and false otherwise. A set of literals B is true in M, denoted by $M \vDash B$, iff each literal in B is true in M.

An interpretation M of $\mathcal{A}$ is an answer set of a GLP P iff

$$M' = least\left(\mathbf{P} \cup \{not\ A \mid A \notin M\}\right) \tag{33}$$

where $M' = M \cup \{not\_A \mid A \notin M\}$, A is an objective literal, and $least(.)$ denotes the least model of the definite program obtained from the argument program by replacing every default literal $not\ A$ by a new atom not_A.

In order to allow for logic programs to evolve, we first need some mechanism for letting older rules be supervened by more recent ones. That is, we must include a mechanism for deletion of previous knowledge along the agent's knowledge evolution. This can be achieved by permitting default negation not just in rule bodies, as in extended logic programming, but in rule heads as well[22]. Furthermore, we need a way to state that, under some conditions, some new rule should be asserted in the knowledge base[11]. In EVOLP this is achieved by augmenting the language with a reserved predicate $assert/1$, whose sole argument is itself a full-blown rule, so that arbitrary nesting becomes possible. This predicate can appear both as rule head (to impose internal assertions of rules) as well as in rule bodies (to test for assertion of rules). Formally:

Definition 2. *Let $\mathcal{A}$ be a set of propositional atoms (not containing $assert/1$). The* extended language $\mathcal{A}_{assert}$ *is defined inductively as follows:*

[11] Note that asserting a rule in a knowledge base does not mean that the rule is simply added to it, but rather that the rule is used to update the existing knowledge base according to some update semantics, as will be seen below.

- *All propositional atoms in $\mathcal{A}$ are propositional atoms in $\mathcal{A}_{assert}$;*
- *If r is a rule over $\mathcal{A}_{assert}$ then $assert(r)$ is a propositional atom of $\mathcal{A}_{assert}$;*
- *Nothing else is a propositional atom in $\mathcal{A}_{assert}$.*

An evolving logic program *over a language $\mathcal{A}$ is a (possibly infinite) set of generalized logic program rules over $\mathcal{A}_{assert}$.*

Example 1. Examples of EVOLP rules are:

$$\begin{array}{l} assert\left(not\, a \leftarrow b\right) \leftarrow not\, c. \\ a \leftarrow assert\left(b \leftarrow\right). \\ assert\left(assert\left(a \leftarrow\right) \leftarrow assert\left(b \leftarrow not\, c\right), d\right) \leftarrow not\, e. \end{array} \tag{34}$$

Intuitively, the first rule states that, if c is false, then the rule $not\, a \leftarrow b$ must be asserted in the agent's knowledge base; the 2nd that, if the fact $b \leftarrow$ is going to be asserted in the agent's knowledge base, then a is true; the last states that, if e is false, then a rule must be asserted stating that, if d is true and the rule $b \leftarrow not\, c$ is going to be asserted then the fact $a \leftarrow$ must be asserted.

This language alone is enough to model the agent's knowledge base, and to cater, within it, for internal updating actions that change it. But self-evolution of a knowledge base is not enough for our purposes. We also want the agent to be aware of events that happen outside itself, and desire the possibility too of giving the agent update "commands" for changing its specification. In other words, we wish a language that allows for influence from the outside, where this influence may be: observation of facts (or rules) that are perceived at some state; assertion commands directly imparting the assertion of new rules on the evolving program. Both can be represented as EVOLP rules: the former by rules without the assert predicate in the head, and the latter by rules with it. Consequently, we shall represent outside influence as a sequence of EVOLP rules:

Definition 3. *Let P be an evolving program over the language $\mathcal{A}$. An* event sequence *over P is a sequence of evolving programs over $\mathcal{A}$.*

A.2 Semantics

In general, we have an EVOLP program describing an agent's initial knowledge base. This knowledge base may already contain rules (with asserts in heads) that describe some forms of its own evolution. Besides this, we consider sequences of events representing observation and messages arising from the environment. Each of these events in the sequence are themselves sets of EVOLP rules, i.e. EVOLP programs. The semantics issue is thus that of, given an initial EVOLP program and a sequence of EVOLP programs as events, to determine what is true and what is false after each of those events.

More precisely, the meaning of a sequence of EVOLP programs is given by a set of *evolution stable models*, each of which is a sequence of interpretations or states. The basic idea is that each evolution stable model describes some possible evolution of one initial program after a given number n of evolution steps, given the events in the sequence. Each evolution is represented by a sequence of programs, each program corresponding to a knowledge state.

The primordial intuitions for the construction of these program sequences are as follows: regarding head asserts, whenever the atom $assert(Rule)$ belongs to an interpretation in a sequence, i.e. belongs to a model according to the stable model semantics of the current program, then $Rule$ must belong to the program in the next state; asserts in bodies are treated as any other predicate literals.

The sequences of programs are treated as in Dynamic Logic Programming [19,5,2], a framework for specifying updates of logic programs where knowledge is given by a sequence of logic programs whose semantics is based on the fact that the most recent rules are set in force, and previous rules are valid (by inertia) insofar as possible, i.e. they are kept for as long as they do not conflict with more recent ones. In DLP, default negation is treated as in answer-set programming [15]. Formally, a *dynamic logic program* is a sequence $\mathcal{P} = (P_1, \ldots, P_n)$ of generalized logic programs and its semantic is determined by (c.f. [19,2] for more details):

Definition 4. *Let $\mathcal{P} = (P_1, \ldots, P_n)$ be a dynamic logic program over language $\mathcal{A}$. An interpretation M is a (refined) dynamic stable model of $\mathcal{P}$ at state s, $1 \leq s \leq n$ iff*

$$M' = least\left(\left[\rho_s\left(\mathcal{P}\right) - Rej_s(M)\right] \cup Def_s(M)\right) \tag{35}$$

where:

$$\begin{array}{l} Def_s(M) = \{not\ A \mid \nexists r \in \rho(\mathcal{P}), H(r) = A, M \vDash B(r)\} \\ Rej_s(M) = \{r \mid r \in \mathbf{P}_i, \exists r' \in \mathbf{P}_j, i \leq j \leq s, r \bowtie r', M \vDash B(r')\} \end{array} \tag{36}$$

and A is an objective literal, $\rho_s\left(\mathcal{P}\right)$ denotes the multiset of all rules appearing in the programs $\mathbf{P}_1, ..., \mathbf{P}_s$, and M' and $least(.)$ are as before. Let $DSM\left(\mathcal{P}\right)$ denote the set of (refined) dynamic stable model of $\mathcal{P}$ at state n.

Intuitively, given an interpretation M, the set $Rej_s(M)$ contains those rules which are overridden by a newer conflicting rule whose body is true according to the interpretation M. The set $Def_s(M)$ contains default negations $not\ A$ of all unsupported atoms A, i.e., those atoms A for which there is no rule, in any program, whose body is true according to the interpretation M, which can thus be assumed false *by default*.

Before presenting the definitions that formalize the above intuitions of EVOLP, let us show some illustrative examples.

Example 2. Consider an initial program P containing the rules

$$\begin{array}{l} a. \\ assert(not\ a \leftarrow) \leftarrow b. \\ c \leftarrow assert(not\ a \leftarrow). \\ assert(b \leftarrow a) \leftarrow not\ c. \end{array} \tag{37}$$

and that all the events are empty EVOLP programs. The (only) answer set of P is $M = \{a, assert(b \leftarrow a)\}$ and conveying the information that program P is ready to evolve into a new program (P, P_2) by adding rule $(b \leftarrow a)$ at the next step, i.e. to P_2. In the only dynamic stable model M_2 of the new program (P, P_2), atom b is true as well as atom $assert(not\ a \leftarrow)$ and also c, meaning that (P, P_2) evolves into a new program (P, P_2, P_3) by adding rule $(not\ a \leftarrow)$ at the next step, i.e. in P_3. This negative fact in P_3

conflicts with the fact in P, and the older is rejected. The rule added in P_2 remains valid, but is no longer useful to conclude b, since a is no longer valid. So, $assert(not\ a \leftarrow)$ and c are also no longer true. In the only dynamic stable model of the last sequence both a, b, and c are false.

This example does not address external events. The rules that belong to the i-th event should be added to the program of state i, and proceed as in the example above.

Example 3. In the example above, suppose that at state 2 there is an external event with the rules, r_1 and r_2, $assert(d \leftarrow b) \leftarrow a$ and $e \leftarrow$. Since the only stable model of P is $I = \{a, assert(b \leftarrow a)\}$ and there is an outside event at state 2 with r_1 and r_2, the program evolves into the new program obtained by updating P not only with the rule $b \leftarrow a$ but also with those rules, i.e. $(P, \{b \leftarrow a;\ assert(d \leftarrow b) \leftarrow a;\ e \leftarrow\})$. The only dynamic stable model M_2 of this program is $\{b, assert(not\ a \leftarrow), assert(d \leftarrow b), e\}$.

If we keep with the evolution of this program (e.g. by subsequent empty events), we have to decide what to do, in these subsequent states, about the event received at state 2. Intuitively, we want the rules coming from the outside, be they observations or assertion commands, to be understood as events given at a state, that are not to persist by inertia. I.e. if rule r belongs to some set E_i of an event sequence, this means that r was perceived, or received, after $i-1$ evolution steps of the program, and that this perception event is not to be assumed by inertia from then onward. In the example, it means that if we have perceived e at state 2, then e and all its possible consequences should be true at that state. But the truth of e should not persist into the subsequent state (unless e is yet again perceived from the outside). In other words, when constructing subsequent states, the rules coming from events in state 2 should no longer be available and considered. As will become clear below, making these events persistent can be specified in EVOLP.

Definition 5. *An* evolution interpretation *of length n of an evolving program P over $\mathcal{A}$ is a finite sequence $\mathcal{I} = (I_1, I_2, \ldots, I_n)$ of interpretations $\mathcal{A}_{assert}$. The* evolution trace *associated with evolution interpretation $\mathcal{I}$ is the sequence of programs $(P_1, P_2, \ldots, P_n)$ where:*

- $P_1 = P$;
- $P_i = \{r \mid assert(r) \in I_{i-1}\}$, *for each* $2 \leq i \leq n$.

Definition 6. *An evolution interpretation $(I_1, I_2, \ldots, I_n)$, of length n, with evolution trace $(P_1, P_2, \ldots, P_n)$ is an* evolution stable model *of an evolving program P given a sequence of events $(E_1, E_2, \ldots, E_k)$, with $n \leq k$, iff for every i ($1 \leq i \leq n$), I_i is a dynamic stable model at state i of $(P_1, P_2 \ldots, (P_i \cup E_i))$.*

Notice that the rules coming from the outside do not persist by inertia. At any given step i, the rules from E_i are added and the (possibly various) I_i obtained. This determines the programs P_{i+1} of the trace, which are then added to E_{i+1} to determine the models I_{i+1}. The definition assumes the whole sequence of events given a priori. In fact this need not be so because the events at any given step n only influence the models in the evolution interpretation from n onward:

Proposition 1. *Let $M = (M_1, \ldots, M_n)$ be an evolution stable model of P given a sequence of events $(E_1, E_2, \ldots, E_n)$. Then, for any sets of events $E_{n+1}, \ldots, E_m$ $(m > n)$, M is also an evolution stable model of P given $(E_1, \ldots, E_n, E_{n+1}, \ldots, E_m)$.*

EVOLP programs may have various evolution models of given length, or none:

Example 4. Consider P with the following two rules, and 3 empty events:

$$\begin{array}{l} assert\,(a \leftarrow) \leftarrow not\, assert\,(b \leftarrow), not\, b. \\ assert\,(b \leftarrow) \leftarrow not\, assert\,(a \leftarrow), not\, a. \end{array} \tag{38}$$

The reader can check that there are 2 evolution stable models of length 3, each representing one possible evolution of the program after those empty events:

$$\begin{array}{l} M_1 = \langle \{assert(a \leftarrow)\}, \{a, assert(a \leftarrow)\}, \{a, assert(a \leftarrow)\} \rangle \\ M_2 = \langle \{assert(b \leftarrow)\}, \{b, assert(b \leftarrow)\}, \{b, assert(b \leftarrow)\} \rangle \end{array}$$

Since various evolutions may exist for a given length, evolution stable models alone do not determine a truth relation. A truth relation can be defined, as usual, based on the intersection of models:

Definition 7. *Let P be an evolving program, $\mathcal{E}$ an event sequence of length n, both over the language $\mathcal{A}$, and M an interpretation over $\mathcal{A}_{assert}$. M is a* Stable Model of P given $\mathcal{E}$ *iff $(M_1, \ldots, M_{n-1}, M)$ is an evolution stable model of P given $\mathcal{E}$ with length n, for some interpretations $M_1, \ldots, M_{n-1}$. Let $SM\,(\langle P, \mathcal{E} \rangle)$ denote the set of* Stable Model of P given $\mathcal{E}$*.We say that propositional atom A of $\mathcal{A}$ is:* true given $\mathcal{E}$*, denoted by $\langle P, \mathcal{E} \rangle \models A$, iff A belongs to all stable models of P given $\mathcal{E}$;* false given $\mathcal{E}$*, denoted by $\langle P, \mathcal{E} \rangle \models not\, A$, iff A does not belong to any stable models of P given $\mathcal{E}$;* unknown given $\mathcal{E}$ *otherwise.*

A consequence of the above definitions is that the semantics of EVOLP is, in fact, a proper generalization of the answer-set semantics, in the following sense:

Proposition 2. *Let P be a generalized (extended) logic program (without predicate $assert/1$) over a language $\mathcal{A}$, and $\mathcal{E}$ be any sequence with $n \geq 0$ of empty EVOLP programs. Then, M is a stable model of P given $\mathcal{E}$ iff the restriction of M to $\mathcal{A}$ is an answer set of P (in the sense of [15, 22]).*

The possibility of having various stable models after an event sequence is of special interest for using EVOLP as a language for reasoning about possible evolutions of an agent's knowledge base. Like for answer-set programs, we define the notion of categorical programs as those such that, for any given event sequence, no "branching" occurs, i.e. a single stable model exists.

Definition 8. *An EVOLP program P is* categorical *given event sequence $\mathcal{E}$ iff there exists only one stable model of P given $\mathcal{E}$.*

The Refinement of Choreographed Multi-Agent Systems

Lăcrămioara Aştefănoaei[1], Frank S. de Boer[1], and Mehdi Dastani[2]

[1] CWI, Amsterdam, The Netherlands
[2] Universiteit Utrecht, The Netherlands

Abstract. This paper generalises the theory of agent refinement from [1] to multi-agent systems in the presence of new coordination mechanisms extended with real time. The generalisation is such that refinement is compositional. This means that refinement at the individual level implies refinement at the multi-agent system level. Compositionality is an important property since it reduces heavily the verification process. Thus having a theory of refinement is a crucial step towards the verification of multi-agent systems' correctness.

1 Introduction

In [1] a general framework is proposed for designing and verifying agent languages. The authors promoted the idea of having different levels of abstractions for agents, emphasising the principles of UNITY [2], a classical design methodology. Such principles state that abstract models should disregard control issues and should specify as little as possible. Control should be added at later phases of design. In the same vein, BUnity and BUpL were proposed as two modelling agent languages in [1], each corresponding to a distinct level of abstraction. BUnity was considered as being a specification language and BUpL as an implementation. The authors then posed the problem of the correctness of a given BUpL implementation with respect to a given BUnity specification. The problem was equally stated as a refinement problem, where refinement was defined in terms of trace inclusion. Traces were sequences of observable actions considered to represent agents' behaviours. Being that a direct approach to deciding trace inclusion is hard, they adapted simulation as a proof technique to agent refinement. Their approach considers only individual agents.

In this paper, we aim at extending the refinement relation to multi-agent systems. A first step consists of lifting the notion of abstraction levels from individual agents to multi-agent systems. Considering that the behaviour of the multi-agent system is simply the sum of the behaviours of individual agents is a too unrealistic idea. Instead, we propose *action-based coordination* mechanisms, to which we refer as *choreographies*. They represent global synchronisation and ordering conditions restricting the execution of individual agents.

Introducing coordination while respecting the autonomy of the agents is still a challenge in the design of multi-agent systems. We note that choreographies might constrain agents' autonomy, however, this is a very common practice in multi-agent systems when specific properties need to be guaranteed. The advantage of the infrastructures we propose lies in their *exogenous* feature: the update of the agent's mental states is separated from the coordination pattern. Nobody changes the agent's beliefs but itself.

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 20–34, 2010.

Besides that choreographies are oblivious to mental aspects, they control without having to know the internal structure of the agent. For example, whenever a choice between plans needs to be taken, a BUpL agent is free to make its own decision. The degree of freedom can be seen also in the mechanism for handling action failures. The agent chooses one among possibly many available repair rules without being constraint by the choreography. In these regards, the autonomy of agents is preserved.

Extending the refinement relation from [1] to multi-agent systems requires solving a new problem since choreographies may introduce deadlocks. It can be the case that though there is refinement at the individual agent level, adding a choreography deadlocks the concrete multi-agent system but not the abstract one. We take, as example, a choreography which "tells" an agent to execute an action not defined in the agent program itself (but only in the agent specification). In this situation, refinement as trace inclusion trivially holds since the set of traces from a deadlocked state (where the agent, after eventually applying repair rules, cannot execute the action specified by the choreography) is empty.

Our methodology in approaching the above stated problem consists of, basically, formalising the following aspects. On the one hand, we define the semantics of multi-agent systems with choreographies as the set of *maximal traces*, where we make the distinction between a *success* and a *deadlock*. These traces consist of the parallel agents' executions guided by the choreography. We define multi-agent system refinement as maximal trace inclusion. On the other hand, agent refinement becomes *ready trace* inclusion, where a ready trace records not only the actions being executed, but also those ones which *might* be executed. We show that multi-agent system refinement is *compositional*. More precisely, the main result of this paper is that agent refinement implies multi-agent system refinement in the presence of *any* choreography. Furthermore, the refined multi-agent system does not introduce deadlocks with respect to the multi-agent system specification.

A more expressive framework can be obtained when action synchronisations depend also on time, not only on the disposal of the agents to perform the actions. Thus, we address the problem of incorporating time into choreographies such that the compositionality result we have remains valid in the timed version. A first step is to extend choreographies by means of timed automata [3] such that they constrain the *timings* of the actions. Having *timed choreographies* requires, however, to introduce time in the agents' programs. Thus, in our case, BUnity and BUpL need to be extended to reflect the passing of time. In this respect, we bare in mind that basic actions are a *common ontology* shared by all agents. Since the nature of basic actions does not specify *when* to be executed, our extension is thought such that the ontology remains timeless and "when" becomes part of the specific agent applications.

Our contribution consists of introducing a general methodology for a top-down design of multi-agent systems by refinement. We emphasise that our the effort is motivated by the need to perform verification. Multi-agent systems are clearly more complex structures, and their verification tends to become harder. However, in our framework, given the compositionality result, it is enough to verify individual agents and/or choreographies in order to conclude properties about the whole multi-agent system.

The paper is structured as follows. Section 2.1 is a brief overview of agent refinement. Section 3 introduces the notion of choreographies and describes multi-agent system refinement. Section 4 discusses time extensions for multi-agent systems. Section 5 concludes the paper.

1.1 Related Works

The design methodology we propose integrates in a unifying approach different concepts and results from process theory [4]. Some aspects we deal with have been taken into account in different works, however, from a distinct angle.

Multi-agent system verification is discussed in [5,6,7]. However, we focus on the compositionality of the refinement relation which reduces the problem of verifying the whole multi-agent system to verifying the agents composing it.

Concerning agent interaction, this is usually achieved by means of communication. Communication, in turn, is implemented by message passing, or channel-based mechanisms. This latter can be seen as the basis of implementing coordination artifacts in terms of resource access relation in [8], or action synchronisation [9]. We also mention the language Linda [10] which has not been applied to a multi-agent setting but to service oriented services, where the notion of *data* plays a more important role than *synchrony*. Organisation-based coordination artifacts are recently discussed in [11,12,13].

The concepts of choreography and orchestration have already been introduced to web services in [14,15]. With respect to [16], though we use the same terminology, our framework is in essence different since we deliberately ignore communication issues. Our choreography model is explicit whereas in [16] is implicit in the communication protocol. Thus, we need to take into account deadlocks that may appear because of "mall-formed" choreographies. Being external, the choreography represents, in fact, contexts while in the other approaches there is a distinction between the modularity and the contextuality of the communication operator.

Considering timed automata, its application in a multi-agent system is new. However, timed automata has already been applied to testing real-time systems specifications [17] or to scheduling problems [18].

2 Preliminaries

A *labelled transition system* (LTS) is a tuple $(\Sigma, s_0, Act, \rightarrow, F)$, where Σ is a set of states, s_0 is an initial state, Act is a set of actions (labels), $\rightarrow$ describes all possible transitions and F is a set of final states. The notation $s \xrightarrow{a} s'$ means that "s becomes s' by performing action a". Invisible actions are denoted by τ steps. The "weak" arrow $\Rightarrow$ denotes the reflexive and transitive closure of $\rightarrow$, and $\stackrel{a}{\Rightarrow}$ stands for $\Rightarrow\xrightarrow{a}\Rightarrow$. A *trace* $\sigma(s)$ is a sequence of actions $a_1, a_2, ...$ such that $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2$.... The set of all traces from s is $T(s)$.

2.1 Recalling Agent Refinement

The configuration of a BUnity agent is a tuple $\langle \mathcal{B}_0, \mathcal{A}, \mathcal{C} \rangle$, where $\mathcal{B}_0$ is an initial belief base, $\mathcal{A}$ is a set of basic actions, and $\mathcal{C}$ is a set of conditional actions. Beliefs are

ground atoms, and basic actions are pairs of pre- and post-conditions. The mental state of a BUnity agent is represented by the belief base, the only structure that changes. The mechanism of basic actions is based on an update operation. When the pre-condition of the action holds, beliefs are added and/or removed from the mental state, according to the post-condition. Basic actions are understood as a common ontology which can be shared among different agents. Conditional actions are built on top of basic actions by adding guards. In order to avoid confusion, we recall that though apparently similar, basic and conditional actions are in essence different as they demand information at distinct levels. The precondition of a basic action is a built-in restriction which internally enables belief updates. It is independent of the particular choice of application. A conditional action is executed in function of certain external requirements (reflected in the mental state). Thus it is application dependent. The external requirements are meant to be independent of the built-in enabling mechanism. Whether is agent i_1 or agent i_2 executing action a, the built-in condition should be the same for both agents. Nevertheless, each agent may have its own external trigger for action a.

We take, as an illustration, a known problem, which we first found in [19], of an agent building a tower of blocks. We represent blocks by naturals. An initial arrangement of three blocks $1, 2, 3$ is given there: 1 and 2 are on the table (0), and 3 is on top of 1. The goal[1] of the agent is to rearrange them such that they form the tower 321 (1 is on 0, 2 on top of 1 and 3 on top of 2). We further imagine that the agent has an extra task to clean the floor, if it is dirty. The only actions the agent can execute is to clean or move one block on the table, or on top of another block, if the latter is free.

$$
\begin{aligned}
\mathcal{B}_0 = \; & \{\, on(3,1), on(1,0), on(2,0), \\
& free(2), free(3), free(0) \,\} \\
\\
\mathcal{A} \; = \; & \{\, move(x,y,z) = \\
& (on(x,y) \wedge free(x) \wedge free(z), \\
& \{\, on(x,z), \neg on(x,y), \neg free(z) \,\}), \\
& clean = (\neg\, cleaned, \{cleaned\}) \,\} \\
\\
\mathcal{C} \; = \; & \{\, \neg(on(2,1) \wedge on(3,2)) \triangleright do(move(x,y,z)), \\
& \neg cleaned \triangleright do(clean)\}
\end{aligned}
$$

Fig. 1. A BUnity Toy Agent

The example from Figure 1 is taken in order to underline the difference between enabling conditions (for basic actions) and triggers (for conditional actions): on the one hand, it is possible to move a block x on top of another block z, if x and z are free; on the other hand, given the goal of the agent, moves are allowed only when the configuration is different than the final one (application specific).

The configuration of a BUpL agent is a tuple $\langle\, \mathcal{B}_0, \mathcal{A}, \mathcal{P}, p_0, \mathcal{R} \rangle$, where $\mathcal{P}$ is a set of plans, p_0 is the initial plan and $\mathcal{R}$ is a set of repair rules. Plans are orderings ";" and/or choices "+" of actions. Repair rules apply when actions fail. Their purpose is to replace

[1] For simplification, goals are not modelled in the agent languages [1].

$$
\begin{aligned}
\mathcal{B}_0 &= \{ \ldots \} \\
\mathcal{A} &= \{ \ldots \} \\
\mathcal{P} &= \{\, rearrange(x,y,z) = \\
&\quad move(x,0,y); move(z,0,x) \\
&\quad mission = clean + rearrange(2,1,3) \,\} \\
p_0 &= mission \\
\mathcal{R} &= \{\, on(x,y) \leftarrow move(x,y,0); mission \\
&\quad \neg cleaned \leftarrow clean; mission \,\}
\end{aligned}
$$

Fig. 2. A BUpL Toy Agent

the plan in execution. The mental states of a BUpL agent are pairs consisting of belief bases and plans.

We take as an example a BUpL agent that can choose between either to clean or to solve the *tower of blocks* problem or to do both tasks.

The BUpL agent from Figure 2 has the same initial belief base and the same basic action as the BUnity agent (we use the dots "…" to substitute them). The agent is modelled such that it illustrates the use of repair rules. The plan $rearrange(2,1,3)$ leads to the desired configuration 321 only if the blocks are already on the table. If the agent initially decides to execute $rearrange(2,1,3)$, that is, to move block 2 on 1 and block 3 on 2, the first move fails because block 3 in on 2. The failure is handled by $on(x,y) \leftarrow move(x,y,table); mission$. Choosing $[x/1][y/3]$ as a matcher, enables the agent to move block 3 on the table and after, the initial plan can be restarted. We note that the application of the repair rule $\neg cleaned \leftarrow clean; mission$ cannot handle the failure. It can, at most, result in the success of the action $clean$ (when the action has not been already executed).

One might be interested whether the BUpL agent in Figure 2 is a refinement of the BUnity agent in Figure 2. For the ease of reference, we identify the BUnity agent by i_a (since it is more abstract) and the BUpL agent by i_c (since it is more concrete). The methodology introduced in [1] consists of defining refinement in terms of trace inclusion. A BUpL agent with the mental state ms refines a BUnity agent with the mental state ms' ($ms \subseteq ms'$) if and only if $T(ms)$ is included in $T(ms')$. However, trace semantics is one of the coarsest (makes the most identifications) in the literature on concurrency. Proving refinement by definition is hard since the sets of traces are considerably large (or infinite). Instead, simulation has been adapted as a proof technique for the refinement of agent programs. The states ms and ms' are in a (weak) simulation relation $\sim$ if and only if whenever ms executes an action eventually with invisible τ steps ($ms \stackrel{a}{\Rightarrow} ms_1$) there is also the possibility that ms' executes the same action ($\exists ms'_1 (ms' \stackrel{a}{\rightarrow} ms'_1)$) and the resulting agents are in a simulation relation ($ms_1 \sim ms'_1$). The following proposition states that in the framework of BUpL and BUnity agents simulation is a sound and complete proof technique for refinement.

Proposition 1. *[1] Given a BUnity agent ms' and a BUpL agent ms we have that ms' simulates ms ($ms' \sim ms$) iff ms refines ms' ($ms \subseteq ms'$).*

We stress that a key factor in having simulation as a complete proof technique is the determinacy of BUnity agents [1].

As a final note, Proposition 1 tells us that i_c refines i_a since i_a can mimic any visible action that i_c executes.

3 Towards Multi-Agent Systems

If previously it was enough to refer to an agent by its current mental state, this is no longer the case when considering multi-agent systems. This is why we associate with each agent an identifier and we consider a multi-agent system as a finite set of such identifiers. We further denote a state of a multi-agent system by $\mathcal{M} = \{(i, ms_i) \mid i \in \mathcal{I}\}$, where $\mathcal{I}$ is the set of agent identifiers and ms_i is a mental state for the agent i. For the moment, we abstract from *what is* the mental state of an agent. The choice of representation is not relevant, we only need to consider that the way to change (update) the mental state of an agent is by performing actions. However, we will instantiate such generic ms_i by either a BUnity or a BUpL mental state whenever the distinction is necessary.

In order to control the behaviour of a multi-agent system we introduce *action-based choreographies*. We understand them as protocols which dictate the way agents behave by imposing ordering and synchrony constraints on their action executions. They represent *exogenous* coordination patterns and they are useful in scenarios where *action synchrony* is more important than *passing data*.

3.1 Action-Based Choreographies

For the ease of presentation, we represent choreographies as regular expressions where the basic elements are pairs (i, a). Such pairs denote that the agent i performs the action a. They can be combined by sequence, parallel, choice or Kleene operators, with the usual meaning: $(i_1, a_1); (i_2, a_2)$ models orderings, agent i_1 executes a_1 which is followed by agent i_2 executing a_2; $(i_1, a_1) \parallel (i_2, a_2)$ models synchronisations between actions, agent i_1 executes a_1 while i_2 executes a_2; $(i_1, a_1) + (i_2, a_2)$ models choices, either i_1 executes a_1 or i_2 executes a_2; $(i, a)^*$ models iterated execution of a by i. The operators respect the usual precedence relation[2]. As expected, the BNF grammar defining a choreography is $c ::= (i, a) \mid c + c \mid c \parallel c \mid c; c \mid c^*$. In order to describe the transitions of a multi-agent system in the presence of a choreography c, we first associate an LTS $\mathcal{S}^c$ to the choreography. We do this in the usual way, inductively on the size of the choreography such that the labels are of the form $\|_{i \in \mathcal{I}} (i, a_i)$. Such a transition system always exists (see [20] or [21] for a direct deterministic construction using the derivatives of a given regular expression). We take, as an illustration, the transition system from Figure 3 which is associated with the choreography c defined as the following regular expression:

$$(i_1, \mathit{clean}) \parallel (i_2, \mathit{move}(3, 1, 0)); \\ (i_1, \mathit{move}(2, 0, 1)); ((i_1, \mathit{move}(2, 0, 1)) \parallel (i_2, \mathit{clean})) + \\ (i_2, \mathit{move}(2, 0, 1)); ((i_2, \mathit{move}(2, 0, 1)) \parallel (i_1, \mathit{clean})).$$

[2] '+' $\leq_p$ '$\parallel$' $\leq_p$ ';' $\leq_p$ '*', where $\leq_p$ denote the precedence relation.

The choreography specifies that two agents i_1, i_2 work together in order to build the tower 321 and that while one is building the tower the other one is cleaning the table. More precisely, the definition of c says that first i_2 deconstructs the initial tower (by moving block 3 on 0) while i_2 is synchronously cleaning; next, either i_1 constructs the final tower while i_1 cleans or the other way around; afterwards, the system is in a final state. Further variations (for example, in the case of a higher tower, one agent builds an intermediate shorter tower leaving the other to finish the construction) are left to the imagination of the reader.

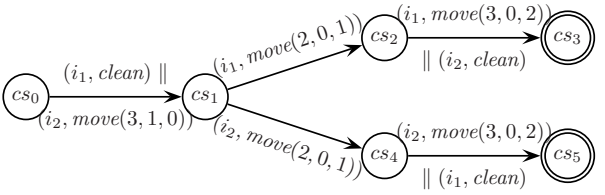

Fig. 3. The LTS associated to a choreography

We denote the synchronised product of a choreography c and a multi-agent system $\mathcal{I}$ by $\mathcal{S}^c \otimes \mathcal{I}$. Its states are pairs $(cs, \mathcal{M})$ with cs being a choreography state and $\mathcal{M}$ a multi-agent system state. The transition rule for $\mathcal{S}^c \otimes \mathcal{I}$ is:

$$\frac{cs \xrightarrow{l} cs' \qquad \bigwedge_{j \in \mathcal{J}} ms_j \overset{a_j}{\Rightarrow} ms'_j}{(cs, \mathcal{M}) \xrightarrow{l} (cs', \mathcal{M}')}(mas)$$

where cs, cs' are states of $\mathcal{S}^c$, l is a choreography label of the form $\|_{j \in \mathcal{J}}\ (j, a_j)$ with $\mathcal{J}$ being a subset of $\mathcal{I}$, ms_j, ms'_j are mental states of agent j and $\mathcal{M}, \mathcal{M}'$ are states of the multi-agent system with $\mathcal{M}'$ being $\mathcal{M} \setminus \{(j, ms_j) \mid j \in \mathcal{J}\} \cup \{(j, ms'_j) \mid j \in \mathcal{J}\}$. The notation $ms_j \overset{a_j}{\Rightarrow} ms'_j$ is used to denote that agent j performs action a_j (eventually with τ steps) in ms_j resulting in ms'_j. "Eventually τ steps" is needed for agents performing internal actions, like making choices among plans or handling failures in the case of BUpL agents. In the case of agents "in the style of BUnity", $\overset{a}{\Rightarrow}$ is simply $\xrightarrow{a}$ since Bunity agents do not have τ steps.

The transition (mas) says that the multi-agent system advances one step when the agents from $\mathcal{J}$ perform the actions specified by label l. The new state of the multi-agent system reflects the updates of the agents' mental states.

3.2 A Finer Notion of Refinement

We would like to have the result that if the agents (for example BUpL) in a multi-agent system $\mathcal{I}_1$ are refining the (BUnity) agents in $\mathcal{I}_2$ then $\mathcal{S}^c \otimes \mathcal{I}_1$ is a refinement of $\mathcal{S}^c \otimes \mathcal{I}_2$. When refinement is defined as trace inclusion, this is, indeed, the case, as it is stated in Proposition 2.

Proposition 2. *Given two multi-agent systems $\mathcal{I}_1$, $\mathcal{I}_2$ such that $(\forall i_1 \in \mathcal{I}_1)(\exists i_2 \in \mathcal{I}_2)\ (ms_{i_1} \subseteq ms_{i_2})$ and a choreography as $\mathcal{S}^c$ we have that $\mathcal{S}^c \otimes \mathcal{I}_1 \subseteq \mathcal{S}^c \otimes \mathcal{I}_2$.*

Proof. Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be the initial states of the multi-agent systems $\mathcal{I}_1$ and $\mathcal{I}_2$. Let also cs_0 be the initial state of the transition system $\mathcal{S}^c$ associated to the choreography c. It is enough to notice that $Tr((cs_0, \mathcal{M}_1)) = Tr(cs_0) \cap Tr(\mathcal{M}_1)$ and that $ms_{i_1} \subseteq ms_{i_2}$ for all $i_1 \in \mathcal{I}_1$ implies $Tr(\mathcal{M}_1) \subseteq Tr(\mathcal{M}_2)$. □

However, adding choreographies to a multi-agent system may introduce deadlocks. On the one hand, we would like to be able to infer from the semantics when a multi-agent system is in a deadlock state. On the other hand, we would like to have that the refinement of multi-agent systems does not introduce deadlocks. Trace semantics is a too coarse notion with respect to deadlocks. There are two consequences: neither is it enough to define the semantics of a multi-agent system as the set of all possible traces, nor is it satisfactory to define agent refinement as trace inclusion. We further illustrate these affirmations by means of simple examples. We take, for example, the choreography $c = (i, move(2, 0, 3))$, where i symbolically points to the agent i_c from Section 2.1. Looking at the plans and repair rules of i_c we see that such an action cannot take place. Thus, conforming to the transition rule (mas), there is no possible transition for the product $\mathcal{S}^c \otimes \mathcal{I}$. Just by analysing the behaviour (the empty trace) we cannot infer anything about deadlocked states: is it that the agent has no plan, or is it that the choreography asks for an impossible execution?

In order to distinguish between successful and deadlocked executions, we explicitly define a transition label $\surd$ different from any other action label. We then define for the product $\mathcal{S}^c \otimes \mathcal{I}$ an operational semantics $\mathcal{O}^{\surd}(\mathcal{S}^c \otimes \mathcal{I})$ as the set of maximal (in the sense that no further transition is possible) traces, ending with $\surd$ when the last state is successful (where the choreography state is final):

$$\{tr\surd \mid (cs_0, \mathcal{M}_0) \xrightarrow{tr} (cs, \mathcal{M}) \not\rightarrow, cs \in F(\mathcal{S}^c)\} \cup$$
$$\{tr \mid (cs_0, \mathcal{M}_0) \xrightarrow{tr} (cs, \mathcal{M}) \not\rightarrow, cs \notin F(\mathcal{S}^c)\} \cup \{\epsilon \mid (cs_0, \mathcal{M}_0) \not\rightarrow\},$$

where tr is a trace with respect to the transition (mas), $\mathcal{M}_0$ (resp. cs_0) is the initial state of $\mathcal{I}$ (resp. $\mathcal{S}^c$), $F(\mathcal{S}^c)$ is the set of final choreography states and ϵ denotes that there are no possible transitions from the initial state. We further denote by $\perp$ all product states $(cs, \mathcal{M})$ which are *deadlocked*, i.e., $(cs, \mathcal{M}) \not\rightarrow$ (no possible transition) and $cs \notin F(\mathcal{S}^c)$.

We can now define the refinement of multi-agent systems with respect to the above definition of the semantics $\mathcal{O}$.

Definition 1 (MAS Refinement). *Given a choreography c, we say that two multi-agent systems $\mathcal{I}_1$ and $\mathcal{I}_2$ are in a refinement relation if and only if the set of maximal traces of $\mathcal{S}^c \otimes \mathcal{I}_1$ are included in the set of maximal traces of $\mathcal{S}^c \otimes \mathcal{I}_2$. That is, $\mathcal{O}^{\surd}(\mathcal{S}^c \otimes \mathcal{I}_1) \subseteq \mathcal{O}^{\surd}(\mathcal{S}^c \otimes \mathcal{I}_2)$.*

We now approach the problem that appears when considering agent refinement defined as trace inclusion. It can be the case that the agents in the concrete system refine (with respect to trace inclusion) the agents in the abstract system, nevertheless the concrete system deadlocks for a particular choreography. We take, for instance, the agents i_a and i_c from Section 2.1. We can easily design a choreography which works fine with i_a (does not deadlock) and on the contrary with i_c. One example is the already mentioned

$c = (i, move(2, 0, 3))$, where now, i points to either i_a or i_c up to a renaming. We recall that i_c is a refinement of i_a. However, i_c cannot execute the move i_c cannot execute the move (since the move is irrelevant for building the tower 321 and at implementation time it matters to be as precise as possible), while i_a can (since in a specification "necessary" is more important than "sufficiency").

What the above illustration implies is that refinement as trace inclusion, though being a satisfactory definition at individual agent level, is not a strong enough condition to ensure refinement at a multi-agent level, in the presence of an arbitrary choreography. It follows that we need to redefine individual agent refinement such that multi-agent system refinement (as maximal trace inclusion) is compositional with respect to any choreography. In this sense, a choreography acts as a context for multi-agent systems, i.e., whatever the context is, it does not affect the visible results of the agents' executions but restricts them by activating only certain traces (the other traces still exist, however, they are inactive).

In order to have a proper definition of agent refinement we look for a finer notion of traces. The key ingredient lies in *enabling* conditions for actions. Given a mental state ms, we look at all the actions enabled to be executed from ms. We denote them by $E(ms) = \{a \in \mathcal{A} \mid \exists ms'(ms \xrightarrow{a} ms')\}$ and we call $E(ms)$ a *ready* set. We can now present *ready traces* as possibly infinite sequences $X_1, a_1, X_2, a_2, \ldots$ where $ms_0 \xrightarrow{a_1} ms_1 \xrightarrow{a_2} ms_2 \ldots$ and $X_{i+1} = E(ms_i)$. We denote the set of all ready traces from a state ms_0 as $RT(ms_0)$. Compared to the definition of traces, ready traces are a much more finer notion in the sense that they record not only actions which have been executed but also sets of actions which are *enabled* to be executed at each step.

Definition 2 (Ready Agent Refinement). *We say that two agents with initial mental states ms and ms' are in a ready refinement relation (i.e., $ms \subseteq_{rt} ms'$) if and only if the ready traces of ms are included in the ready traces of ms' (i.e., $RT(ms) \subseteq RT(ms')$).*

We can now present our main result which states that refinement is compositional, in the sense that if there is a ready refinement between the agents composing two multi-agent sytems it is then the case that one multi-agent system refines the other in the presence of any choreography.

Theorem 1. *Let $\mathcal{I}_1$, $\mathcal{I}_2$ be two multi-agent systems such that $(\forall i_1 \in \mathcal{I}_1)\ (\exists i_2 \in \mathcal{I}_2)\ (ms_{i_1} \subseteq_{rt} ms_{i_2})$ and a choreography c with the associated LTS $\mathcal{S}^c$. We have that $\mathcal{I}_1$ refines $\mathcal{I}_2$, that is, $\mathcal{O}^{\surd}(\mathcal{S}^c \otimes \mathcal{I}_1) \subseteq \mathcal{O}^{\surd}(\mathcal{S}^c \otimes \mathcal{I}_2)$.*

Proof. What we need to further prove with respect to Proposition 2 is that the set of enabled actions is a key factor in identifying failures in both implementation and specification. Assume a maximal trace tr in $\mathcal{O}^{\surd}(\mathcal{S}^c \otimes \mathcal{I}_1)$ leading to a non final choreography state cs. Given cs_0 and $\mathcal{M}_1$ as the initial states of $\mathcal{S}^c$, $\mathcal{I}_1$, we have that $(cs_0, \mathcal{M}_1) \xrightarrow{tr} (cs, \mathcal{M})\ (cs, \mathcal{M}) \not\xrightarrow{l}$ for all $l = \|_{j \in \mathcal{J}}\ (j, a_j)$ such that $cs \xrightarrow{l} cs'$. By rule (mas) this implies that there exists an agent identified by j which cannot perform the action indicated. Thus the corresponding trace of j ends with a ready set X with the property that a_j is not included in it. We know that each implementation agent has a corresponding specification, be it j', such that j ready refines j'. If we, on the other hand, assume that j' can, on the contrary, execute a_j we would have that in a given state

j' has besides the ready set X another ready set Y which includes a_j. This contradicts the maximality of the ready set. □

As a direct consequence of the above theorem, we are able to infer the absence of deadlock in the concrete system from the absence of deadlock in the abstract one:

Corollary 1. *Let $\mathcal{I}_1, \mathcal{I}_2$ be two multi-agent systems with initial states $\mathcal{M}_1$ and $\mathcal{M}_2$. Let c be a choreography with the associated LTS $\mathcal{S}^c$ and initial state cs_0. We have that if $\mathcal{I}_1$ refines $\mathcal{I}_2$ ($\mathcal{O}^{\surd}(\mathcal{S}^c \otimes \mathcal{I}_1) \subseteq \mathcal{O}^{\surd}(\mathcal{S}^{c'} \otimes \mathcal{I}_2)$) and c does not deadlock the specification ($(cs_0, \mathcal{M}_2) \not\mapsto^* \perp$) it is then also the case that c does not deadlock the implementation ($(cs_0, \mathcal{M}_1) \not\mapsto^* \perp$).*

As we have briefly explained in Section 2.1, proving refinement by deciding trace inclusion is an inefficient procedure. Since the same argument holds for ready refinement, a more adequate approach is needed. If previously we have adopted simulation as a proof technique for refinement, now we consider *weak ready simulation*.

Definition 3 (Weak Ready Simulation). *We say that two agents with initial mental states ms and ms' are in a (weak) ready simulation relation ($ms \lesssim_{rs} ms'$) if and only if $ms \lesssim ms'$ and the corresponding ready sets are equal ($E(ms) = E(ms')$).*

As it is the case for simulation being a sound and complete proof technique for refinement, analogously we can have a similar result for ready simulation. We recall that determinacy plays an important role in the proof for completeness.

Proposition 3. *Given two agents with initial mental states ms and ms', where the one with ms is deterministic, we have that $ms \lesssim_{rs} ms'$ iff $ms' \subseteq_{rt} ms$.*

Remark 1. For the sake of generality, in the definitions from this section we have used the symbolic notations ms, ms'. BUnity and BUpL agents can be seen as (are, in fact) instantiations. Proposition 3 relates to Proposition 1.

Recalling the BUpL and BUnity agents i_a and i_c, we note that though i_a simulates i_c it is not also the case that it ready simulates. This is because the ready set of the BUnity agent is always larger than the one of the BUpL agent. One basic argument is that i_a can always "undo a block move", while i_a cannot. However, let us see what would have happened if we were to consider changing i_a by replacing the conditional action $\neg(on(2,1) \wedge on(3,2)) \triangleright do(move(x,y,z))$ with the set from Figure 4:

$$\begin{aligned} \mathcal{C} = \{\ & \neg on(2,1) \triangleright do(move(2,0,1)), \\ & \neg on(3,2) \wedge on(2,1) \triangleright do(move(3,0,2)), \\ & \neg(on(2,1) \wedge on(3,2)) \triangleright do(move(x,y,0))\ \} \end{aligned}$$

Fig. 4. Adapting i_a to ready simulate i_c

We now have a BUnity agent which is less abstract. Basically, the instantiation from the first two conditional actions disallows any spurious "to and fro" sequence of moves like $move(x,y,z)$ followed by $move(x,z,y)$ which practically undoes the

previous step leading to exactly the previous configuration. The instantiation is obvious when one looks at the final (wanted) configuration. The last conditional action allows "destructing" steps by moving blocks on the table. It can still be considered as a specification. It provides no information about the order of executing the moves since this is not important at the abstraction level. With the above change, the new BUnity agent ready simulates i_c. To see this, it suffices to notice that the only BUpL ready trace is $\{move(3,1,0)\}$, $move(3,1,0)$, $\{move(2,0,1)\}$, $move(2,0,1)$, $\{move(3,0,2)\}$, $move(3,0,2)$ which is also the only BUnity ready trace. The same equality of ready traces holds when we consider the additional $clean$ action.

We recall the choreography from Figure 3 and we consider a BUnity multi-agent system which consists of two copies of i_a (enabled to execute also *clean*). For either branch, the executions (with respect to the transition (mas)) of the multi-agent system are successful (the choreography reaches a final state). Since i_c ready refines i_a, by Corollary 1 we can deduce that also the executions of a multi-agent system which consists of a two i_c copies are successful.

4 Timing Extensions of MAS

Our approach in adding time to multi-agent systems consists of adapting the theory of timed-automata [3]. A *timed automaton* is a finite transition system extended with real-valued clock variables. Time advances only in states since transitions are instantaneous. Clocks can be reset at zero simultaneously with any transition. At any instant, the reading of a clock equals the time elapsed since the last time it was reset. States and transitions have *clock constraints*, defined by the following grammar:

$$\phi_c ::= x \leq t \mid t \leq x \mid x < t \mid t < x \mid \phi_c \wedge \phi_c,$$

where $t \in \mathbb{Q}$ is a constant and x is a clock. When a clock constraint is associated with a state, it is called *invariant*, and it expresses that time can elapse in the state as long as the invariant stays true. When a clock constraint is associated with a transition, it is called *guard*, and it expresses that the action may be taken only if the current values of the clocks satisfy the guard.

In our multi-agent setting, *timed choreographies* are meant to impose time constraints on the actions executed by the agents. We model them as timed automata. We take, as an example, the choreography from Figure 5. There is a single clock x. The initial state cs_0 has no invariant constraint and this means that an arbitrary amount of time can elapse in cs_0. The clock x is always reset with the transition from cs_0 to cs_1. The invariant $x < 5$ associated with the state cs_1 ensures that the synchronous actions $clean$ and $move(C, A, floor)$ must be executed within 5 units of time. The guard $x > 6$ associated with the transition from cs_2 to cs_3 ensures that the agents cannot spend an indefinite time in cs_2 because they must finish their tasks after 6 units of time.

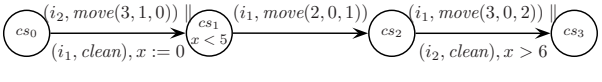

Fig. 5. A timed choreography

We now approach the issue of modelling time in BUnity and BUpL agents. In this regard, we consider that agents have a set of local clocks and that clock valuations are can be performed by an observer. We further pose the problem of how agents make use of clocks. We recall the design principle: "the specification of basic actions does not come with time", thus actions are instantaneous. This implies that, in order to make the time pass, we need to extend the syntax of the agent languages with new application specific constructions such that the ontology of basic actions remains timeless (basic actions being specified only in terms of pre/post conditions). This is why we introduce *delay actions*, $\phi \rightarrow I$, where ϕ is a query on the belief base and I is an invariant like $x \leq 1$. Basically, their purpose is to make time elapse in a mental state where certain beliefs hold. As long as the invariant is true, the agent can stay in the same state while time passes. We refer to $\mathcal{D}$ as the set of delays of either a BUnity or a BUpL agent. In what follows, we discuss the time extension for each language separately.

In order to extend BUnity with time, we only have to focus on conditional actions. First, their queries are defined on both belief bases and clock valuations. Second, conditional actions specify the set of clocks to be reset after the execution of basic actions. The syntax becomes $\{\phi \wedge \phi_c\} \triangleright do(a), \lambda$. Timed conditional actions are meant to say that if certain beliefs ϕ hold in the current mental state of a BUnity agent (as before) and additionally, certain clock constraints ϕ_c are satisfied, then the basic action a is executed and the clocks from the set λ are reset to 0.

We note that our design decision is to separate the implementation of action delays from the one of conditional actions. This is because a construction like $\{\phi\} \triangleright I, do(a), \lambda$ is ambiguous. If ϕ holds, it can either be the case that time elapses with respect to the invariant I and a is suspended, or that a is immediately executed.

To illustrate the above constructions we recall the BUnity agent i_a with the update from Figure 4. We basically extend the BUnity agent such that the agent has one clock, be it x, which is reset by conditional actions, and such that the agent can delay in given states, thus letting the time pass.

$$
\begin{aligned}
\mathcal{C} = \{\ & \top \triangleright (do(clean), x := 0),\\
& \neg on(2,1) \triangleright do(move(2,0,1)),\\
& \neg on(3,2) \wedge on(2,1) \triangleright do(move(3,0,2)),\\
& \neg(on(2,1) \wedge on(3,2)) \triangleright\\
& (do(move(x_1,x_2,0)), x := 0)\ \}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{D} = \{\ & on(3,0) \vee cleaned \leftarrow (x < 9),\\
& on(2,1) \vee cleaned \leftarrow (x < 10)\ \}
\end{aligned}
$$

Fig. 6. Extending i_a with clock constraints

Figure 6 shows a possible timed extension. The clock x is reset after either performing *clean* or moving a block on the table. The agent can delay until the clock valuates to 9 (resp. 10) units of time after moving 3 on 0 (resp. 2 on 1).

In order to extend BUpL with time we focus on plans and repair rules. With respect to plans, previous calls $a; p$ are replaced by $(\phi_c, a, \lambda); p$ and $(\phi \rightarrow I); p$, where ϕ_c is time constraining the execution of action a and λ is the set of clocks to be reset. To

simplify notation, if clock constraints and clock resets are absent we use a instead of (a). With respect to repair rules, we only need to consider that the enabling conditions can be defined not only on belief bases but also on clock valuations.

We remark that if previously actions failed when certain beliefs did not hold in a given mental state, it is now the case that actions fail also when certain clock constraints are not satisfied. Consider, for example, the plan $((x < 1), a, [x := 0]); ((x > 2), b, \emptyset)$. There is no delay action between a and b, thus the time does not pass and x remains 0, meaning that b cannot be executed. Such situations are handled by means of the general semantics of the repair rules. There are two possibilities: either to execute an action with a time constraint that holds, or to make time elapse. The latter is achieved by triggering a repair rule like $true \leftarrow \delta$, where for example δ is a delay action $true \rightarrow true$ which allows an indefinite amount of time to pass.

To see a concrete example, we recall the BUpL agent i_c. We consider two delay actions $true \leftarrow (x < 9)$ and $true \leftarrow (x < 10)$. We further make the delays and the clock resets transparent in the plans. The plan $rearrange(x_1, x_2, x_3)$ changes to $true \leftarrow (x < 9); move(x_1, 0, x_2); true \leftarrow (x < 10); move(x_3, 0, x_1)$ such that time passes between moves. The plan $mission$ changes to $(true, clean, x := 0) + rearrange(2, 1, 3)$ such that the clock x is reset after the action $clean$ has been executed.

The observal behaviour of either timed BUnity or BUpL agents is defined in terms of timed traces. A *timed trace* is a (possibly infinite) sequence $(t_1, a_1)\ (t_2, a_2) \ldots (t_i, a_i) \ldots$ where $t_i \in \mathbb{R}_+$ with $t_i \leq t_{i+1}$ for all $i \geq 1$. We call t_i a *time-stamp* of action a_i since it denotes the absolute time that passed before a_i was executed. We then have that a timed BUnity or BUpL agent computation over a timed trace $(t_1, a_1)(t_2, a_2) \ldots (t_i, a_i) \ldots$ is a sequence of transitions:

$$ms_0, \nu_0 \xrightarrow{\delta_1} \xrightarrow{a_1} ms_1, \nu_1 \xrightarrow{\delta_2} \xrightarrow{a_2} ms_2, \nu_2 \ldots$$

where ms_i is a BUnity (BUpL) mental state and t_i are satisfying the condition $t_i = t_{i-1} + \delta_i$ for all $i \geq 1$.

For example, a possible timed trace for either the timed BUpL or BUnity agent is $(0, \mathit{clean}), (7, \mathit{move}(3, 1, 0)), (8, \mathit{move}(2, 0, 1)), (9, \mathit{move}(3, 0, 2))$. It is, in fact, the case that any BUpL timed trace is also a BUnity timed trace, thus the two agents are again in a refinement relation. We do not elaborate more on timed refinement. We only mention that the methodology we described in Section 3 applies in the timed framework. Along the same line, the key factor is determinacy, which is ensured by the *disjointness* condition, i.e., clocks associated with the same action must be disjoint.

5 Conclusion

We have extended the notion of refinement of individual agents to multi-agent systems, where the behaviour of the agents is coordinated by choreographies. Our approach to introducing choreographies to multi-agent systems consisted of defining them as action-based coordination mechanisms. In such a framework, we have the results that agent refinement is a sufficient condition for multi-agent system refinement and that this latter notion preserves deadlock freeness. We have further illustrated a timed extension of multi-agent systems by means of timed automata where the same refinement methodology can be adapted.

We have stressed the importance of verification from the introduction. Our goal was to describe a general methodology for a top-down design of multi-agent systems which makes it simple to execute and verify agent programs. Concerning this practical side we mention that we have already implemented our formalism in Maude [22], a rewriting logic software. Maude is an encompassing framework where we prototyped the agent languages we described such that it is possible to (1) execute agents by rewriting; (2) verify agents by means of simulation, model-checking, searching, or testing. Since we were mainly interested in refinement, the properties we focused on were correctness properties, i.e., model-checking for the absence of deadlock in the product of a BUpL and BUnity agent. However, we have experimented with different other safety and liveness properties. With respect to the timed extensions of the languages, we have prototyped them using Real-Time Maude. We have also experimented with UPPAAL [23], however at a more abstract and syntactical level. Further details, the current version and updates of the implementation can be found at `http://homepages.cwi.nl/~astefano/agents`. Extensions with respect to model-checking timed agents and automatically generating test cases for verifying infinite state agents need to be further investigated.

References

1. Astefanoaei, L., de Boer, F.S.: Model-checking agent refinement. In: Padgham, L., Parkes, D.C., Müller, J., Parsons, S. (eds.) AAMAS, IFAAMAS, pp. 705–712 (2008)
2. Chandy, K.M., Misra, J.: Parallel Program Design: A Foundation. Addison Wesley Publishing Company, Inc., Reading (1988)
3. Alur, R.: Timed automata. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 8–22. Springer, Heidelberg (1999)
4. van Glabbeek, R.J.: The linear time-branching time spectrum (extended abstract). In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 278–297. Springer, Heidelberg (1990)
5. Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: Verifying multi-agent programs by model checking. Autonomous Agents and Multi-Agent Systems 12(2), 239–256 (2006)
6. Bosse, T., Jonker, C.M., van der Meij, L., Sharpanskykh, A., Treur, J.: Specification and verification of dynamics in cognitive agent models. In: IAT, pp. 247–254 (2006)
7. Raimondi, F., Lomuscio, A.: Automatic verification of multi-agent systems by model checking via ordered binary decision diagrams. J. Applied Logic 5(2), 235–251 (2007)
8. Ricci, A., Viroli, M., Omicini, A.: Give agents their artifacts: the A&A approach for engineering working environments in mas. In: Durfee, E.H., Yokoo, M., Huhns, M.N., Shehory, O. (eds.) AAMAS, IFAAMAS, p. 150 (2007)
9. Arbab, F., Astefanoaei, L., de Boer, F.S., Dastani, M., Meyer, J.J.C., Tinnemeier, N.: Reo connectors as coordination artifacts in 2APL systems. In: Bui, T.D., Ho, T.V., Ha, Q.T. (eds.) PRIMA 2008. LNCS (LNAI), vol. 5357, pp. 42–53. Springer, Heidelberg (2008)
10. Gelernter, D., Zuck, L.D.: On what linda is: Formal description of linda as a reactive system. In: Garlan, D., Le Métayer, D. (eds.) COORDINATION 1997. LNCS, vol. 1282, pp. 187–204. Springer, Heidelberg (1997)
11. Dastani, M., Grossi, D., Meyer, J.J.C., Tinnemeier, N.: Normative multi-agent programs and their logics. In: KRAMAS 2008: Proceedings of the Workshop on Knowledge Representation for Agents and Multi-Agent Systems (2008)

12. Tinnemeier, N., Dastani, M., Meyer, J.J.: Orwell's nightmare for agents? programming multi-agent organisations. In: Proc. of ProMAS 2008 (2008)
13. Boella, G., Broersen, J., van der Torre, L.: Reasoning about constitutive norms, counts-as conditionals, institutions, deadlines and violations. In: Bui, T.D., Ho, T.V., Ha, Q.T. (eds.) PRIMA 2008. LNCS (LNAI), vol. 5357, pp. 86–97. Springer, Heidelberg (2008)
14. Meng, S., Arbab, F.: Web services choreography and orchestration in Reo and constraint automata. In: Cho, Y., Wainwright, R.L., Haddad, H., Shin, S.Y., Koo, Y.W. (eds.) SAC, pp. 346–353. ACM, New York (2007)
15. Misra, J.: A programming model for the orchestration of web services. In: SEFM, pp. 2–11. IEEE Computer Society, Los Alamitos (2004)
16. Baldoni, M., Baroglio, C., Chopra, A.K., Desai, N., Patti, V., Singh, M.P.: Choice, interoperability, and conformance in interaction protocols and service choreographies. In: Sierra, C., Castelfranchi, C., Decker, K.S., Sichman, J.S. (eds.) AAMAS (2), IFAAMAS, pp. 843–850 (2009)
17. Hessel, A., Larsen, K.G., Mikucionis, M., Nielsen, B., Pettersson, P., Skou, A.: Testing real-time systems using UPPAAL. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) FORTEST. LNCS, vol. 4949, pp. 77–117. Springer, Heidelberg (2008)
18. Bouyer, P., Brinksma, E., Larsen, K.G.: Optimal infinite scheduling for multi-priced timed automata. Formal Methods in System Design 32(1), 3–23 (2008)
19. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.J.C.: Agent programming in 3APL. Autonomous Agents and Multi-Agent Systems 2(4), 357–401 (1999)
20. Hopcroft, J.E., Motwani, R., Rotwani, U.J.D.: Introduction to Automata Theory, Languages and Computability. Addison-Wesley Longman Publishing Co., Inc., Boston (2000)
21. Brzozowski, J.A.: Derivatives of regular expressions. J. ACM 11(4), 481–494 (1964)
22. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L. (eds.): All About Maude - A High-Performance Logical Framework. LNCS, vol. 4350. Springer, Heidelberg (2007)
23. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: UPPAAL 4.0. In: QEST, pp. 125–126. IEEE Computer Society, Los Alamitos (2006)

Goal Generation from Possibilistic Beliefs Based on Trust and Distrust

Célia da Costa Pereira and Andrea G.B. Tettamanzi

Università degli Studi di Milano
Dipartimento di Tecnologie dell'Informazione
via Bramante 65, I-26013 Crema, Italy
{celia.pereira,andrea.tettamanzi}@unimi.it

Abstract. The extent to which a rational agent changes its beliefs may depend on several factors like the trustworthiness of the source of new information, the agent's competence in judging the truth of new information, the mental spirit of the agent (optimistic, pessimistic, pragmatic, etc), the agent's attitude towards information coming from unknown sources, or sources the agent knows as being malicious, or sources the agent knows as providers of usually correct information, and so on.

We propose and discuss three different agent's belief behaviors to be used in a goal (desire) generation and adoption framework. The originality of the proposals is that the trustworthiness of a source depends not only on the degree of trust but also on an *independent* degree of distrust. Explicitly taking distrust into account allows us to mark a clear difference between the distinct notions of *negative* trust and *insufficient* trust. More precisely, it is possible, unlike in approaches where only trust is accounted for, to "weigh" differently information from *helpful*, *malicious*, *unknown*, or *neutral* sources.

1 Introduction and Motivation

The goals to be adopted by a BDI agent [22] in a given situation may depend on the agent's beliefs, desires, and obligations [23]. Most of the existing works on goal generation, like for example the one proposed in [11], consider the notion of belief as an all-or-nothing concept: either the agent believes something, or it does not. However, as pointed out by Hansson in [16] for example, believing is a matter of degree. He underlines two notions of degree of belief. One is the static concept of degree of confidence. In this sense, the more an agent's degree of belief in a sentence is higher, the more confidently it entertains that belief. The other notion is the dynamic concept of degree of resistance to change. In that sense, the higher an agent's degree of belief in a sentence is, the more difficult it is to change that belief.

In classical logic, beliefs are considered to be certainly true, and the negation of beliefs to be certainly false. This assumption does not cover the intermediate cases pointed out by Hansson, that is, the cases in which beliefs are neither fully believed nor fully disbelieved. In the possibility theory setting [12, 13], the notion of graded belief is captured in terms of two measures: necessity and possibility. A first degree of belief in a proposition, computed thanks to the necessity measure, is valued on a

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 35–50, 2010.

unipolar scale, where 0 means an absence of belief rather than believing the opposite (Paul does not believe p does not mean that Paul believes $\neg p$). A second degree, valued on a different unipolar scale and attached to propositions, expresses plausibility and is computed thanks to the possibility measure. If the plausibility of a proposition is 0, it means certainty of falseness (If Paul thinks that it is fully impossible that it will rain tomorrow, it means that Paul is certain that tomorow it will not rain), whereas 1 just reflects possibility, not certainty (If Paul thinks that it is fully possible that it will rain tomorrow, it does not mean that Paul is certain that tomorow it will rain). Thanks to the duality between necessity and possibility measure, the set of disbelieved proposition can then be inferred from the set of believed ones.

Recently, we have proposed [7] a belief change operator for goal generation in line with Hansson's considerations. In that approach, the sources of information may also be partially trusted and, as a consequence, information coming from such sources may be partially believed.

The main lack in that approach is the way the concept of *distrust* is implicitly considered, that is, as the complement of *trust* (*trust* $= 1 -$ *distrust*). However, trust and distrust may derive from different kinds of information (or from different sides of the personality) and, therefore, can coexist without being complementary [15, 9, 20]. For instance, one may not trust a source because of lack of positive evidence, but this does not necessarily mean (s)he distrusts it. Distrust can play an important role in an agent's goal generation reasoning complementing trust. In particular, in a framework of goal generation based on beliefs and desires[1], taking distrust explicitly into account allows an agent, e.g., to avoid dropping a goal just because favorable information comes from an unknown source (neither trusted nor distrusted) — the absence of trust does not erroneously mean full distrust.

We propose a way to take these facts into consideration by using possibility theory for representing degrees of beliefs and by concentrating on the influence of new information in the agent's beliefs. The latter point supposes that a source may be trusted and/or distrusted to a certain extent. This means that we explicitly consider not only the trust degree in the source but also the distrust degree. To this aim, the trustworthiness of a source is represented as a (trust, distrust) pair, and intuitionistic fuzzy logic [1] is used to represent the uncertainty on the trust degree introduced by the explicit presence of distrust. On that basis, we propose three belief change operators to model the attitude of an agent towards information from trusted, malicious, neutral (trusted and malicious to same extent) or unknown sources. Besides, such operators allow us to account for the fact that the acceptance or rejection of new information depends on several factors like the agent's competence in judging the truth of incoming information, the agent's state of spirit, the correctness of information provided by the source in the past, and so on.

The paper is organized as follows: Section 2 provides minimal background on fuzzy sets, possibility theory, and intuitionistic fuzzy logic; Section 3 motivates and discusses a bipolar view of trust and distrust; Section 4 introduces an abstract beliefs-desires-goals agent model; Section 5 presents a trust-based belief change operator adapted from previous work and proposes three extensions thereof for dealing with explicitly given trust and distrust; Section 6 describes the goal generation process; and Section 7 concludes.

[1] Here, for sake of simplicity, we will not consider intentions.

2 Basic Considerations

2.1 Fuzzy Sets

Fuzzy sets [26] allow the representation of imprecise information. Information is imprecise when the value of the variable to which it refers cannot be completely determined within a given universe of discourse. For example, among the existing fruits, it is easy to define the set of apples. Instead, it is not so easy to define in a clear cut-way the set of ripe apples because ripeness is a gradual notion. A fuzzy set is appropriate to represent this kind of situation.

Fuzzy sets are a generalization of classical sets obtained by replacing the characteristic function of a set A, χ_A, which takes up values in $\{0, 1\}$ ($\chi_A(x) = 1$ iff $x \in A$, $\chi_A(x) = 0$ otherwise) with a *membership function* μ_A, which can take up any value in $[0, 1]$. The value $\mu_A(x)$ or, more simply, $A(x)$ is the membership degree of element x in A, i.e., the degree to which x belongs in A. A fuzzy set is then completely defined by its membership function.

2.2 Possibility Theory and Possibility Distribution

The membership function of a fuzzy set describes the more or less possible and mutually exclusive values of one (or more) variable(s). Such a function can then be seen as a possibility distribution [27]. Indeed, if F designates the fuzzy set of possible values of a variable X, $\pi_X = \mu_F$ is called the possibility distribution associated to X. The identity $\mu_F(u) = \pi_X(u)$ means that the membership degree of u to F is equal to the possibility degree of X being equal to u when all we know about X is that its value is in F. A possibility distribution for which there exists a completely possible value ($\exists u_0; \pi(u_0) = 1$) is said to be *normalized*.

Possibility and Necessity Measures. A possibility distribution π induces a *possibility measure* and its dual *necessity measure*, denoted by Π and N respectively. Both measures apply to a crisp set A and are defined as follows:

$$\Pi(A) \equiv \sup_{s \in A} \pi(s); \tag{1}$$

$$N(A) \equiv 1 - \pi(\bar{A}) = \inf_{s \in \bar{A}} \{1 - \Pi(s)\}. \tag{2}$$

In words, the possibility measure of set A corresponds to the greatest of the possibilities associated to its elements; conversely, the necessity measure of A is equivalent to the impossibility of its complement $\bar{A}$.

A few properties of possibility and necessity measures induced by a normalized possibility distribution on a finite universe of discourse U are the following, for all subsets $A, B \subseteq U$:

1. $\Pi(A \cup B) = \max\{\Pi(A), \Pi(B)\}$;
2. $\Pi(\emptyset) = 0, \Pi(U) = 1$;
3. $N(A \cap B) = \min\{N(A), N(B)\}$;
4. $N(\emptyset) = 0, N(U) = 1$;

5. $\Pi(A) = 1 - N(\bar{A})$ (duality);
6. $\Pi(A) \geq N(A)$;
7. $N(A) > 0$ implies $\Pi(A) = 1$;
8. $\Pi(A) < 1$ implies $N(A) = 0$.

An immediate consequence of these properties is that either $\Pi(A) = 1$ or $\Pi(\bar{A}) = 1$. Both a set A and its complement having a possibility of 1 is the case of complete ignorance on A.

2.3 Intuitionistic Fuzzy Logic

Fuzzy set theory has been extended to intuitionistic fuzzy set (IFS for short) theory [1]. In fuzzy set theory, it is implicitly assumed that the fact that an element x "belongs" with a degree $\mu_A(x)$ in a fuzzy set A implies that x should "not belong" to A to the extent $1 - \mu_A(x)$. An intuitionistic fuzzy set S, instead, explicitly assigns to each element x of the considered universe of discourse both a degree of membership $\mu_S(x) \in [0, 1]$ and one of non-membership $\nu_S(x) \in [0, 1]$ which are such that $\mu_S(x) + \nu_S(x) \leq 1$. Obviously, when $\mu_S(x) + \nu_S(x) = 1$ for all the elements of the universe, the traditional fuzzy set concept is recovered.

Deschrijver and Kerr showed [10] that IFS theory is formally equivalent to interval-valued fuzzy set (IVFS) theory, which is another extension of fuzzy set theory in which the membership degrees are subintervals of $[0, 1]$ [24]. The IFS pair $(\mu_S(x), \nu_S(x))$ corresponds to the IVFS interval $[\mu_S(x), 1 - \nu_S(x)]$, indicating that the degree to which x "belongs" in S can range from $\mu_S(x)$ to $1 - \nu_S(x)$. The same authors define the *hesitation degree*, $h \in [0, 1]$, as the length of such interval, $h = 1 - \mu_S(x) - \nu_S(x)$. Hesitation h represents the uncertainty about the actual membership degree of x.

IFS is suitable to representing gender, for example. Indeed, according to the Intersex Society of North America [21], approximately 1 in 2000 children are born with a condition of "ambiguous" external genitalia – it is not clear if they are female or male. Let M be a fuzzy set representing males, and F be a fuzzy set representing females. A newborn x can be identified as a male if $\mu_M(x) = 1$, $\nu_M(x) = 0$, and $h = 0$; or as a female if $\mu_F(x) = 1$, $\nu_F(x) = 0$, and $h = 0$; or as "ambiguous" if $\mu_M(x)(\mu_F(x)) = \alpha$, $\nu_M(x)(\nu_F(x)) = \beta$, and $h = 1 - \alpha - \beta > 0$.

3 Representing Trust and Distrust

Most existing computational models usually deal with trust in a binary way: they assume that a source is to be trusted or not, and they compute the probability that the source can be trusted. However, sources can not always be divided into trustworthy and untrustworthy in a clear-cut way. Some sources may be trusted to a certain extent. To take this fact into account, we represent trust and distrust as fuzzy degrees. A direct consequence of this choice is that facts may be believed to a degree and desires and goals may be adopted to a given extent.

It must be stressed that our aim is not to compute degrees of trust and distrust of sources; like in [18], we are just interested in how these degrees influence the agent's beliefs, and, by way of a deliberative process, desires and goals.

Approaches to the problem of deriving/assigning degrees of trust (and distrust) to information sources can be found, for example, in [5], [8], [2].

We propose to define the *trustworthiness* score of a source for an agent as follows:

Definition 1 (Trustworthiness of a Source). *Let $\tau_s \in [0,1]$ be the degree to which an agent trusts source s, and $\delta_s \in [0,1]$ the degree to which it distrusts s, with $\tau_s + \delta_s \leq 1$. The trustworthiness score of s for the agent is the pair (τ_s, δ_s).*

Following Deschrijver and Kerr's viewpoint, the trustworthiness (τ, δ) of a source corresponds to the interval $[\tau, 1-\delta]$, indicating that the trust degree can range from τ to $1-\delta$. Therefore, the hesitation degree $h = 1 - \tau - \delta$ represents the uncertainty, or doubt, about the actual trust value. E.g., if a source has trustworthiness $(0.2, 0)$, this means that the agent trusts the source to degree 0.2, but possibly more, because there is much room for doubt ($h = 0.8$). More precisely, it means that the agent may trust the source to a degree varying from 0.2 to 1. Instead, if the trustworthiness is $(0.6, 0.4)$, the agent trusts the source to degree 0.6 but not more ($h = 0$).

Thanks to these considerations, we can represent the trustworthiness score of a source more faithfully than in existing approaches. In particular, we can explicitly represent the following special cases:

- $(0,1)$: the agent has reasons to fully distrust the source, hence it has no hesitation ($h = 0$);
- $(0,0)$: the agent has no information about the source and hence no reason to trust the source, but also no reason to distrust it; therefore, it fully hesitates in trusting it ($h = 1$);
- $(1,0)$: the agent has reasons to fully trust the source, hence it has no hesitation ($h = 0$).

As we can see, by considering both the (not necessarily related) concepts of trust and distrust, it is possible to differentiate between absence of trust caused by presence of distrust (e.g., information provided by a malicious source) versus by lack of knowledge (e.g., as towards an unknown source).

Let us consider the following example. Paul's son needs urgent treatment. John, Robert and Jimmy claim they can treat Paul's son. In Situation 1 (S_1), John is the first who meets Paul and his son. In Situation 2 (S_2), the first to meet them is Robert; and in Situation 3 (S_3), the first to meet them is Jimmy.

S_1. Paul has reason to think that John is an excellent physician, therefore he trusts John;

S_2. Paul knows that Robert is an ex-physician. Paul has motivations to think that the reason why Robert does not practice medicine anymore is he failed several times when treating people in a situation similar to that of his son. In this case, the rational behavior would be to distrust Robert;

S_3. Paul does not know Jimmy who introduces himself as a physician. In this case trusting or distrusting Jimmy depends on Paul's internal factors like his competence in judging the truth of incoming information, his state of spirit (optimistic, pessimistic, or pragmatic), etc. For example, if Paul is wary and pessimistic, he will distrust Jimmy; instead, if Paul is optimistic and does not perceive malicious purposes from somebody he does not know, he will accept Jimmy's help.

We can notice that in S_3, Paul has lack of positive and negative evidence. The fact is that Paul neither trusts nor distrusts Jimmy. He just does not know him.

This case shows that trust is not always the complement of distrust. Indeed, here we have $trust\ degree = 0$; $distrust\ degree = 0$, and we do not have $trust \neq 1 - distrust$. In this particular case the doubt (hesitation degree) is maximal, i.e., 1. It means that Paul behavior can depend on uncertain trust values going from completely trusting Jimmy to completely distrusting Jimmy.

We can distinguish information sources thanks to the extent of negative and positive evidence we dispose of for each source. Let (τ, δ) be the trustworthiness of a source for an agent.

Definition 2. *A source is said to be* helpful *if* $\tau > \delta$, malicious *if* $\tau < \delta$, neutral *if* $\tau = \delta \neq 0$. *When* $\tau = \delta = 0$, *the source is said to be* unknown.

Definition 3 (Comparing Sources). *Let* s_1 *and* s_2 *be two sources with trustworthiness scores of* (τ_1, δ_1) *and* (τ_2, δ_2) *respectively. Source* s_1 *is said to be more trustworthy than source* s_2, *noted* $s_1 \succeq s_2$, *if and only if* $\tau_1 \geq \tau_2$ *and* $\delta_1 \leq \delta_2$.

This is a partial order in the sense that it is not always possible to compare two trustworthiness scores.

Working Example. John thinks his house has become too small for his growing family and would like to buy a larger one. Of course, John wants to spend as little money as possible. A friend who works in the real estate industry tells John prices are poised to go down. Then John reads in a newspaper that the real estate market is weak and prices are expected to go down. Therefore, John's desire is to wait for prices to lower before buying. However, John later meets a real estate agent who has an interesting house on sale, and the agent tells him to hurry up, because prices are soaring. On the way home, John hears a guy on the bus saying his cousin told him prices of real estate are going up.

To sum up, John got information from four sources with different scores. The first source is friendly and competent; therefore, its score is $(1, 0)$. The second is supposedly competent and hopefully independent: therefore, its score might be something like $(\frac{1}{2}, \frac{1}{4})$. The third source is unknown, but has an obvious conflict of interest; therefore John assigns it a score of $(0, 1)$. Finally, the guy on the bus is a complete stranger reporting the opinion of another complete stranger. Therefore, its score cannot be other than $(0, 0)$.

We all have a basic intuition, suggested by common sense, of how information from these sources should be accounted for to generate a goal (buy now vs. wait for prices to go down) and planning actions accordingly. Below, we provide a formalization of the kinds of deliberations a rational agent is expected to make in order to deal with information scored with trust and distrust degrees.

4 Representing Graded Beliefs, Desires and Goals

An agent's belief is a piece of information that the agent believes in. An agents's desire is something (not always material) that the agent would like to possess or perform.

Here, we do not distinguish between what is positively desired and what is not rejected by explicitly considering positive and negative desires like it is done by Casali and colleagues [3]. This is an interesting point that we will consider in a future work. Nevertheless, our "desires" can be regarded as positive desires.

Desires (or motivations) are necessary but not sufficient conditions for action. When a desire is met by other conditions that make it possible for an agent to act, that desire becomes a *goal*. Therefore, given this technical definition of a desire, all goals are desires, but not all desires are goals. The main distinction we made here between desires and goals is in line with the one made by Thomason [23] and other authors: goals are required to be consistent whereas desires need not be.

4.1 Basic Notions

In this section, we present the main aspects of the adopted formalism.

Definition 4 (Language). *Let $\mathcal{A}$ be a set of atomic propositions and let $\mathcal{L}$ be the propositional language such that $\mathcal{A} \cup \{\top, \bot\} \subseteq \mathcal{L}$, and, $\forall \phi, \psi \in \mathcal{L}$, $\neg\phi \in \mathcal{L}$, $\phi \wedge \psi \in \mathcal{L}$, $\phi \vee \psi \in \mathcal{L}$.*

$\Omega = \{0,1\}^{\mathcal{A}}$ is the set of all possible interpretations on $\mathcal{A}$. An interpretation $\mathcal{I} \in \Omega$ is a function $\mathcal{I} : \mathcal{A} \rightarrow \{0,1\}$ assigning a truth value to every atomic proposition and, by extension, to all formulas in $\mathcal{L}$.

Representing Graded Beliefs. As convincingly argued by Dubois and Prade [12,13], a *belief* can be regarded as a necessity degree. A cognitive state of an agent can then be modeled by a normalized possibility distribution π:

$$\pi : \Omega \rightarrow [0,1]. \tag{3}$$

$\pi(\mathcal{I})$ is the possibility degree of interpretation $\mathcal{I}$. It represents the plausibility order of the possible word situation represented by interpretation $\mathcal{I}$. If the agent deems more plausible the world $\mathcal{I}_1$ than $\mathcal{I}_2$, then $\pi(\mathcal{I}_1) \geq \pi(\mathcal{I}_2)$.
The notation $[\phi]$ denotes the set of all models of a formula $\phi \in \mathcal{L}$:

$$[\phi] = \{\mathcal{I} \in \Omega : \mathcal{I} \models \phi\}.$$

Definition 5 (Possibility of a world situation). *The extent to which the agent considers ϕ as possible, $\Pi([\phi])$, is given by:*

$$\Pi([\phi]) = \max_{\mathcal{I} \in [\phi]} \{\pi(\mathcal{I})\}. \tag{4}$$

A desire-generation rule is defined as follows:

Definition 6 (Desire-Generation Rule). *A desire-generation rule R is an expression of the form $\beta_R, \psi_R \Rightarrow_D^+ d$, where $\beta_R, \psi_R \in \mathcal{L}$, and $d \in \{a, \neg a\}$ with $a \in \mathcal{A}$. The unconditional counterpart of this rule is $\alpha \Rightarrow_D^+ d$, which means that the agent (unconditionally) desires d to degree α.*

Intuitively this means: "an agent desires d as much as it believes β_R and desires ψ_R.

Given a desire-generation rule R, we shall denote $\mathrm{rhs}(R)$ the literal on the right-hand side of R.

Example (continued). John's attitude towards buying a larger house may be described as follows:

$$\begin{aligned} R_1 &: \quad \text{need\_larger\_house}, \top \Rightarrow_D^+ \text{buy\_house}, \\ R_2 &: \neg\text{prices\_down}, \text{buy\_house} \Rightarrow_D^+ \neg\text{wait}. \\ R_3 &: \quad \text{prices\_down}, \text{buy\_house} \Rightarrow_D^+ \text{wait}. \end{aligned}$$

4.2 Agent's State

The state of an agent is completely described by a triple $\mathcal{S} = \langle \mathcal{B}, \mathcal{R}_J, \mathcal{J} \rangle$, where

- $\mathcal{B}$ is the agent's belief set induced by a possibility distribution π;
- $\mathcal{R}_J$ is a set of desire-generation rules, such that, for each desire d, $\mathcal{R}_J$ contains at most one rule of the form $\alpha \Rightarrow_D^+ d$;
- $\mathcal{J}$ is a fuzzy set of literals.

$\mathcal{B}$ is the agent's belief set induced by a possibility distribution π (Equation 3); the the degree to which an agent believes ϕ is given by Equation 5. $\mathcal{R}_J$ contains the rules which generate desires from beliefs and other desires (subdesires). $\mathcal{J}$ contains all literals (positive and negative form of atoms in $\mathcal{A}$) representing desires which may be deduced from the agent's desire-generation rules. We suppose that an agent can have inconsistent desires, i.e., for each desire d we can have $\mathcal{J}(d) + \mathcal{J}(\neg d) > 1$.

4.3 Semantics of Belief and Desire Formulas

The semantics of belief and desire formulas in $\mathcal{L}$ are the following.

Definition 7 (Graded belief and desires formulas). *Let $\mathcal{S} = \langle \mathcal{B}, \mathcal{J}, \mathcal{R}_{\mathcal{J}} \rangle$ be the state of the agent, and ϕ be a formula, the degree to which the agent believes ϕ is given by:*

$$\mathcal{B}(\phi) = N([\phi]) = 1 - \Pi([\neg\phi]). \tag{5}$$

Straightforward consequences of the properties of possibility and necessity measures are that $\mathcal{B}(\phi) > 0 \Rightarrow \mathcal{B}(\neg\phi) = 0$ and

$$\mathcal{B}(\top) = 1, \tag{6}$$
$$\mathcal{B}(\bot) = 0, \tag{7}$$
$$\mathcal{B}(\phi \wedge \psi) = \min\{\mathcal{B}(\phi), \mathcal{B}(\psi)\}, \tag{8}$$
$$\mathcal{B}(\phi \vee \psi) \geq \max\{\mathcal{B}(\phi), \mathcal{B}(\psi)\}. \tag{9}$$

If ϕ is a literal, $\mathcal{J}(\phi)$ is directly given by the state of the agent. Instead, the desire degree of non-literal propositions is given by:

$$\mathcal{J}(\neg\phi) = 1 - \mathcal{J}(\phi), \tag{10}$$
$$\mathcal{J}(\phi \wedge \psi) = \min\{\mathcal{J}(\phi), \mathcal{J}(\psi)\}, \tag{11}$$
$$\mathcal{J}(\phi \vee \psi) = \max\{\mathcal{J}(\phi), \mathcal{J}(\psi)\}. \tag{12}$$

Note that since $\mathcal{J}$ needs not be consistent, the De Morgan laws do not hold, in general, for desire formulas.

Definition 8 (Degree of Activation of a Rule). *Let R be a desire-generation rule. The degree af activation of R, $\mathrm{Deg}(R)$, is given by*

$$\mathrm{Deg}(R) = \min(\mathcal{B}(\beta_R), \mathcal{J}(\psi_R))$$

and for its unconditional counterpart $R = \alpha \Rightarrow_D^+ d$: $\mathrm{Deg}(R) = \alpha$.

Definition 9 (Degree of Justification). *The degree of justification of desire d is defined as*

$$\mathcal{J}(d) = \max_{R \in \mathcal{R}_{\mathcal{J}} : \mathrm{rhs}(R) = d} \mathrm{Deg}(R).$$

This represents how rational it is for an agent to desire d.

Example (continued). John's initial state may be described by a possibility distribution π on

$$\Omega = \left\{ \begin{array}{l} \mathcal{I}_0 = \{\text{need\_larger\_house} \mapsto 0, \text{prices\_down} \mapsto 0\}, \\ \mathcal{I}_1 = \{\text{need\_larger\_house} \mapsto 0, \text{prices\_down} \mapsto 1\}, \\ \mathcal{I}_2 = \{\text{need\_larger\_house} \mapsto 1, \text{prices\_down} \mapsto 0\}, \\ \mathcal{I}_3 = \{\text{need\_larger\_house} \mapsto 1, \text{prices\_down} \mapsto 1\} \end{array} \right\},$$

such that

$$\pi(\mathcal{I}_0) = 0, \quad \pi(\mathcal{I}_1) = 0, \quad \pi(\mathcal{I}_2) = 1, \quad \pi(\mathcal{I}_3) = 1,$$

whereby

$$\begin{aligned} \mathcal{B}(\text{need\_larger\_house}) &= 1 - \max\{\pi(\mathcal{I}_0), \pi(\mathcal{I}_1)\} = 1, \\ \mathcal{B}(\neg\text{need\_larger\_house}) &= 1 - \max\{\pi(\mathcal{I}_2), \pi(\mathcal{I}_3)\} = 0, \\ \mathcal{B}(\text{prices\_down}) &= 1 - \max\{\pi(\mathcal{I}_0), \pi(\mathcal{I}_2)\} = 0, \\ \mathcal{B}(\neg\text{prices\_down}) &= 1 - \max\{\pi(\mathcal{I}_1), \pi(\mathcal{I}_3)\} = 0. \end{aligned}$$

Therefore, the set of John's justified desires will be

$$\begin{aligned} \mathcal{J}(\text{buy\_house}) &= 1 \quad \text{(because of Rule } R_1\text{)}, \\ \mathcal{J}(\neg\text{buy\_house}) &= 0 \quad \text{(no justifying rule)}, \\ \mathcal{J}(\text{wait}) &= 0 \quad \text{(because of Rule } R_3\text{)}, \\ \mathcal{J}(\neg\text{wait}) &= 0 \quad \text{(because of Rule } R_2\text{)}. \end{aligned}$$

5 Belief Change

Here, we discuss and compare three possible extensions of a trusted-based operator akin to the one proposed in [7] to deal with bipolar trust and distrust degrees. To begin with, we define a basic belief change operator based on trust.

5.1 Trust-Based Belief Change Operator

Here, we suppose that a source of information may be considered trusted to a certain extent. This means that its membership degree to the fuzzy set of trusted sources is $\tau \in [0, 1]$. Let $\phi \in \mathcal{L}$ be incoming information from a source trusted to degree τ. The belief change operator is defined as follows:

Definition 10 (Belief Change Operator *). *The possibility distribution π' which induces the new belief set $\mathcal{B}'$ after receiving information ϕ is computed from possibility distribution π relevant to the previous belief set $\mathcal{B}$ ($\mathcal{B}' = \mathcal{B} * \frac{\tau}{\phi}$) as follows: for all interpretation $\mathcal{I}$,*

$$\pi'(\mathcal{I}) = \bar{\pi}'(\mathcal{I}) / \max_{\mathcal{I}} \{\bar{\pi}'(\mathcal{I})\}, \tag{13}$$

where

$$\bar{\pi}'(\mathcal{I}) = \begin{cases} \pi(\mathcal{I}), & \text{if } \mathcal{I} \models \phi \text{ and } \mathcal{B}(\neg\phi) < 1; \\ \tau, & \text{if } \mathcal{I} \models \phi \text{ and } \mathcal{B}(\neg\phi) = 1; \\ \pi(\mathcal{I}) \cdot (1 - \tau), & \text{if } \mathcal{I} \not\models \phi. \end{cases} \tag{14}$$

Notice that Equation 13 guarantees that π' is a normalized possibility distribution. The condition $\mathcal{B}(\neg\phi) < 1$ in Equation 14 is equivalent to $\exists \mathcal{I}' : \mathcal{I}' \models \phi \Rightarrow \pi(\mathcal{I}') > 0$, i.e., $\Pi([\phi]) > 0$; likewise, the condition $\mathcal{B}(\neg\phi) = 1$ is equivalent to $\Pi([\phi]) = 0$, which implies $\pi(\mathcal{I}) = 0$. Therefore, the second case in Equation 14 provides for the *revision* of beliefs that are in contradiction with new information ϕ. In general, the operator treats new information ϕ in the negative sense: being told ϕ denies the possibility of world situations where ϕ is false (third case of Equation 14). The possibility of world situations where ϕ is true may only increase due to normalization (Equation 13) or revision (second case of Equation 14).

In the following, we present three alternatives for extending the belief change operator $*$. In such extensions, we suppose that a source of information may be considered trusted to a degree $\tau \in [0, 1]$ and/or distrusted to a degree $\delta \in [0, 1]$; τ and δ are respectively the source's membership degrees in the fuzzy sets of trusted and distrusted sources.

5.2 Open-Minded Belief Change Operator

This operator represents the changes in the beliefs of an agent which is both optimistic and does not perceive malicious purposes from neutral sources ("An optimistic agent discerns (him)herself as luckier" [17]). The proposed operator provides a formal representation of how an agent which gives the benefit of the doubt to the sources could change its beliefs when new information is received. More precisely, the following definition illustrates the attitude of an open-minded agent when choosing which among the possible trust degrees in $[\tau, 1 - \delta]$ to consider as its trust degree.

Definition 11 (Open-Minded Operator). *Let ϕ be the incoming information with the trustworthiness score (τ, δ). An open-minded belief change operator $*_m$ can be defined as follows:*

$$\mathcal{B} *_m \frac{(\tau, \delta)}{\phi} = \mathcal{B} * \frac{\tau + (h/2)}{\phi}. \tag{15}$$

As we can see, such an agent chooses a degree of trust which is proportional to the degree of hesitation. Due to its optimism, the greater the hesitation, the higher the adopted trust degree.

Observation 1. *If $h = 0$, then, by applying the $*_m$ operator, information ϕ coming from any source with trustworthiness degree (τ, δ), is perceived as trusted with degree τ: $\mathcal{B} *_m \frac{(\tau,\delta)}{\phi} = \mathcal{B} * \frac{\tau}{\phi}$.*

Observation 2. *By applying the $*_m$ operator, information coming from a neutral source is considered as half-trusted.*

Indeed, $\forall\, \alpha \in [0, 1/2]$[2] we have: $\mathcal{B} *_m \frac{(\alpha,\alpha)}{\phi} = \mathcal{B} * \frac{1/2}{\phi}$. In particular, when a piece of information comes from a completely unknown source, (i.e., with a trustworthiness score equal to $(0, 0)$), the agent considers that information anyway by giving it a half degree of trust.

Observation 3. *By applying the $*_m$ operator, information is not considered at all only in the cases in which it comes from a completely distrusted source, i.e., a source with a trustworthiness score equal to $(0, 1)$. Benefit of the doubt is given in all the other cases.*

Example (continued). By using the open-minded change operator, John's initial beliefs would change to

$$\begin{aligned}\mathcal{B}' &= \mathcal{B} *_m \frac{(1,0)}{\phi} *_m \frac{(1/2,1/4)}{\phi} *_m \frac{(0,1)}{\neg\phi} *_m \frac{(0,0)}{\neg\phi} \\ &= \mathcal{B} * \frac{1}{\phi} * \frac{5/8}{\phi} * \frac{0}{\neg\phi} * \frac{1/2}{\neg\phi},\end{aligned}$$

where $\phi =$ prices_down. This is how the possibility distribution over belief interpretations changes, given that $\mathcal{I}_1, \mathcal{I}_3 \models \phi$ and $\mathcal{I}_0, \mathcal{I}_2 \models \neg\phi$:

	$\pi(\mathcal{I}_0)$	$\pi(\mathcal{I}_1)$	$\pi(\mathcal{I}_2)$	$\pi(\mathcal{I}_3)$
initial	0	0	1	1
$*\frac{1}{\phi}$	0	0	0	1
$*\frac{5/8}{\phi}$	0	0	0	1
$*\frac{0}{\neg\phi}$	0	0	0	1
$*\frac{1/2}{\neg\phi}$	1	0	1	1

This yields: $\mathcal{B}'(\text{prices\_down}) = \mathcal{B}'(\neg\text{prices\_down}) = 0$. Therefore, an open-minded John would not change the justification degree of his desires, that is, his justification to buy a house would still be full, while he would be unresolved whether to wait or not, for neither desire would be justified.

5.3 Wary Belief Change Operator

Here, we present a belief change operator which illustrates the attitude of a conservative (pessimistic) agent which does not give the benefit of the doubt to unknown sources and perceives information coming from malicious sources as false (to some extent, depending on the degree of distrust).“A pessimistic agent discerns (him)herself as unluckier”.

Definition 12 (Effective Trust/Distrust). *Let (τ, δ) be the trustworthiness score of a helpful source (malicious source). The degre of effective trust τ_e (effective distrust δ_e) is given by $\tau_e = \tau - \delta \in (0, 1]$ ($\delta_e = \delta - \tau \in (0, 1]$).*

[2] by definition, i.e. because $\tau + \delta \leq 1$, the highest value τ and δ can take up in case of equality is $1/2$.

Definition 13 (Wary Operator). *Let ϕ be incoming information with trustworthiness score (τ, δ). A wary belief change operator $*_w$ can be defined as follows:*

$$\mathcal{B} *_w \frac{(\tau,\delta)}{\phi} = \begin{cases} \mathcal{B} * \frac{\tau_e}{\phi} & \text{if } \tau_e > 0 \text{ and } h \neq 0; \\ \mathcal{B} * \frac{\delta_e}{\neg\phi} & \text{if } \delta_e > 0 \text{ and } h \neq 0; \\ \mathcal{B} & \text{if } \tau = \delta. \end{cases} \tag{16}$$

Observation 4. *Whereas in the previous generalization proposal the special case that falls back to operator $*$ is when $\delta = 1 - \tau$, here we start from the assumption that operator $*$ applies to the special case where $\delta = 0$.*

Observation 5. *If the agent has motivations for (effectively) distrusting, with degree α, the source of information from which comes the new piece of information ϕ, by applying the operator $*_w$, it will trust its opposite $\neg\phi$, with degree α.*

Observation 6. *In both cases of unknown sources (i.e., trustworthiness score equal to $(0,0)$), or neutral sources (i.e., trustworthiness score of the form (α,α)), applying $*_w$ will not change at all the agent degrees of beliefs.*

Example (continued). By using the wary change operator, John's initial beliefs would change to

$$\begin{aligned} \mathcal{B}' &= \mathcal{B} *_w \frac{(1,0)}{\phi} *_w \frac{(1/2,1/4)}{\phi} *_w \frac{(0,1)}{\neg\phi} *_w \frac{(0,0)}{\neg\phi} \\ &= \mathcal{B} * \frac{1}{\phi} * \frac{1/4}{\phi} * \frac{1}{\phi} * \frac{0}{\neg\phi}, \end{aligned}$$

where $\phi =$ prices_down. This is how the possibility distribution over belief interpretations changes, given that $\mathcal{I}_1, \mathcal{I}_3 \models \phi$ and $\mathcal{I}_0, \mathcal{I}_2 \models \neg\phi$:

	$\pi(\mathcal{I}_0)$	$\pi(\mathcal{I}_1)$	$\pi(\mathcal{I}_2)$	$\pi(\mathcal{I}_3)$
initial	0	0	1	1
$*\frac{1}{\phi}$	0	0	0	1
$*\frac{1/4}{\phi}$	0	0	0	1
$*\frac{1}{\phi}$	0	0	0	1
$*\frac{0}{\neg\phi}$	0	0	0	1

This yields $\mathcal{B}'(\text{prices\_down}) = 1$ and $\mathcal{B}'(\neg\text{prices\_down}) = 0$. Therefore, a wary John would increase the justification degree of the desire to wait for prices to go down before buying. The justification of waiting is now equal to that of buying a house.

5.4 Content-Based Belief Change Operator

Neither of the previous two proposed operator extensions attempts to consider the content of incoming information as well as the agent competence in judging its truth.

Some experiments have shown, however, that this consideration can help when evaluating the truth of new information. Fullam and Barber [14] showed that integrating

both policies for information valuation based on characteristics of the information and the sources providing it yields significant improvement in belief accuracy and precision over no-policy or single-policy belief revision.

Experiments on human trust and distrust of information in the context of a military sensemaking task have concluded that people tend to trust information from an unknown source (whose prototypical case is $\tau = \delta = 0$) to the extent that it does not contradict their previous beliefs [19]. The basic rationale for this behavior appears to be that people trust themselves, if anybody.

The third proposed extension of the belief change operator intends to model this type of behavior.

Definition 14 (Content-Based Operator). *Let ϕ be incoming information with trustworthiness score (τ, δ). A content-based belief change operator $*_c$ can be defined as follows:*

$$\mathcal{B} *_c \frac{(\tau, \delta)}{\phi} = \mathcal{B} * \frac{\tau + h \cdot \mathcal{B}(\phi)}{\phi}. \tag{17}$$

Observation 7. *By applying operator $*_c$, the only case in which information will be completely rejected is when the source is fully distrusted.*

Observation 8. *By applying operator $*_c$, the only case in which the information content does not influence the agent's beliefs is when $h = 0$.*

Example (continued). By using the content-based change operator, John's initial beliefs would change to

$$\begin{aligned}\mathcal{B}' &= \mathcal{B} *_c \frac{(1,0)}{\phi} *_c \frac{(1/2, 1/4)}{\phi} *_c \frac{(0,1)}{\neg\phi} *_c \frac{(0,0)}{\neg\phi} \\ &= \mathcal{B} * \frac{1}{\phi} * \frac{3/4}{\phi} * \frac{0}{\neg\phi} * \frac{0}{\neg\phi},\end{aligned}$$

where $\phi =$ prices_down. This is how the possibility distribution over belief interpretations changes, given that $\mathcal{I}_1, \mathcal{I}_3 \models \phi$ and $\mathcal{I}_0, \mathcal{I}_2 \models \neg\phi$:

	$\pi(\mathcal{I}_0)$	$\pi(\mathcal{I}_1)$	$\pi(\mathcal{I}_2)$	$\pi(\mathcal{I}_3)$
initial	0	0	1	1
$*\frac{1}{\phi}$	0	0	0	1
$*\frac{3/4}{\phi}$	0	0	0	1
$*\frac{0}{\neg\phi}$	0	0	0	1
$*\frac{0}{\neg\phi}$	0	0	0	1

This yields: $\mathcal{B}'(\text{prices\_down}) = 1$ and $\mathcal{B}'(\neg\text{prices\_down}) = 0$. Therefore, in this case, John would behave like a wary John.

5.5 Further Observations

Observation 9. *By applying the operators $*_m$ and $*_c$, a completely distrusted piece of information ($\delta = 1$, $\tau = 0$ and $h = 0$) is not considered at all.*

When there is no doubt about the trust degree, both the open-minded and content-based agents behave like a trust-based agent.

Observation 10. *If there is no hesitation, i.e., if* $h = 0$*,*

$$\mathcal{B} *_m \frac{(\tau, \delta)}{\phi} = \mathcal{B} *_c \frac{(\tau, \delta)}{\phi} = \mathcal{B} * \frac{\tau}{\phi}.$$

Observation 11. *By using the open minded operator or the content-based operator, the more different not fully distrusted sources confirm formula* ϕ*, the more* ϕ *is believed.*

Observation 12. *By using a wary operator, the more different and:*

- *helpful sources confirm a belief by providing the same information, the more believed is the information provided;*
- *malicious sources confirm a belief by providing the same information, the less believed is the information provided.*

6 Desire and Goal Change

Belief change may induce changes in the justification degree of some desires/goals. Actually, this is the only way to modify the agent goals from the outside (external reason) [4]. As we have seen in the previous sections, different kinds of agent may have different behaviors (*open-minded*, *wary*, or *content-based*, for example) when changing their beliefs on the basis of new incoming information. This may result in different belief behaviours for different agents in the same situation and, as a consequence, in different desire/goal sets. To account for the changes in the desire set caused by belief change, one has to recursively [7]:

(i) calculate for each rule $R \in \mathcal{R}_\mathcal{J}$ its new activation degree by considering $\mathcal{B}'$ and
(ii) update the justification degree of all desires in its right-hand side ($\mathrm{rhs}(R)$).

Desires may also change for internal reasons. This is represented by the insertion of a new desire-generation rule in $\mathcal{R}_\mathcal{J}$ or the retraction of an existing rule. The new fuzzy set of justified desires, $\mathcal{J}'$, is computed as follow:

(i) calculate for each rule $R \in \mathcal{R}_\mathcal{J}$ its new activation degree by considering the fact that a rule is inserted or retracted and
(ii) update the justification degree of all desires in the right-hand side of the rules in $\mathcal{R}_\mathcal{J}$.

Goals serve a dual role in the deliberation process, capturing aspects of both *intentions* [6] and *desires* [25]. The main point about desires is that we expect a rational agent to try and manipulate its surrounding environment to fulfill them. In general, considering a problem $\mathcal{P}$ to solve, not all generated desires can be adopted at the same time, especially when they are not feasible at the same time. We assume we dispose of a $\mathcal{P}$-dependent function $\mathcal{F}_\mathcal{P}$ wich, given a possibility distribution inducing a set of graded beliefs $\mathcal{B}$ and a fuzzy set of desires $\mathcal{J}$, returns a degree γ which corresponds to the certainty degree of the most certain feasible solution found. We may call γ the *degree of feasibility* of $\mathcal{J}$ given $\mathcal{B}$, i.e., $\mathcal{F}_\mathcal{P}(\mathcal{J}|\mathcal{B}) = \gamma$.

Definition 15 (γ-Goal Set). *A γ-goal set, with $\gamma \in [0,1]$, in state $\mathcal{S}$ is a fuzzy set of desires $\mathcal{G}$ such that:*

1. *$\mathcal{G}$ is justified: $\mathcal{G} \subseteq \mathcal{J}$, i.e., $\forall d \in \{a, \neg a\}$, $a \in \mathcal{A}$, $\mathcal{G}(d) \leq \mathcal{J}(d)$;*
2. *$\mathcal{G}$ is γ-feasible: $\mathcal{F}_{\mathcal{P}}(\mathcal{G}|\mathcal{B}) \geq \gamma$;*
3. *$\mathcal{G}$ is consistent: $\forall d \in \{a, \neg a\}$, $a \in \mathcal{A}$, $\mathcal{G}(d) + \mathcal{G}(\neg d) \leq 1$.*

In general, given a fuzzy set of desires $\mathcal{J}$, there may be more than one possible γ-goal sets $\mathcal{G}$. However, a rational agent in state $\mathcal{S} = \langle \mathcal{B}, \mathcal{J}, \mathcal{R}_{\mathcal{J}} \rangle$, for practical reasons, may need to elect one precise set of goals, $\mathcal{G}^*$, to pursue, which depends on $\mathcal{S}$.

In that case, a goal election function should be defined. Let us call G_γ the function which maps a state $\mathcal{S}$ into the γ-goal set elected by a rational agent in state $\mathcal{S}$: $\mathcal{G}^* = G_\gamma(\mathcal{S})$.

The choice of one goal set over the others may be based on a preference relation on desire sets. Therefore, G_γ must be such that:

- $(\mathbf{G1})$ $\forall \mathcal{S}$, $G_\gamma(\mathcal{S})$ is a γ-goal set;
- $(\mathbf{G2})$ $\forall \mathcal{S}$, if $\mathcal{G}$ is a γ-goal set, then $G_\gamma(\mathcal{S})$ is preferred at least as $\mathcal{G}$, or more.

(G1) requires that a goal election function G_γ does indeed return a γ-goal set; while **(G2)** requires that the γ-goal set returned by function G_γ be "optimal", i.e., that a rational agent always selects one of the most preferrable γ-goal sets.

7 Conclusion

The issue of how to deal with independently and explicitly given trust and distrust degrees of information sources within the context of goal generation has been approached by generalizing a trusted-based belief change operator.

The three proposed alternative extensions have different scopes. The open-minded operator makes sense in a collaborative environment, where all sources of information intend to be helpful, except that, perhaps, some of them may lack the knowledge needed to help. The wary operator is well suited to contexts where competition is the main theme and the agents are utility-driven participants in a zero-sum game, where a gain for an agent is a loss for its counterparts. The content-based operator is aimed at mimicking the usual way people change their beliefs.

References

1. Atanassov, K.T.: Intuitionistic fuzzy sets. Fuzzy Sets Syst. 20(1), 87–96 (1986)
2. Ben-Naim, J., Prade, H.: Evaluating trustworthiness from past performances: Interval-based approaches. In: Greco, S., Lukasiewicz, T. (eds.) SUM 2008. LNCS (LNAI), vol. 5291, pp. 33–46. Springer, Heidelberg (2008)
3. Casali, A., Godo, L., Sierra, C.: Graded BDI models for agent architectures. In: Leite, J., Torroni, P. (eds.) CLIMA 2004. LNCS (LNAI), vol. 3487, pp. 18–33. Springer, Heidelberg (2005)
4. Castelfranchi, C.: Reasons: Belief support and goal dynamics. Mathware and Soft Computing 3, 233–247 (1996)

5. Castelfranchi, C., Falcone, R., Pezzulo, G.: Trust in information sources as a source for trust: a fuzzy approach. In: Proceedings of AAMAS 2003, pp. 89–96 (2003)
6. Cohen, P.R., Levesque, H.J.: Intention is choice with commitment. Artif. Intell. 42(2-3), 213–261 (1990)
7. da Costa Pereira, C., Tettamanzi, A.: Goal generation and adoption from partially trusted beliefs. In: Proceedings of ECAI 2008, pp. 453–457. IOS Press, Amsterdam (2008)
8. Dastani, M., Herzig, A., Hulstijn, J., Van Der Torre, L.: Inferring trust. In: Leite, J., Torroni, P. (eds.) CLIMA 2004. LNCS (LNAI), vol. 3487, pp. 144–160. Springer, Heidelberg (2005)
9. De Cock, M., da Silva, P.P.: A many valued representation and propagation of trust and distrust. In: Bloch, I., Petrosino, A., Tettamanzi, A.G.B. (eds.) WILF 2005. LNCS (LNAI), vol. 3849, pp. 114–120. Springer, Heidelberg (2006)
10. Deschrijver, G., Kerre, E.E.: On the relationship between some extensions of fuzzy set theory. Fuzzy Sets Syst. 133(2), 227–235 (2003)
11. Dignum, F., Kinny, D.N., Sonenberg, E.A.: From desires, obligations and norms to goals. Cognitive Science Quarterly Journal 2(3-4), 407–427 (2002)
12. Dubois, D., Prade, H.: Possibility theory, probability theory and multiple-valued logics: A clarification. Annals of Mathematics and Artificial Intelligence 32(1-4), 35–66 (2001)
13. Dubois, D., Prade, H.: An introduction to bipolar representations of information and preference. Int. J. Intell. Syst. 23(8), 866–877 (2008)
14. Fullam, K.K., Barber, K.S.: Using policies for information valuation to justify beliefs. In: AAMAS 2004, pp. 404–411. IEEE Computer Society, Los Alamitos (2004)
15. Griffiths, N.: A fuzzy approach to reasoning with trust, distrust and insufficient trust. In: Klusch, M., Rovatsos, M., Payne, T.R. (eds.) CIA 2006. LNCS (LNAI), vol. 4149, pp. 360–374. Springer, Heidelberg (2006)
16. Hansson, S.O.: Ten philosophical problems in belief revision. Journal of Logic and Computation 13(1), 37–49 (2003)
17. Jacquemet, N., Rullière, J.-L., Vialle, I.: Monitoring optimistic agents. Technical report, Université Paris 1 Sorbonne-Panthéon (2008)
18. Liau, C.-J.: Belief, information acquisition, and trust in multi-agent systems: a modal logic formulation. Artif. Intell. 149(1), 31–60 (2003)
19. McGuinness, B., Leggatt, A.: Information trust and distrust in a sensemaking task. In: Command and Control Research and Technology Symposium (2006)
20. McKnight, D.H., Chervany, N.L.: Trust and distrust definitions: One bite at a time. In: Proceedings of the workshop on Deception, Fraud, and Trust in Agent Societies, pp. 27–54. Springer, Heidelberg (2001)
21. Intersex Society of North America, http://www.isna.org/
22. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In: KR 1991, pp. 473–484 (1991)
23. Thomason, R.H.: Desires and defaults: A framework for planning with inferred goals. In: Proceedings of KR 2000, pp. 702–713 (2000)
24. Türksen, I.B.: Interval valued fuzzy sets based on normal forms. Fuzzy Sets Syst. 20(2), 191–210 (1986)
25. Wellman, M.P., Doyle, J.: Preferential semantics for goals. In: Proceedings of AAAI 1991, vol. 2, pp. 698–703 (1991)
26. Zadeh, L.A.: Fuzzy sets. Information and Control 8, 338–353 (1965)
27. Zadeh, L.A.: Fuzzy sets as a basis for a theory of possibility. Fuzzy Sets and Systems 1, 3–28 (1978)

Monitoring Directed Obligations with Flexible Deadlines: A Rule-Based Approach

Henrique Lopes Cardoso and Eugénio Oliveira

LIACC, DEI / Faculdade de Engenharia, Universidade do Porto
R. Dr. Roberto Frias, 4200-465 Porto, Portugal
{hlc,eco}@fe.up.pt

Abstract. Real-world business relationships have an essentially cooperative nature. However, when modeling contractual norms using normative multi-agent systems, it is typical to give norms a strict and domain independent semantics. We argue that in B2B contract enactment cooperation should be taken into account when modeling contractual commitments through obligations. We introduce an approach to model such commitments based on directed obligations with time windows. Our proposal is based on authorizations granted at specific states of an obligation lifecycle model, made possible by handling deadlines in a flexible way. We formalize such obligations using linear temporal logic and provide an implementation to their semantics using a set of monitoring rules employed in a forward-chaining inference engine. We show, through experimentation, the correctness of the obtained monitoring tool in different contract enactment situations.

1 Introduction

Real-world business relationships have an essentially cooperative nature. When considering B2B Virtual Organizations, different enterprises share their own competences and skills in a regulated way, through commitments expressed as norms in contracts. The essence of such contracts is commitment [1]: contracts provide a legally binding agreement including legal sanctions in case of failure to honor commitments. However, the importance of successfully proceeding with business demands for flexibility of operations: contractors should try to facilitate the compliance of their partners. This common goal of conducting business is based on the fact that group success also benefits each partner's private goals. These goals are not limited to the ongoing business relationship, but may also concern future opportunities that may arise.

Multi-agent systems have been used to address B2B settings, where agents represent different enterprises and engage in (automated) contract negotiations. Furthermore, software frameworks are being designed and engineered that try to provide normative environments [2, 3, 4, 5, 6] enabling monitoring and enforcement of contractual norms. Nevertheless, many approaches to normative multi-agent systems are abstracted away from their potential application domain. As such, deontic operators used to describe norms are typically modeled with a

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 51–67, 2010.

universal and domain independent semantics. For instance, deadline obligations are violated if the obliged action or state is not obtained before the deadline.

We argue that in some domains – such as in business contracts – such an approach is not desirable. For instance, the United Nations Convention on Contracts for the International Sale of Goods (CISG) [7] establishes what parties may do in case of deadline violations. In some cases they are allowed to fulfill their obligations after the deadline (Article 48), or even to extend deadlines with the allowance of their counterparties. Furthermore, a party may extend his counterparty's deadlines (Articles 47 and 63), which denotes a flexible and even cooperative facet of trade contracts.

In this paper we present and explore a different approach (in comparison with [8] [9] [10] [11]) to the use of obligations in agent-based contracts. Following a cooperative business enactment principle, we argue that obligations should be directed (as in [11]), and that deadlines should be flexible (as they seem to be in the real world [7]). In section 2 we motivate our research with insights from a real-world legislation, and we present an approach to model contractual obligations with time windows. Our approach (more deeply analyzed in [12]) is based on authorizations, and includes a new lifecycle for directed obligations with temporal restrictions. A formalization is given using linear temporal logic. Section 3 describes a normative environment for norm monitoring, and includes a rule-based inference approach to our obligation semantics. Section 4 presents an implementation of the system, including monitoring rules, an example contract, and illustration of the system's response in different contract enactment situations. Finally, section 5 concludes and discusses our developments in the light of other approaches in the literature.

2 Modeling Contractual Obligations

Norms in MAS have been used for modeling regulated environments for agents. Deontic operators – obligation, permission and prohibition – form the basis for such approaches. In our case, we find obligations to be particularly relevant in the scope of business contracts.

Approaches to model obligations in MAS that have an implementation in mind typically consider two attributes: the bearer of the obligation and the deadline. We may represent such an obligation as $O_b(f, d)$: a *deadline obligation* indicating that agent b (the *bearer* of the obligation) is obliged to bring about fact f (a state of affairs) before deadline d (either a time reference or more generally defined as a state of affairs).

When recalling the usual approach to model the semantics of deadline obligations, as well as when presenting our proposal, we will make use of linear temporal logic (LTL) [13], with a discrete time model. Let $x = (s_0, s_1, s_2, ...)$ be a timeline, defined as a sequence of states s_i. The syntax $x \models p$ reads that p is true in timeline x. We write x^k to denote state s_k of x, and $x^k \models p$ to mean that p is true at state x^k. We use a weak version of the *before* LTL operator B, where q is not mandatory: $x \models (p\ B\ q)$ *iff* $\exists_j(x^j \models p \wedge \forall_{k<j}(x^k \models \neg q))$.

The semantics of deadline obligations has been studied before (e.g. [8] [9]). The usual approach is to consider the following entailments:

- $O_b(f, d) \wedge (f\ B\ d) \models Fulf_b(f, d)$ — If the fact to bring about occurs before the deadline, the agent has *fulfilled* his obligation.
- $O_b(f, d) \wedge (d\ B\ f) \models Viol_b(f, d)$ — If the deadline occurs before the fact to bring about, the agent has *violated* his obligation.

The introduction of *Fulf* and *Viol* enables reasoning about the respective situations. The implementation of this semantics using forward-chaining rules has been studied in [9]. Although intuitive, this semantics is quite rigid in that violations are all defined in a universal way.

The analysis of contracts brings into discussion the notion of *directed obligations* [14]. An obligation $O_{b,c}(f)$ is seen as directed from agent b (the *bearer* responsible for fulfilling the obligation) to agent c (the *counterparty*). We interpret obligations of this kind as claims from counterparties to bearers (as in [11]): if b does not bring about f then c is *authorized* to react against b. Note that this reaction is discretionary, not mandatory.

In our approach we combine deadline obligations [8] with directed obligations [14] [11], in order to obtain a more precise definition of when it is that a counterparty may claim against the inability of a bearer to fulfill the obligation. We will motivate and formalize the notion of *directed deadline obligation* – $O_{b,c}(f, d)$: agent b is obliged towards agent c to bring about f before d. An extension of directed (contractual) obligations with temporal restrictions is also introduced in [10], but that approach is based on a rigid model of violations, in that they are automatically obtained at the deadline. In our approach deadlines have a distinct role in the semantics of obligations. We will introduce the notion of *deadline violation* (as opposed to obligation violation) in order to obtain a flexible approach to handle non-ideal situations: each deadline violation is different, as each may have a different impact on the ongoing business, and each occurs between a specific pair of agents with a unique trust relationship.

2.1 Directed Obligations with Time Windows

When specifying norms in contracts, deadline handling is central to define the semantics of contractual obligations. In order to motivate our approach, we take some inspiration from the United Nations Convention on Contracts for the International Sale of Goods (CISG) [7]. Some excerpts of this legislation are included in the following discussion.

> *Article 48: (1) [...] the seller may, even after the date for delivery, remedy at his own expense any failure to perform his obligations, if he can do so without unreasonable delay [...]; (2) If the seller requests the buyer to make known whether he will accept performance and the buyer does not comply with the request within a reasonable time, the seller may perform within the time indicated in his request. [...]*

This means that even though a deadline has been violated, the bearer may still be entitled to fulfill *the same* obligation. This kind of delay is also called a *grace period*: a period beyond a due date during which an obligation may be met without penalty or cancellation.

In fact, the successful enactment of a contract is dependent on the need to make contractual provisions performable in a flexible way:

> *Article 47: (1) The buyer may fix an additional period of time of reasonable length for performance by the seller of his obligations.*
>
> *Article 63: (1) The seller may fix an additional period of time of reasonable length for performance by the buyer of his obligations.*

These articles emphasize the need for flexible deadlines. Note that the counterparty's benevolence on conceding an extended deadline to the bearer does not prescribe a new obligation; instead, *the same* obligation may be fulfilled within a larger time window. Furthermore, it is also in the counterparty's best interest that this option is available, given the importance of reaching success in the performance of the contract.

In some other cases, a party may decide that the non-fulfillment of an obligation should be handled in a more strict way. The CISG convention specifies conditions for cancelling a contract in case of breach:

> *Article 49: (1) The buyer may declare the contract avoided: (a) if the failure by the seller to perform any of his obligations [...] amounts to a fundamental breach of contract; [...]; (2) However, in cases where the seller has delivered the goods, the buyer loses the right to declare the contract avoided unless he does so: (a) in respect of late delivery, within a reasonable time after he has become aware that delivery has been made; [...]*
>
> *Article 64: (1) The seller may declare the contract avoided: (a) if the failure by the buyer to perform any of his obligations [...] amounts to a fundamental breach of contract; [...]; (2) However, in cases where the buyer has paid the price, the seller loses the right to declare the contract avoided unless he does so: (a) in respect of late performance by the buyer, before the seller has become aware that performance has been rendered; [...]*

These articles allow contract termination in both non-performance and late performance cases. However, the second case is limited to the awareness of the offended party.

The deadline approach is often taken to be appropriate for specifying temporal restrictions on obligations. However, in certain cases a time window should be provided. In international trade transactions, for instance, storage costs may be relevant. Also, perishable goods should be delivered only when they are needed, not before.

> *Article 52: (1) If the seller delivers the goods before the date fixed, the buyer may take delivery or refuse to take delivery.*

Therefore, anticipated fulfillments are not always welcome. We find it necessary to include a variation of directed deadline obligations, to which we add a *liveline*: a time reference after which the obligation should be fulfilled. In this case we have $O_{b,c}(f, l, d)$: agent b is obliged towards agent c to bring about f between l (a liveline) and d (a deadline).

The intuitive semantics of directed deadline obligations and directed obligations with liveline and deadline are illustrated in Figures 1 and 2. The shaded areas represent the period of time within which the achievement of f will certainly bring a fulfillment of the obligation. The region to the left of l (Figure 2) entitles c to react if f is accomplished; also, regions to the right of d in both figures indicate that counterparty c is entitled to react if f is not accomplished. However, as long as no reaction is taken, b can still fulfill his obligation.

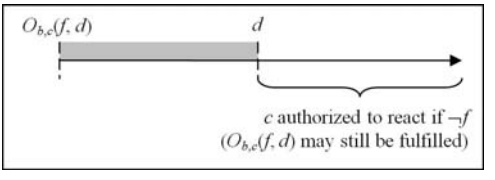

Fig. 1. Directed obligation with deadline

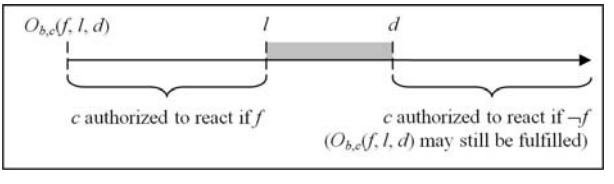

Fig. 2. Directed obligation with liveline and deadline

2.2 Formalization with LTL

Following the discussion above, in Table 1 we identify the possible states for an obligation, together with the elements we shall use to signal some of those states (when obtained, these elements are supposed to persist over time).

We now proceed to formalizing each type of obligation using LTL.

Directed Deadline Obligations. Figure 3 illustrates, by means of a state transition diagram, the lifecycle of directed deadline obligations. We take obligations as being prescribed from conditional norms; the confirmation of the norm's conditions will change the prescribed obligation's state from inactive to active. The obligation is also automatically pending, since it may be legitimately fulfilled right away. We set the obligation to have a violated deadline – $DViol_{b,c}(f, d)$ – when the deadline occurs before the obliged fact. The counterparty's reaction

Table 1. Obligation states

inactive	the obligation is not yet in effect, but will eventually be prescribed by a norm
active	the obligation was prescribed by a norm: $O_{b,c}(f,d)$ or $O_{b,c}(f,l,d)$
pending	the obligation may be fulfilled from now on
liveline violation	the fact being obliged has been brought ahead of time: $LViol_{b,c}(f,l,d)$
deadline violation	the fact being obliged should have been brought already: $DViol_{b,c}(f,d)$ or $DViol_{b,c}(f,l,d)$
fulfilled	the obligation was fulfilled: $Fulf_{b,c}(f,d)$ or $Fulf_{b,c}(f,l,d)$
violated	the obligation was violated and cannot be fulfilled anymore: $Viol_{b,c}(f,d)$ or $Viol_{b,c}(f,l,d)$

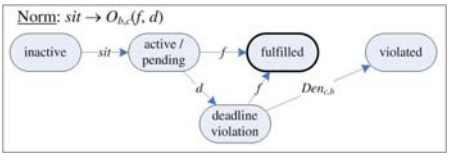

Fig. 3. Lifecycle of a directed deadline obligation

to a deadline violation will only change the obligation's state if the option is to deem the obligation as violated, by *denouncing* this situation. For this we introduce the element $Den_{c,b}(f,d)$, which is a denounce from agent c towards agent b regarding the failure of the latter to comply with his obligation to bring about f before d.

The lifecycle of directed deadline obligations is formalized as follows:

- $O_{b,c}(f,d) \wedge (f\ B\ d) \models Fulf_{b,c}(f,d)$
- $O_{b,c}(f,d) \wedge (d\ B\ f) \models DViol_{b,c}(f,d)$
- $DViol_{b,c}(f,d) \wedge (f\ B\ Den_{c,b}(f,d)) \models Fulf_{b,c}(f,d)$
- $DViol_{b,c}(f,d) \wedge (Den_{c,b}(f,d)\ B\ f) \models Viol_{b,c}(f,d)$

Directed Obligations with Liveline and Deadline. Figure 4 contains the state transition diagram for directed obligations with liveline and deadline. In this case, the obligation will only be pending when l arises, since only then it may be fulfilled in a way that is compliant with the terms of the contract. We have now two kinds of temporal violations: liveline violations of the form $LViol_{b,c}(f,l,d)$ and deadline violations of the form $DViol_{b,c}(f,l,d)$. In both cases, a denounce $(Den_{c,b}(f,l,d))$ may establish the obligation as violated, if issued before l or f, respectively.

The lifecycle of directed obligations with liveline and deadline is formalized as follows:

- $O_{b,c}(f,l,d) \wedge (f\ B\ l) \models LViol_{b,c}(f,l,d)$
- $LViol_{b,c}(f,l,d) \wedge (l\ B\ Den_{c,b}(f,l,d)) \models Fulf_{b,c}(f,l,d)$

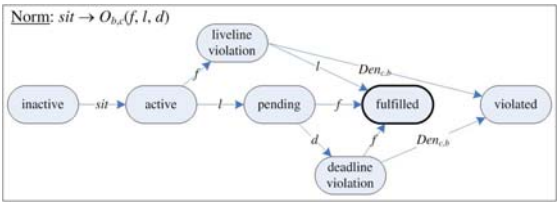

Fig. 4. Lifecycle of a directed obligation with liveline and deadline

- $LViol_{b,c}(f, l, d) \wedge (Den_{c,b}(f, l, d)\ B\ l) \models Viol_{b,c}(f, l, d)$
- $O_{b,c}(f, l, d) \wedge (l\ B\ f) \wedge (f\ B\ d) \models Fulf_{b,c}(f, l, d)$
- $O_{b,c}(f, l, d) \wedge (d\ B\ f) \models DViol_{b,c}(f, l, d)$
- $DViol_{b,c}(f, l, d) \wedge (f\ B\ Den_{c,b}(f, l, d)) \models Fulf_{b,c}(f, l, d)$
- $DViol_{b,c}(f, l, d) \wedge (Den_{c,b}(f, l, d)\ B\ f) \models Viol_{b,c}(f, l, d)$

3 A Normative Environment for Monitoring Norms

In this section we describe our efforts to bring the formalizations presented above towards an implementation of a normative environment with norm monitoring in place. The normative environment is composed of three main ingredients: *normative state*, *institutional rules* and *norms*. The normative state is a set of fully-grounded facts describing relevant events. Institutional rules are rules that manipulate the normative state. Norms are a special kind of rules, in that they are used to prescribe behavior. The next subsections detail each of these elements.

3.1 Normative State

In business contracts it is common to have deadlines that are dependent on the fulfillment date of other obligations. Therefore, instead of having fixed (absolute) dates, these may at times be relative, calculated according to other events. CISG [7] expresses this by saying that dates can be *determinable* from the contract:

> *Article 33: The seller must deliver the goods: (a) if a date is fixed by or determinable from the contract, on that date; (b) if a period of time is fixed by or determinable from the contract, at any time within that period [...]*
>
> *Article 59: The buyer must pay the price on the date fixed by or determinable from the contract [...]*

It is therefore useful to timestamp each event. For this reason, we will use explicit time references in the constituting elements of the normative state. Furthermore, rules and norms will make use of these time references.

We will call each element of the normative state an *institutional reality element* (*IRE*). These elements allow us to record all relevant events regarding the social context that is being monitored. We distinguish several kinds of *IRE*, as shown in Table 2.

Table 2. Institutional reality elements

$Ifact(f)^t$	fact f is institutionally recognized as being the case at time t
$O_{b,c}(f,l,d)^t$	agent b is obliged, since t, towards agent c to bring about f between l and d
$LViol_{b,c}(f,l,d)^t$	there was a liveline violation at time t (fact f has been brought too early)
$DViol_{b,c}(f,l,d)^t$	there was a deadline violation at time t (fact f should have been brought already)
$Fulf_{b,c}(f,l,d)^t$	agent b has fulfilled, at time t, his obligation to bring about f between l and d
$Viol_{b,c}(f,l,d)^t$	agent b has violated, at time t, his obligation to bring about f between l and d
$Den_{c,b}(f,l,d)^t$	agent c has denounced, at time t, the failure of agent b to bring about f between l and d
$Time(t)$	instant t has elapsed

An institutional fact (*Ifact*) is a piece of evidence that certifies the occurrence of an event, where such event (a consequence of an agent action) may be external to the normative environment itself, while being part of the social context it is supposed to regulate. Time elements are used to signal the reach of contractually relevant time instants (namely those concerning livelines and deadlines).

3.2 Rules and Norms

Some of the IRE's that are part of the normative state are interrelated, in the sense that some IRE's are obtained from other IRE's. These interrelations are captured with institutional rules and norms. In the literature [15] [16] a distinction has been made between regulative and constitutive rules. We see norms as regulative rules, that is, rules that change the normative positions of agents, e.g. by prescribing obligations. Our institutional rules allow us to iterate through institutional facts and to further obtain IRE's other than obligations. For instance, institutional rules may be defined to indicate how a denouncement may be obtained from an institutional fact. This kind of inference has therefore a constitutive nature. In this paper we will concentrate on a particular kind of institutional rules: monitoring rules, that are used to implement the semantics of directed obligations with liveline and deadline.

Both institutional rules and norms have a rule-based materialization. Their left-hand-sides (conditions) are composed of (possibly negated) conjunctions of patterns of IRE's, which may contain (universally quantified) variables; restrictions may be imposed on such variables through relational conditions. The right-hand-sides (conclusions) of institutional rules are conjunctions of non-deontic IRE's which are allowed to contain bounded variables; the right-hand-sides of norms are conjunctions of deontic IRE's (obligations) which are allowed to contain bounded variables.[1]

[1] A detailed formalization can be found in [9], with a more complex model that includes context-dependent information elements, which is out of the scope of this paper.

When the conditions of a rule (or norm) match the normative state using a first-order logic substitution Θ, and if all the relational conditions over variables hold, the atomic formulae obtained by applying Θ to the consequent of the rule are added to the normative state as fully-grounded elements.

3.3 Monitoring Rules

The LTL logical relationships in Section 2.2 provided us a formalism to define directed obligations with liveline and deadline. However, in order to monitor contracts at run-time, we need to ground this semantics into a reasoning engine capable of responding to events in a timely fashion. Elements describing obligation states should allow us to reason about those states *as soon as they occur*. A natural choice we have made before [9] is the use of a rule-based inference engine, with which the following (forward-chaining) rules can be defined to implement the semantics of directed obligations with liveline and deadline:

- $O_{b,c}(f,l,d)^i \wedge Ifact(f)^t \wedge t < l \rightarrow LViol_{b,c}(f,l,d)^t$
- $LViol_{b,c}(f,l,d)^i \wedge Time(l) \wedge \neg(Den_{c,b}(f,l,d)^u \wedge u < l) \rightarrow Fulf_{b,c}(f,l,d)^l$
- $LViol_{b,c}(f,l,d)^i \wedge Den_{c,b}(f,l,d)^u \wedge u < l \rightarrow Viol_{b,c}(f,l,d)^u$
- $O_{b,c}(f,l,d)^i \wedge Ifact(f)^t \wedge l < t \wedge t < d \rightarrow Fulf_{b,c}(f,l,d)^t$
- $O_{b,c}(f,l,d)^i \wedge Time(d) \wedge \neg(Ifact(f)^t \wedge t < d) \rightarrow DViol_{b,c}(f,l,d)^d$
- $DViol_{b,c}(f,l,d)^i \wedge Ifact(f)^t \wedge \neg(Den_{c,b}(f,l,d)^u \wedge u < t) \rightarrow Fulf_{b,c}(f,l,d)^t$
- $DViol_{b,c}(f,l,d)^i \wedge Den_{c,b}(f,l,d)^u \wedge \neg(Ifact(f)^t \wedge t < u) \rightarrow Viol_{b,c}(f,l,d)^u$

These rules take into account the time instants when each institutional reality element occurs in order to assert other elements with accurate timestamps. Notice the use of relational conditions in order to assess the temporal ordering of events that may match each rule's conditions.

In the next section we provide an implementation for these rules using a forward-chaining rule-based inference engine.

4 Implementation with Jess

We have chosen Jess[2] [17] to implement our norm monitoring system. Jess is a very efficient rule engine based on the Rete algorithm for pattern matching. We start by defining appropriate templates (through `deftemplate` constructs) for each type of element in the normative state. Jess facts follow a frame-like notation, in which each fact has associated slots to be filled in. Template inheritance is possible via the `extends` keyword. We have:

```
(deftemplate institutional-reality-element
  (slot when) )

(deftemplate ifact extends institutional-reality-element
  (multislot fact) )
```

[2] The code presented in this section includes some simplifications in order to make it more simple to understand.

```
(deftemplate obligation extends institutional-reality-element
  (slot bearer) (slot counterparty) (multislot fact)
  (slot liveline) (slot deadline) )

(deftemplate liveline-violation extends institutional-reality-element
  (slot obl) )

(deftemplate deadline-violation extends institutional-reality-element
  (slot obl) )

(deftemplate fulfillment extends institutional-reality-element
  (slot obl) )

(deftemplate violation extends institutional-reality-element
  (slot obl) )

(deftemplate denounce extends institutional-reality-element
  (slot obl) )

(deftemplate time extends institutional-reality-element)

(deftemplate cancel-contract extends institutional-reality-element)
```

For simplification, we included an obligation reference inside some templates. The `time` template is used to assert the occurrence of time events (associated with livelines and deadlines), which is done by scheduling alerts using a system clock. The `cancel-contract` template enables contract cancellation assertions, and will be used in our contract example below.

4.1 Monitoring Rules

Implementing monitoring rules in Jess is straightforward. A Jess rule is written in the form $LHS \Rightarrow RHS$, where LHS includes fact patterns that will be matched against facts in working memory (our normative state). The RHS indicates actions to execute (such as asserting new facts) when the rule is fired. The following rules (defined with `defrule` constructs) translate directly from the monitoring rules shown in Section 3.3 (identifiers starting with a question mark are variables).

```
(defrule detect-liveline-violation
  ?obl <- (obligation (fact $?f) (liveline ?l))
  ?ifa <- (ifact (fact $?f) {when < ?l})
  =>
  (assert (liveline-violation (obl ?obl) (when ?ifa.when))) )

(defrule detect-early-fulfillment
  ?lviol <- (liveline-violation (obl ?obl))
  ?obl <- (obligation (liveline ?l))
  (time (when ?l))
  (not (denounce (obl ?obl) {when < ?l}))
  =>
  (assert (fulfillment (obl ?obl) (when ?l))) )
```

```
(defrule detect-violation-before-liveline
  ?lviol <- (liveline-violation (obl ?obl))
  ?obl <- (obligation (liveline ?l))
  ?den <- (denounce (obl ?obl) {when < ?l})
  =>
  (assert (violation (obl ?obl) (when ?den.when))) )

(defrule detect-fulfillment
  ?obl <- (obligation (fact $?f) (liveline ?l) (deadline ?d))
  ?ifa <- (ifact (fact $?f) {when >= ?l && when <= ?d})
  =>
  (assert (fulfillment (obl ?obl) (when ?ifa.when))) )

(defrule detect-deadline-violation
  ?obl <- (obligation (fact $?f) (deadline ?d))
  (time (when ?d))
  (not (ifact (fact $?f) {when <= ?d}))
  =>
  (assert (deadline-violation (obl ?obl) (when ?d))) )

(defrule detect-belated-fulfillment
  (deadline-violation (obl ?obl))
  ?obl <- (obligation (fact $?f))
  (ifact (fact $?f) (when ?t))
  (not (denounce (obl ?obl) {when <= ?t}))
  =>
  (assert (fulfillment (obl ?obl) (when ?t))) )

(defrule detect-violation-after-deadline
  (deadline-violation (obl ?obl))
  ?obl <- (obligation (fact $?f))
  (denounce (obl ?obl) (when ?u))
  (not (ifact (fact $?f) {when < ?u}))
  =>
  (assert (violation (obl ?obl) (when ?u))) )
```

In order to determine denouncements from institutional facts, we define the following (constitutive) institutional rule:

```
(defrule denounce-recognition
    (ifact (fact denounce $?f) (when ?w))
    ?obl <- (obligation (fact $?f))
    =>
    (assert (denounce (obl ?obl) (when ?w))) )
```

These rules enable us to monitor the compliance of agents with their obligations. Norms are used to prescribe such obligations, by asserting them into the normative state. The normative environment's monitoring capabilities may be used as a tool for alerting agents when certain contract-related events occur. Further rules may be defined with such a purpose. The *RHS* of Jess rules may include function calls that implement the desired level of responsiveness of the normative environment in which notifications are concerned.

4.2 Example Contract

In this section we show a simple example where the concept of flexible deadlines is exploited in an electronically supervised business relationship. We have a contract between two agents, say B and S, wherein S commits to supply, whenever ordered, good X for 7.5 per unit.

The norms below (implemented as Jess rules) define the contractual relationship and are included in the institutional normative environment for monitoring purposes. Agent S is supposed to deliver the ordered goods between 3 to 5 days after the order (norm n1), and agent B shall pay within 30 days (norm n2). Furthermore, if agent B does not pay in due time, he will incur in a penalty consisting of an obligation to pay an extra 10% on the order total (norm n3). Finally, if agent S violates his obligation to deliver, the contract shall be canceled (norm n4).

```
(defrule n1
  (ifact (fact order item X quantity ?q) (when ?w))
  =>
  (assert
   (obligation (bearer S) (counterparty B) (fact delivery X qt ?q)
               (liveline (+ ?w 3)) (deadline (+ ?w 5)) (when ?w)) ) )

(defrule n2
  (fulfillment (obl ?obl) (when ?w))
  ?obl <- (obligation (fact delivery X qt ?q))
  =>
  (assert
   (obligation (bearer B) (counterparty S) (fact payment (* ?q 7.5))
               (liveline ?w) (deadline (+ ?w 30)) (when ?w)) ) )

(defrule n3
  (deadline-violation (obl ?obl))
  ?obl <- (obligation (fact payment ?p) (deadline ?d))
  =>
  (assert
   (obligation (bearer B) (counterparty S) (fact payment (* ?p 0.10))
               (liveline ?d) (deadline (+ ?d 30)) (when ?d)) ) )

(defrule n4
  (violation (obl ?obl) (when ?w))
  ?obl <- (obligation (fact delivery X qt ?q))
  =>
  (assert (cancel-contract (when ?w)) ) )
```

In this example we can see that interests applied on payments are automatic once deadline violations are detected (norm n3). On the other hand, a contract cancellation (norm n4) requires that agent B denounces the inability of agent S to

Enactment 1: everything goes as agreed

```
f-6  * (ifact (when 1) (fact order item X quantity 5))
f-7    (obligation (when 1) (bearer S) (counterparty B)
                   (fact delivery X qt 5) (liveline 4) (deadline 6))
f-11   (time (when 4))
f-13 * (ifact (when 5) (fact delivery X qt 5))
f-14   (fulfillment (when 5) (obl <Fact-7>))
f-15   (obligation (when 5) (bearer B) (counterparty S)
                   (fact payment 37.5) (liveline 5) (deadline 35))
f-16   (time (when 5))
f-18   (time (when 6))
f-26 * (ifact (when 13) (fact payment 37.5))
f-27   (fulfillment (when 13) (obl <Fact-15>))
f-50   (time (when 35))
```

Fig. 5. A perfect enactment

fulfill the delivery. It is therefore up to agent `B` whether to wait further and accept a delayed delivery or not. If the agreed upon contract conditions are important enough, allowing a counterparty deviation (and hence taking a cooperative attitude regarding the compliance of the contract) may be a good decision.

4.3 Contract Enactments

In this section we show the outcomes of applying monitoring rules and contractual norms in different contract enactment situations, taking into account the contract presented above.

Figures 5, 6 and 7 show the response of monitoring rules to different enactment situations. The listings in these figures show the normative state after contract enactment, including relevant *IRE* that are produced by rules and norms, together with institutional facts originated by agent actions (these are marked with an asterisk). Time events (associated with livelines and deadlines) triggered by a system clock are also shown.

Figure 5 shows the normative state after a perfect contract enactment, where everything goes as agreed. No temporal violations are detected in this case, since agents abide to their obligations. Figure 6 depicts several enactment outcomes in which delivery problems are detected. In enactments 2 and 3 the delivery liveline or deadline is violated (and detected by rules adding `f-10` and `f-15`, respectively), while the counterparty does not denounce this situation. Enactment 4 shows a situation in which the counterparty chooses to denounce (`f-17`) a deadline violation (detected in `f-14`); as indicated in contractual norm `n4`, this results in a contract cancellation (`f-20`). Finally, figure 7 shows an enactment in which the payment deadline was violated, bringing an interest rate to be applied according to contractual norm `n3`. Agent `B` eventually payed both the price (`f-57`) and the interest rate (`f-58`).

Enactment 2: delivery liveline violation without denounce

```
f-6  * (ifact (when 1) (fact order item X quantity 5))
f-7    (obligation (when 1) (bearer S) (counterparty B)
                   (fact delivery X qt 5) (liveline 4) (deadline 6))
f-9  * (ifact (when 2) (fact delivery X qt 5))
f-10   (liveline-violation (when 2) (obl <Fact-7>))
f-13   (time (when 4))
f-14   (fulfillment (when 4) (obl <Fact-7>))
f-15   (obligation (when 4) (bearer B) (counterparty S)
                   (fact payment 37.5) (liveline 4) (deadline 34))
f-18   (time (when 6))
f-26 * (ifact (when 13) (fact payment 37.5))
f-27   (fulfillment (when 13) (obl <Fact-15>))
f-49   (time (when 34))
```

Enactment 3: delivery deadline violation without denounce

```
f-6  * (ifact (when 1) (fact order item X quantity 5))
f-7    (obligation (when 1) (bearer S) (counterparty B)
                   (fact delivery X qt 5) (liveline 4) (deadline 6))
f-11   (time (when 4))
f-14   (time (when 6))
f-15   (deadline-violation (when 6) (obl <Fact-7>))
f-19 * (ifact (when 9) (fact delivery X qt 5))
f-20   (fulfillment (when 9) (obl <Fact-7>))
f-21   (obligation (when 9) (bearer B) (counterparty S)
                   (fact payment 37.5) (liveline 9) (deadline 39))
f-22   (time (when 9))
f-27 * (ifact (when 13) (fact payment 37.5))
f-28   (fulfillment (when 13) (obl <Fact-21>))
f-55   (time (when 39))
```

Enactment 4: delivery deadline violation with denounce

```
f-5  * (ifact (when 1) (fact order item X quantity 5))
f-6    (obligation (when 1) (bearer S) (counterparty B)
                   (fact delivery X qt 5) (liveline 4) (deadline 6))
f-10   (time (when 4))
f-13   (time (when 6))
f-14   (deadline-violation (when 6) (obl <Fact-6>))
f-17 * (ifact (when 8) (fact denounce delivery X qt 5))
f-18   (denounce (when 8) (obl <Fact-6>))
f-19   (violation (when 8) (obl <Fact-6>))
f-20   (cancel-contract (when 8))
```

Fig. 6. Delivery problems

Enactment 5: payment deadline violation

```
f-7  * (ifact (when 1) (fact order item X quantity 5))
f-8    (obligation (when 1) (bearer S) (counterparty B)
                   (fact delivery X qt 5) (liveline 4) (deadline 6))
f-12   (time (when 4))
f-14 * (ifact (when 5) (fact delivery X qt 5))
f-15   (fulfillment (when 5) (obl <Fact-8>))
f-16   (obligation (when 5) (bearer B) (counterparty S)
                   (fact payment 37.5) (liveline 5) (deadline 35))
f-17   (time (when 5))
f-19   (time (when 6))
f-49   (time (when 35))
f-50   (deadline-violation (when 35) (obl <Fact-16>))
f-51   (obligation (when 35) (bearer B) (counterparty S)
                   (fact payment 3.75) (liveline 35) (deadline 65))
f-57 * (ifact (when 40) (fact payment 3.75))
f-58 * (ifact (when 40) (fact payment 37.5))
f-59   (fulfillment (when 40) (obl <Fact-16>))
f-60   (fulfillment (when 40) (obl <Fact-51>))
f-86   (time (when 65))
```

Fig. 7. Payment problems

5 Conclusions

In B2B relationships contracts specify, through obligations, the interdependencies between different partners, and provide legal options to which parties can resort in case of conflict. However, when this joint activity aims at pursuing a common goal, the successful performance of business benefits all involved parties. Therefore, when developing automated monitoring tools, one should take into account that partners may be cooperative enough to allow counterparty deviations. Taking this into account, we have developed a novel model for contractual obligations, where these are seen as either *directed deadline obligations* or *directed obligations with liveline and deadline*. The directed aspect concerns the need to identify the agent who will be authorized to react in case of non-fulfillment. We link authorizations with a flexible model of livelines and deadlines. Obligation violations are dependent on the counterparty motivation to claim them. Using flexible deadlines ensures a degree of freedom for agents to make decisions in the execution phase of contracts, which is important for dealing with business uncertainty. Our approach is based on real-world evidence from business contracts (namely the United Nations Convention on Contracts for the International Sale of Goods [7]), which denotes a flexible and even cooperative facet of trade contracts.

Most implementations of norms in multi-agent systems ignore the need for having directed obligations from bearers to counterparties. The most likely reason for this is that in those approaches obligations are seen as (implicitly) directed from an agent to the normative system itself. It is up to the system (e.g.

an electronic organization [18] or an electronic institution [19]) to detect violations and to enforce the norms which are designed into the environment (in some cases they are even regimented in such a way that violation is not possible). On the contrary, our flexible approach towards an Electronic Institution [5, 20] allows agents to define the norms that will regulate their mutual commitments.

An issue that we have not included in our model is the need for agents to communicate their intentions regarding an obligation with a violated deadline. In fact, CISG's Article 48 seems to go in this direction, in order to protect the bearer's efforts toward a late fulfillment of the obligation. This concern has been taken into account in [21], where a contract fulfillment protocol demanding agents to communicate their intentions drives an obligation lifecycle model. The states of this lifecycle are obtained according to the performance of a contractual relationship.

Our implementation using a forward-chaining rule-based approach is applicable to run-time monitoring of contracts. A requirement of this kind of usage however is that events are reported in a timely fashion to the normative environment. We assume that agents are interested in publicizing their abidance to commitments. The monitoring capabilities of our implementation may, however, be used as a tool to alert agents when certain contract-related events are eminent, such as a forthcoming deadline. Jess [17] allows for an easy integration of our monitoring rules implementation with other rules including function calls that address the level of responsiveness that is intended.

We have shown how the normative environment may effectively monitor contract enactment at run-time. Monitoring rules may also be used *a posteriori*, in order to check off-line if contractual norms were indeed followed by every partner. In this case, after collecting all events concerning a contract, the inference engine may run in order to check if the contract was enacted in a conforming way.

Acknowledgments. The first author is supported by FCT (Fundação para a Ciência e a Tecnologia) under grant SFRH/BD/29773/2006.

References

1. Masten, S.E.: Contractual choice. In: Bouckaert, B., De Geest, G. (eds.) Encyclopedia of Law and Economics. Volume Volume III: The Regulation of Contracts, pp. 25–45. Edward Elgar, Cheltenham (2000)
2. Arcos, J.L., Esteva, M., Noriega, P., Rodríguez-Aguilar, J.A., Sierra, C.: Environment engineering for multiagent systems. Engineering Applications of Artificial Intelligence 18(2), 191–204 (2005)
3. Artikis, A., Sergot, M., Pitt, J.: Specifying norm-governed computational societies. ACM Transactions on Computational Logic 10(1), 1:1–1:42 (2009)
4. Fornara, N., Colombetti, M.: Specifying and enforcing norms in artificial institutions. In: Padgham, P., Müller, P. (eds.) 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, pp. 1481–1484 (2008)
5. Lopes Cardoso, H., Oliveira, E.: Electronic institutions for b2b: Dynamic normative environments. Artificial Intelligence and Law 16(1), 107–128 (2008)

6. Modgil, S., Faci, N., Meneguzzi, F., Oren, N., Miles, S., Luck, M.: A framework for monitoring agent-based normative systems. In: Decker, S., Sierra, C. (eds.) Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems. IFAAMAS, Budapest, Hungary, pp. 153–160 (2009)
7. UNCITRAL: United nations convention on contracts for the international sale of goods, cisg (1980)
8. Broersen, J., Dignum, F., Dignum, V., Meyer, J.J.C.: Designing a deontic logic of deadlines. In: Lomuscio, A., Nute, D. (eds.) DEON 2004. LNCS (LNAI), vol. 3065, pp. 43–56. Springer, Heidelberg (2004)
9. Lopes Cardoso, H., Oliveira, E.: A context-based institutional normative environment. In: Hubner, J., Matson, E., Boissier, O., Dignum, V. (eds.) Coordination, Organizations, Institutions, and Norms in Agent Systems IV. LNCS (LNAI), vol. 5428, pp. 140–155. Springer, Heidelberg (2009)
10. Ryu, Y.U.: Relativized deontic modalities for contractual obligations in formal business communication. In: 30th Hawaii International Conference on System Sciences (HICSS), Hawaii, USA, vol. 4, pp. 485–493 (1997)
11. Tan, Y.H., Thoen, W.: Modeling directed obligations and permissions in trade contracts. In: Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences, vol. 5. IEEE Computer Society, Los Alamitos (1998)
12. Lopes Cardoso, H., Oliveira, E.: Directed deadline obligations in agent-based business contracts. In: Artikis, A., Vasconcelos, W. (eds.) AAMAS 2009 Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN), Budapest, Hungary, pp. 77–92 (2009)
13. Emerson, E.A.: Temporal and modal logic. In: Leeuwen, J.v. (ed.) Handbook of Theoretical Computer Science. Formal Models and Sematics, vol. B, pp. 995–1072. North-Holland Pub. Co./MIT Press (1990)
14. Herrestad, H., Krogh, C.: Obligations directed from bearers to counterparties. In: Proceedings of the 5th international conference on Artificial intelligence and law, College Park, Maryland, United States, pp. 210–218. ACM, New York (1995)
15. Searle, J.R.: The Construction of Social Reality. Free Press, New York (1995)
16. Boella, G., van der Torre, L.: Regulative and constitutive norms in normative multiagent systems. In: Dubois, D., Welty, C., Williams, M.A. (eds.) Ninth International Conference on Principles of Knowledge Representation and Reasoning, pp. 255–266. AAAI Press, Whistler (2004)
17. Friedman-Hill, E.: Jess in Action. Manning Publications Co. (2003)
18. Vázquez-Salceda, J., Dignum, F.: Modelling electronic organizations. In: Mařík, V., Müller, J.P., Pěchouček, M. (eds.) CEEMAS 2003. LNCS (LNAI), vol. 2691, pp. 584–593. Springer, Heidelberg (2003)
19. Esteva, M., Rodríguez-Aguilar, J.A., Sierra, C., Garcia, P., Arcos, J.L.: On the formal specifications of electronic institutions. In: Dignum, F., Sierra, C. (eds.) Agent-mediated Electronic commerce: The European AgentLink Perspective. LNCS (LNAI), vol. 1991, pp. 126–147. Springer, Heidelberg (2001)
20. Lopes Cardoso, H., Oliveira, E.: Norm defeasibility in an institutional normative framework. In: Proceedings of The 18th European Conference on Artificial Intelligence (ECAI 2008), Patras, Greece, pp. 468–472. IOS Press, Amsterdam (2008)
21. Sallé, M.: Electronic contract framework for contractual agents. In: Cohen, R., Spencer, B. (eds.) Canadian AI 2002. LNCS (LNAI), vol. 2338, pp. 349–353. Springer, Heidelberg (2002)

Unifying the Intentional and Institutional Semantics of Speech Acts

Carole Adam[1], Andreas Herzig[2], Dominique Longin[2], and Vincent Louis

[1] RMIT University, Melbourne, VIC, Australia
carole.adam.rmit@gmail.com
[2] Université de Toulouse, CNRS, IRIT (UMR 5505), France
firstname.surname@irit.fr

Abstract. Research about the semantics of agent communication languages traditionally sees the opposition between the mentalist and social approaches. In this paper we adopt a mixed approach since we propose a logical framework allowing us to express both the intentional and institutional dimensions of a communicative action. We use this framework to give a semantics for some speech acts representing each of Searle's categories except expressives. This semantics relaxes the criticized constraints imposed in FIPA-ACL and also extends this standard with new speech acts and new institutional features to characterise them. It has been implemented in an extension of the Semantic Add-on for the JADE agent development platform, and used in an industrial application in the context of automated B2B exchanges.

1 Introduction

Designing efficient Agent Communication Languages (ACL) is an essential issue in Multi-Agent Systems in order to standardise exchanges between the agents. Research about the semantics of ACL sees the subscribers of the social approach [17,8,27] criticize mentalist approaches [24,15] for only grounding on the agents' private mental attitudes. But one can similarly reproach to social approaches to provide a semantics only based on the agents' public commitments, independently of their mental attitudes. Now these "social attitudes" are mainly descriptive, while mental attitudes allow one to predict the agents' behaviour. Moreover mental attitudes allow agents to reason about social notions. It is thus essential to consider both mental and social attitudes. Some researchers thus propose a mixed approach based both on public and private aspects [18]. But they do not formalise institutional speech acts like declarations. Now such speech acts are essential in new application fields involving communication about norms, roles or powers of agents, for instance in electronic commerce or automated business to business exchanges.

In this paper we thus want to propose an alternative to the well-known standard FIPA-ACL [16] through the following changes: relaxed feasibility preconditions to allow a more flexible utilisation of the speech acts in various contexts;

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 68–84, 2010.

new institutional speech acts like declarations and promises; an institutional interpretation of speech acts coupled with their classical intentional interpretation. Therefore we adapt an existing logical framework for the formalisation of institutional notions like roles, powers and norms [10]. We then formalise in this logical framework the institutional interpretation of some specific communicative actions, each one representing one of Searle's categories of speech acts (except expressive ones) [25]. Our notion of institution is very large: it is a set of rules and facts adopted by a group of agents, like the rules of a game, or the laws of a country; it covers formal, legal institutions as well as informal ones (*e.g.* social rules in a group...).

The paper is structured as follow. Section 2 discusses some other social semantics of speech acts. Section 3 briefly describes the syntax, semantics and axiomatics of our logical framework. The core of the paper (Section 4) is dedicated to the unified semantics of speech acts. We are then able to compare our semantics of ACL with some related ones in more details (Section 5). Finally we conclude about the future prospects opened by this work (Section 6).

2 State of the Art

The mentalist approach consists in grounding the semantics of speech acts on the agents' internal mental attitudes. These are represented by belief, desire and intention modalities provided by BDI logics, that are classically used to formalise the reasoning of autonomous agents [23,28]. This resulted in the design of several standards of agent communication languages like KQML [15] or FIPA [16], this latter one grounding on Sadek's rational interaction theory [24].

These approaches were criticised a lot for being only based on private concepts (mental attitudes) instead of public verifiable notions (like commitments). Therefore some work exist aiming at enriching BDI logics with deontic operators like obligation [13,3] or with institutional operators like *count as* or institutional power [21], in order to formalise the institutional interpretation of speech acts exchanged by the agents. In previous work we used such an extended BDI framework to express the semantics of speech acts with institutional effects [11] but we were limited to declarative speech acts, and the intentional and institutional dimensions were quite blended.

Various other work aim at providing an institutional semantics for speech acts. For example Dignum and Weigand [14] propose a logical framework combining illocutionary and deontic logic to study and model the norms resulting from communication between agents; however, they only consider directive speech acts. Boella *et al.* [1] propose a role-based semantics allowing them to combine social commitments and mental attitudes to express the semantics of speech acts in the context of persuasion dialogues. Actually they rewrite the FIPA feasibility precondition and rational effects of speech acts but replace the private mental attitudes involved by public mental attitudes attributed to the agents' roles instead of the individual agents. This solves the flaw of mentalist approaches, criticised for grounding on unverifiable mental attitudes, but finally there is

no distinct institutional interpretation of speech acts, that could differ from one institution to another. In the following subsections we give some details about two approaches: Fornara and Colombetti's approach based on the notion of commitments, and Lorini *et al.* 's approach based on the notion of acceptance.

2.1 Fornara and Colombetti: Semantics in Terms of Social Commitments

As opposed to the mentalist approach, the social one [26,27,8] assumes that private mental attitudes are not verifiable and thus grounds on the concept of public (thus verifiable) commitments [7] to express the semantics of speech acts. All the commitments taken by the agents are stored for possible future reference. The semantics of speech acts is expressed only in terms of such commitments.

For example Fornara and Colombetti [17] ground on Castelfranchi's notion of commitment [7] to define a library of communicative acts. From the classification of speech acts into four categories (assertives, directives, commissives and declaratives) inspired from Searle's work [25], they redefine for each category the semantics of its speech acts in terms of social commitments. Thanks to this library, they provide a communication tool based on social commitments, alternative to the FIPA-ACL standard. This tool allows rational agents to reason about the underlying rules of communication and to respect them in order for the system to behave well.

However these authors are limited to the institutional dimension of speech acts and neglect their relations with the agents' mental attitudes. Yet agents must be able to reason autonomously about the institution before making their decision to perform a given speech act. Moreover no specific institution is explicit in their commitments, making it impossible to have different commitments in different institutions; therefore it is also impossible for speech acts to have different effects depending on the institution within which they are interpreted. For example the action of nodding one's head is interpreted in the context of French gestural language as meaning "yes", while in the context of Bulgarian gestural language it is interpreted as meaning "no". In this example the considered institutions are the respective sets of communicative rules in these two cultures.

2.2 Lorini *et al.* : Semantics in Terms of Group Acceptance

Lorini *et al.* [22] define a new semantics for speech acts using Gaudou *et al.* 's Acceptance Logic [19]. $\mathscr{AL}$ is a modal logic extended with the notion of acceptance, representing what a group of agents willingly accept to consider as true (even if some (or all) members of the group believe the opposite) in a given institutional context (and that they can refuse in another context). Acceptances in an institutional context influence the agents' behaviour and utterances in this context. They are represented with the operator $[C : x]\varphi$ reading "agents in group C accept that φ while functioning as members of this group in the institutional context x".

Institutional notions are not primitive but defined from this notion of acceptance. Thereby a fact is an *institutional fact* (that it, a fact that is only valid

in an institutional context, but not objectively valid) if and only if, for every non-empty set of agents, the agents in this set accept this fact as true while functioning as group members in this institutional context. In the context of ACLs, this may be a particular rule of the specific *Ordinary Communication* institution that these authors consider.

The authors then consider the speech act Promise in the institutional context of Ordinary Communication (OC). According to them, if i informs j that he is going to perform action α for him, and j intends i to perform this action for him, this counts as a promise at the next instant. The consequence of this promise is that i is obliged to perform action α for j. Moreover the acceptance by these two agents i and j while functioning as a group in institution OC that i has promised to perform action α for j and that j intends him to do so implies a social commitment of i towards j to perform α for him. This framework is interesting but Lorini *et al.* have only formalised the promise yet. Moreover they do not seem to make a clear distinction between the intentional and institutional preconditions to perform a speech act.

3 Our Logical Framework

We adapt here an existing logical framework for norms, institutional powers and roles defined in [10]. It is a multi-modal logic with modal operators of belief, intention, obligation, institutional facts and consequences, and action.

3.1 Syntax

Let $AGT = \{i, j, ...\}$ be a finite set of agents. Let $ACT = \{\alpha, \beta...\}$ be the set of actions. We suppose that some actions in ACT are of the form $i{:}\alpha$, where i is the author of action α (the agent who performs it). Let $ATM = \{p, q, ...\}$ be the set of atomic formulas. Let $INST = \{s, t, ...\}$ be the set of institutions. Complex formulas are denoted by φ, ψ... The language of our logic is defined by the following BNF grammar:

$$\varphi ::= p|\neg\varphi|\varphi \vee \varphi|B_i\varphi|Ch_i\varphi|I_i\varphi|D_s\varphi|\varphi \Rightarrow_s \varphi|O\varphi|before_\alpha\varphi|after_\alpha\varphi$$

where p ranges over ATM, α over ACT, i over AGT, and s over $INST$. The classical boolean connectives $\wedge$, $\rightarrow$, $\leftrightarrow$, $\top$ (tautology) and $\bot$ (contradiction) are defined from $\vee$ and $\neg$ in the usual manner. The operators $done_\alpha\varphi$, $happens_\alpha\varphi$, $P\varphi$, $F\varphi$ and $power(i, s, \varphi, \alpha, \psi)$ will be defined as abbreviations.

3.2 Semantics and Axiomatics

We only give here the informal meanings of our operators. It is sufficient to know that they have a Kripke semantics in terms of possible worlds. We also give some useful axioms. This framework is adapted from Demolombe and Louis' logic of norms, roles and institutional powers [10]. But please notice that actually, the details of the semantics of operators is not important, and any other institutional logic would work.

Belief, Intention and Action. $B_i p$ means that agent i believes that p. $Ch_i p$ means that agent i prefers p to be true. These two normal operators have a standard KD45 axiomatics. $I_i p$ means that agent i intends that p. I_i are operators defined in a KD normal modal logic. Their axiomatics is that defined for FIPA by Sadek [24]. In particular intention is linked with belief by the following mix axioms:

- introspection: $I_i p \leftrightarrow B_i I_i p$
- automatic dropping of achieved intentions: $I_i p \rightarrow \neg B_i p$

$before_\alpha$ and $after_\alpha$ are normal modal operators defined in standard tense logic in linear time version [6]. $done_\alpha \varphi = \neg before_\alpha \neg \varphi$ means that action α has just been performed, and φ was true before. $happens_\alpha \varphi = \neg after_\alpha \neg \varphi$ means that action α is about to be performed and φ will be true just after.

Institutional Modalities. Finally this framework also provides some specific operators to formalise institutional concepts. These operators have a parameter s specifying the institution within which they are valid. Here we consider an institution as a set of institutional facts and rules that a group of agents (the "members" of this institution) adopt. This is a general view that can account for various institutional contexts, be they formal institutions or informal ones: the law of a country, a contract between two parties in a business relationship, a social structure, the rules of a game...

An **institutional fact** is a fact that is recognised to be valid in the context of a given institution, but that can make no sense in itself; *i.e.* it is not a physically observable fact (what Searle calls a "brute fact") but something written in the registry of this institution. For example the fact that two people are married, or that one is authorised to drive a truck, is only valid *w.r.t.* the law of a country; all deontic facts should also be encapsulated in an institutional fact to make the institution in which they hold explicit. We represent these institutional facts with the operator $D_s \varphi$ meaning that for institution s, it is officially established that φ holds. In particular if φ is an agent's mental attitude, then $D_s \varphi$ can be understood as this agent's commitment (either a propositional commitment if φ is a belief, or a commitment in action if φ is an intention). For instance, $D_{FrenchLaw}\, votingAgeis18$ means that following the French law, voting age is reached at 18 years; $D_{EU}\, euroOfficialMoney$ means that in the European Union, the official money is Euro.

Institutional facts can be deduced from other facts thanks to the rules of the institution. For example the presentation of an invoice by a provider to his client *counts as* an obligation for the client to pay it. The existence of the invoice is physically observable, while the obligation is only valid in an institutional context. We represent these **normative consequences** with the primitive operator $p \Rightarrow_s q$, meaning that according to the norms holding in institution s, p entails q. This operator is known in the literature as *count as*, and has been first

formalised by Sergot and Jones [21]. The following mix axioms explicit the link between institutional facts and normative consequences:

$$(\varphi \Rightarrow_s \psi) \rightarrow D_s(\varphi \rightarrow \psi) \quad \text{(SD)}$$
$$(\varphi \Rightarrow_s \psi) \rightarrow (\varphi \rightarrow D_s\varphi) \quad \text{(SC)}$$

From these axioms and the properties of D_s (see [10, p.8] for details) we can deduce:

$$(\varphi \Rightarrow_s \psi) \rightarrow (\varphi \rightarrow D_s\psi) \quad \text{(SP)}$$

A particular case of normative consequence concerns the consequences of the performance of an official procedure. Actually some agents can have the power when performing a given procedure under some conditions to create new institutional facts. We represent these **institutional powers** as an abbreviation $power(i, s, cond, \alpha, \varphi) = ((done_{i:\alpha}\top \wedge cond) \Rightarrow_s \varphi)$. Intuitively this means that i has the power in institution s, by performing action α and if condition *cond* holds, to see to it that φ becomes officially true in institution s. For example a mayor has the power in the law of the French Republic, by performing a declaration, and on condition that the two people agree, to marry them. Obviously these powers result from the agent's role in the institution, but this is not the focus of this paper so we will not remind how roles are formalised in the original framework (the interested reader can refer to [10] for details on this point).

Deontic Modalities. We have a modality for impersonal *obligation to be*: $O\varphi$ reads "it is obligatory that φ", and its axiomatic is that of the Standard Deontic Logic [20], that is KD. *Obligations to do* can be expressed as obligations to be in a state where the obliged action has been performed. Obligations are impersonal since no agent is explicitly responsible for their fulfilment, but such an agent can implicitly appear in their content. For instance $O done_{i:\alpha}\top$ means that it is obligatory (for no one in particular) to be in a state where i has just performed action α; this can be understood as "i has the obligation to perform action α".

Permissions and interdiction are defined from obligations in a standard way: $P\varphi = \neg O\neg p$ means that it is permitted that φ, and $F\varphi = O\neg\varphi$ means that it is forbidden that φ.

Please notice that no institution is explicit as a parameter of this obligation modality. But such obligations will be encapsulated in institutional facts to express the institution in which they are valid. For example $D_s O\varphi$ means that "in institution s, it is obligatory that φ".

4 Semantics of Speech Acts

4.1 Preliminary Remarks

Intentional and Institutional Dimensions. The FIPA-ACL standard [16] defines features allowing one to give an **intentional dimension** to the observation and interpretation of a communicative action: the feasibility precondition

(the appropriate mental attitudes to perform the speech act) and the rational effect (this is a formula φ representing the content of the speaker i's intention that he intends the receiver j to know; so the performance of the speech act allows any observer k to deduce this corresponding intentional effect: $B_k I_i B_j I_i \varphi$). Please notice that, following [24], the performance of the speech act does **not** automatically allow one to deduce its rational effect, but only its intentional effect, meaning that any agent k believes that the speaker i intends the hearer j to recognize its (i's) intention to achieve the rational effect φ. However, nothing ensures that i indeed achieves φ, his speech act may fail, for example the hearer may not obey an order, or may not believe an assertion. Thus the rational effect can only be deduced under some constraining hypotheses such as the sincerity and competence hypotheses used in FIPA.

In a similar way, we want to provide here the **institutional dimension** of the observation and interpretation of a communicative action relative to one or several institutions. This institutional interpretation is composed of the following features:

- a *permission condition* that is necessary and sufficient for the speaker to be allowed to perform this speech act;
- a *power condition* that also needs to be true for the speech act to have an institutional effect;
- an *explicit institutional effect* that is obtained when the speech act is performed while permission and power conditions were true.

We will thus be able to combine the intentional and institutional dimensions of communicative actions (formalised as speech acts [25]), both essential to fully characterise their interpretation. Lorini *et al.* have also investigated such a unified approach but they have only formalised the interpretation of a promise in the context of ordinary communication; we aim at being much more generic. In particular we formalise one speech act from each of Searle's categories of illocutionary forces, except the expressive one.

Actually we have relaxed some of the (widely criticised) strong constraints imposed by FIPA-ACL semantics on the appropriate context of performance of speech acts. Instead of imposing these conditions as strong constraints, we have moved them into the permission preconditions of the speech act. The agents are thus physically able to disobey these constraints, but it is forbidden by the interaction norms, and they may incur sanctions for such violations. For example, relaxing the sincerity hypothesis physically allows the agents to lie, but this will be interpreted by other agents as a violation of communicative norms.

Notations. In the sequel we use the following abbreviations:

- FP = feasibility preconditions
- RE = rational effect
- PermC = (institutional) permission condition
- PowC = power condition
- EE = institutional explicit effect

Speech acts are actions of the form $Force(sp, ad, inst, content)$ where $sp \in AGT$ is the speaker, $ad \in AGT$ is the addressee, $inst \in INST$ is the institutional context, *content* is the propositional content and can be any formula of our language, and $Force \in \{inform, promise, command, declare\}$ is the illocutionary force.

Action Laws. We now explain how the intentional and institutional dimensions of actions interact by providing the action laws governing the performance of speech acts.

We notice that FP is a factual executability precondition, while $PermC$ is an ideal one. But even ideal worlds are submitted to physical world laws, *i.e.* $PermC$ is not sufficient for the action to be executable, FP also has to be true. For example a mayor has the permission to marry people by making a declaration, but the declaration must be executable; thus if he is voiceless one day, he will be unable to marry anyone.

We thus have the following executability laws. The factual executability law (FEL_α) means that an action happens iff its feasibility precondition is true and the agent chooses to perform it. The ideal executability law (IEL_α) means that ideally, an action should happen only if it is permitted.

$$happens_\alpha \top \leftrightarrow (FP(\alpha) \wedge Ch_i happens_{i:\alpha} \top) \qquad (FEL_\alpha)$$

$$O(happens_\alpha \top \rightarrow PermC(\alpha)) \qquad (IEL_\alpha)$$

We also have the following effect laws. The rational effect law (REL_α) means that if the power precondition of an action is false, then only its rational effect can be deduced after its performance. The power effect law (PEL_α) means that if the power condition of an action is true, then both its rational and institutional effects can be deduced after its performance.

$$\neg PowC(\alpha) \rightarrow after_\alpha RE(\alpha) \qquad (REL_\alpha)$$

$$PowC(\alpha) \rightarrow after_\alpha (RE(\alpha) \wedge EE(\alpha)) \qquad (PEL_\alpha)$$

From these laws we can deduce the following theorems clarifying the factual executability and effects of α depending on the different combinations of its feasibility and power preconditions. If $FP(\alpha)$ is false then α is not executable.

$$\neg FP(\alpha) \rightarrow after_\alpha \bot$$

If i chooses to perform α when $FP(\alpha)$ is true but $PowC(\alpha)$ is false, then α is about to happen after which its rational effect will be true.

$$(Ch_i happens_{i:\alpha} \top \wedge FP(\alpha) \wedge \neg PowC(\alpha)) \rightarrow (happens_\alpha \top \wedge after_\alpha RE(\alpha))$$

Finally if i chooses to perform α when both $FP(\alpha)$ and $PowC(\alpha)$ are true, then α is about to happen after which both its rational and institutional effects will be true.

$$(Ch_i happens_{i:\alpha} \top \wedge FP(\alpha) \wedge PowC(\alpha)) \rightarrow (happens_\alpha \top \wedge after_\alpha (RE(\alpha) \wedge EE(\alpha)))$$

Please notice that $PermC(\alpha)$ does not appear in these last two theorems, since it does not influence the feasibility of α. Indeed an agent can choose to perform a forbidden action. If we had specified an explicit sanction $S(\alpha)$ for forbidden performance in the institutional interpretation of α, then we could write the following theorem:

$$Ch_i happens_{i:\alpha} \top \wedge FP(\alpha) \wedge \neg PermC(\alpha) \rightarrow (happens_{\alpha} \top \wedge after_{\alpha} S(\alpha))$$

However we did not specified such a sanction because it depends on many contextual parameters.

4.2 Assertives: Inform

The assertive speech act *Inform* commits the speaker to the truth of a proposition. The notation $inform(i, j, s, \varphi)$ reads "agent i informs j in institution s that φ is true".

Intentional Interpretation. As we said before we have relaxed FIPA constraints on the executability of speech acts. We thus impose no feasibility precondition here.

$$FP(inform(i, j, s, \varphi)) = \top$$

The rational effect (the content of the speaker i's intention that he intends the receiver j to know) is that j believes the promised proposition φ to be true:

$$RE(inform(i, j, s, \varphi)) = B_j \varphi$$

Institutional Interpretation. The permission precondition to inform j that φ in institution s includes the constraints removed from the factual feasibility preconditions: the speaker should not believe that the hearer already knows if φ, and he should not be already committed on $\neg\varphi$ in the same institution.

$$PermC(inform(i, j, s, \varphi)) = \neg D_s B_i Bif_j \varphi \wedge \neg D_s B_i \neg\varphi$$

Now the institutional effect of *Inform* is to retract possible opposite commitments contracted before and to assert a new commitment on φ. Indeed, even if agent i was previously committed on $\neg\varphi$ (and therefore was not permitted to inform anyone that φ), he may violate that obligation. But these two commitments are inconsistent so the previous one must be retracted while asserting the new contradictory one. Though one can still detect that the opposite commitment was true when i performed the action and that he has thus violated the rules of the institution. Actually due to the seriality of D_s we have that $D_s B_i \varphi \rightarrow \neg D_s B_i \neg\varphi$. So the explicit institutional effect of inform is a new institutional fact that can be interpreted as i's commitment to the truth of φ:

$$EE(inform(i, j, s, \varphi)) = D_s B_i \varphi$$

This effect is always obtained and does not depend on particular powers of i, so the power condition is trivial.

$$PowC(inform(i, j, s, \varphi)) = \top$$

Example. For example in the context of B2B exchanges, if a provider sends his catalogue to a client, this counts as information about the prices given in this catalogue. As an effect of this action, the provider is thus committed to these prices during the validity of his catalogue. In the specific institution s constituted by the contract between the provider and the client, we assume that we have a specific rule forbidding to contradict one's commitments, which takes the form $D_s O(D_s B_i p \rightarrow after_{\alpha} D_s B_i p)$, for every speech act α, where p is the proposition denoting that the price is 100. This means that according to the institutional contract s between i and j, it is obligatory that if an agent i is committed to believe that the price of an item is 100, then after any speech act he is still committed to this (in other words it is forbidden to retract this commitment by any speech act). From this we can deduce that the provider is obliged to respect the given prices, *i.e.* $D_s O(D_s B_i p \rightarrow after_{Inform(i,j,s,\neg p)} \bot)$ (it is obligatory that if i is committed to p, then the action of informing agent j that $\neg p$ is not feasible).

Please note that the provider i can set up different contracts with different clients, in particular with different prices. This is made possible by making the institution explicit in the semantics of speech acts, and thus allowing us to specify different semantics in different institutions.

4.3 Commissives: Promise to

This commissive speech act commits the speaker on a course of action. The notation *promise-to*(i, j, s, α) reads "i promises to j in institution s to perform action α".

Intentional Interpretation. We begin with specifying the intentional dimension of this speech act, that is not given in FIPA-ACL. A promise-to is feasible if the speaker believes that the hearer intends the concerned action to be performed[1]. For example it makes no sense that a child promises to his father to play (this is rather an assertive), while it makes sense to promise him to make his schoolwork. So:

$$FP(\textit{promise-to}(i, j, s, \alpha)) = B_i I_j \, done_{\alpha} \top$$

The rational effect pursued by the speaker is that the hearer be aware of his intention to perform the promised action:

$$RE(\textit{promise-to}(i, j, s, \alpha)) = B_j I_i \, done_{i:\alpha} \top$$

Institutional Interpretation. In an institutional context s, this promise to perform an action α is permitted on condition that the action $i{:}\alpha$ is not explicitly forbidden itself, and that the speaker is not committed to an opposite course of action. So the permission precondition is the following:

$$PermC(\textit{promise-to}(i, j, s, \alpha)) = \neg D_s O \neg happens_{i:\alpha} \top \wedge \neg D_s I_i \neg done_{i:\alpha} \top$$

[1] Please notice that threats such as "I promise that I will kill you" cannot be considered as promises in the sense of Searle.

The institutional effect consists in ratifying in institution s the speaker's intention to perform action α; so after *promise-to*(i, j, s, α) the speaker has stored in the registry of s its intention to perform α, which is similar to him being committed in s to this course of action.

$$EE(\textit{promise-to}(i, j, s, \alpha)) = D_s I_i \, done_{i:\alpha} \top$$

This is thus similar to the *inform*(i, j, s, φ) speech act except that a promise stores a commitment in action while an inform stores a propositional commitment.

There is no power precondition, so the institutional effect of a (permitted) promise is always reached.

$$PowC(\textit{promise-to}(i, j, s, \alpha)) = \top$$

Example. A client c promises to pay his provider p once the ordered goods have been delivered. The action to pay is denoted by α_{pay} This promise is valid in the context of a B2B exchange contract, that is a particular institution denoted $b2b$ here. So this promise is formalised as: $promise(c, p, b2b, \alpha_{pay})$. This promise is permitted since obviously the promised action to pay is not forbidden ($\neg D_{b2b} O \neg happens_{c:\alpha_{pay}} \top$) and the client is not committed not to pay ($\neg D_{b2b} I_c \neg done_{c:\alpha_{pay}} \top$). So when the client receives the delivery, his promise allows to deduce his commitment (or ratified intention) to pay: $D_{b2b} I_c \, done_{c:\alpha_{pay}} \top$, that is the institutional effect of this speech act.

4.4 Directives: Command

This directive speech act is commonly used by the speaker to make the hearer perform some action. The notation *command*$(i, j, s, \alpha, cond)$ reads "i orders to j in institution s, in virtue of condition *cond*, to perform action α".

Intentional Interpretation. According to the FIPA-ACL semantics, a request is feasible only if the speaker does not believe the hearer to already intend to perform the commanded action, and does believe that the part of the feasibility preconditions of the commanded action that concerns him (*i.e.* that are his mental attitudes) are valid. Here we consider that when α is an action of agent j then $FP(\alpha)$ is of the form $FP_i(\alpha) \wedge FP_{\neq i}(\alpha)$ where the former is "i's part of $FP(\alpha)$" (similar to FIPA-ACL notation $FP(\alpha)[i\backslash j]$, that is the part of $FP(\alpha)$ that are mental attitudes of agent i). But we do not impose this latter constraint on the feasibility of α as a feasibility precondition of the command. So:

$$FP(\textit{command}(i, j, s, \alpha, cond)) = \neg B_i I_j \, done_{j:\alpha} \top$$

The rational effect of a command (*i.e.* the effect that i intends j to believe that i intends to achieve) is that j has performed the commanded action:

$$RE(\textit{command}(i, j, s, \alpha, cond)) = done_{j:\alpha} \top$$

Institutional Interpretation. The permission precondition to command someone to perform an action is to be empowered to do so, *i.e.* to dispose of the institutional power to create the obligation to perform the commanded action by commanding it under some condition *cond* given as an explicit attribute of the command.[2]

An additional permission precondition is the constraint coming from FIPA feasibility precondition that we relaxed, that is that the part of the feasibility preconditions of α that depends on i hold (one should not command someone to perform an action whose preconditions are made false by his own mental attitudes). Finally, one is not permitted to command someone to perform a forbidden action.

$$PermC(command(i,j,s,\alpha,cond)) = \neg D_s\, O \neg happens_{j:\alpha} \top \wedge$$
$$power(i,s,cond,command(i,j,s,\alpha,cond), O\, done_{j:\alpha} \top) \wedge FP_i(\alpha)$$

The explicit institutional effect of this power is to create two new institutional facts, corresponding to the obligation for j to perform α, and the recording of j's knowledge of his obligation. Actually this obligation could exist before, and in this case the command corresponds to a notification; but it can also be created from scratch by the command (see the examples in the next paragraph for clarification).

$$EE(command(i,j,s,\alpha,cond)) = D_s\, O\, done_{j:\alpha} \top \wedge D_s B_j\, O\, done_{j:\alpha} \top$$

This explicit institutional effect is only deduced if the power applies in the current context, *i.e.* if its condition is true. So:

$$PowC(command(i,j,s,\alpha,cond)) = cond$$

Example. For example a parent can command his children to clean his room. In this case, the action becomes obligatory through the command, because of the parent's authority over his son. In other words, his parent role gives him the institutional power to command his child to perform actions, under some conditions on the nature of the actions. Similarly a professor commanding his students to make some schoolwork creates the obligation for them to do so, on the strength of his role of professor. Indeed the role of professor gives an institutional power to command students to perform schoolwork, under the condition that it is related to the course.

But an order does not necessarily *create* an obligation, and may just put in focus an existing one. For example a bailiff can be sent to officially command an uncooperative client to pay an invoice. In this case the obligation already exists (and is attested by the invoice) so the bailiff only reminds the client of it[3]. He is

[2] Institutional powers obviously depend on roles. This notion has been explored in previous work [10] but we will not enter in the details here since they are not in the scope of this paper.

[3] Actually this seems to be a notification rather than a command, but the aim is to make the client behave, while the aim of a notification is only to make the receiver officially aware of what is notified. In further work we expect to study into more details the links between declarations, commands and notifications.

permitted to perform such a command in virtue of his role of bailiff (which gives him the power to force clients to pay) and because he is sent by the provider (which constitutes the applicability condition of this power).

4.5 Declaratives: Declare

This declarative speech act changes the institutional reality by creating a new institutional fact. The notation $declare(i, j, s, cond, \varphi)$ reads "i declares to j in institution s that given condition *cond*, the fact φ is now established". The condition usually bears upon the speaker's role that empowers him to perform such a declaration.

Intentional Interpretation. This intentional interpretation is partly inspired from the intentional interpretation of an $inform(i, j, s, D_s\varphi)$. The feasibility precondition of a declaration is that the speaker does not believe the declared fact to be already established (indeed a declaration must create a **new** institutional fact). The rational effect (*i.e.* the intended effect) has two parts: the first one is to make the declared institutional fact true; the second part is similar to the rational effect of an inform about $D_s\varphi$, *i.e.* to make the hearer aware of this information. So:

$$FP(declare(i, j, s, cond, \varphi)) = \neg B_i D_s \varphi$$
$$RE(declare(i, j, s, cond, \varphi)) = B_j D_s \varphi \wedge D_s \varphi$$

Institutional Interpretation. The permission precondition to perform $declare(i, j, s, cond, \varphi)$ is that i really has the power to establish the declared fact φ by declaring it under the announced conditions *cond*. This power is locally granted by each specific institution to some agents depending on their role. For example the French republic grants the mayors the right to pronounce two people husband and wife, under the condition that they both consent to it. Thus an ordinary agent who is not mayor does not have this power, so that he is not allowed to pronounce marriages.

$$PermC(declare(i, j, s, cond, \varphi)) =$$
$$power(i, s, cond, declare(i, j, s, cond, \varphi), \varphi)$$

The explicit effect of a declaration is to store the declared fact in the institution, as well as the fact that the hearer is officially aware of this fact.

$$EE(declare(i, j, s, cond, \varphi)) = D_s\varphi \wedge D_s B_j D_s \varphi$$

This explicit effect is only obtained under the additional condition that *cond* is valid:

$$PowC(declare(i, j, s, cond, \varphi)) = cond$$

Example. For example a country can declare war to another one, by the voice of its representative that is empowered to do so, and under some conditions like the agreement of some counsellors. A mayor is empowered by its country to pronounce weddings under some conditions that the people are of age and consenting.

Citizens have to declare their income to the public treasury in order to calculate the amount of tax that they will pay. This is a declaration since the effect is a new institutional fact officially establishing one's declared income as being believed by him. Any citizen is empowered to do so. Moreover the law imposes a constraint on the generated commitment, that is an obligation to believe this income to be true. Thereby if the declared income was false the citizen is liable for prosecution and sanctions.

4.6 Example of Reasoning with Our Action Laws

This example is situated in the context of a B2B exchange (in institution $b2b$) between a buyer b and a seller s. The seller intends potential clients to know the prices of his products, *e.g.* $I_s B_b p$. With our relaxed feasibility precondition, he can use an assertive speech act whatever the context. Though if the buyer has already been informed of the prices before ($D_{b2b} B_s B_b p$), the seller is not permitted to inform him again. Thus if he informs him anyway, according to IEL_α he violates an obligation. This can be detected by other agents, and specific rules of the institution may specify sanctions to compensate this. Being aware of such pre-specified sanctions, an agent can deliberately choose to violate an obligation if the intended outcome (here, that clients be aware of the seller's offer) is more important than the incurred sanction. This shows the importance of having both intentional and institutional semantics of speech acts, to allow agents to reason about the relative importance of their goals and their obligations, in order to make an appropriate decision.

5 Detailed Comparison with Other Work

In this section we compare our semantics of speech acts with those proposed by Fornara and Colombetti, and by Lorini *et al.* (that we have presented above).

5.1 Concept of Commitment

We have shown before that what we mean by commitment in this work is a ratified mental attitude, *i.e.* a mental attitude (belief or intention) stored in the institution. This notion is similar to Fornara and Colombetti's commitment that is also a public concept, except that we have not made explicit its creditor. Actually the debtor is committed towards the whole institution, but an implicit creditor can sometimes be found in the content φ of the commitment. For example if agent i promises to j to pay him, he commits himself to a proposition involving agent j, expressing that j will be payed at some future instant. The creditor can sometimes be found in the sanction associated with the violation of the commitment, too; for example the obligation to pay damages to an agent.

Our notion of ratified mental attitude is also similar to Gaudou *et al.* 's notion of acceptance, because it must influence the agent's behaviour and utterances. Indeed, the agent's ratified mental attitudes are mental attitudes that he has expressed, that are stored in the institution, and to which he must conform while subsequently acting and speaking, even if they are not consistent with his real mental attitudes. For example an agent who promises that he has seen a given movie must then be able to talk about it in order to be consistent with his promise; if he is unable to narrate the end of the movie one can notice that he is contradicting his commitment.

5.2 Notion of Institution

By institution we mean a set of rules and facts that are adopted by a group of agents (the members of the institution). This seems to be a more generic notion than Lorini *et al.* 's concept of informal institution, since it accounts for this particular kind of institutions but also for various other ones: laws of a country, rules of a game, contract between businesses, social norms of a culture... In particular it allows to have institutional rules that are ignored by the members of the institution, what is the case for law for example, since one cannot be aware of the whole set of laws of his country, while he is one of its citizens. Fornara and Colombetti do not make explicit the institutional context in which their speech acts are interpreted, so we believe that they also consider a kind of "ordinary communication" institutional context.

In our view informal institutions are described by a specific set of facts and rules, determining their specific functioning. In particular the fact that all agents must accept a fact for it to become institutional is a particular institutional law. In other kinds of institutions, facts must be adopted by a majority of members (voting to create a law or to elect the president for example), or the opinion of one single member can suffice (the referee is always right). Thus we cannot adopt such an hypothesis in our account. Indeed on the contrary we consider the generic interpretation of speech acts in any institution s. More specific rules can be additionally specified in each particular interpretation, but the object of this paper is to identify for each category of speech acts the features that are common to their institutional interpretations whatever the institutional context.

6 Conclusion

In this work we have provided an expressive logical framework blending the agents' mental attitudes (beliefs, intentions) with their social attitudes (obligations, institutional facts and powers...). To illustrate its expressivity, please notice that our framework allows to represent some forms of contrary-to-duty obligations-to-do. Such obligations take the form:

$$O\mathit{after}_{\alpha}\bot \rightarrow \mathit{after}_{\alpha}O\mathit{done}_{\mathit{repair}_{\alpha}}\top$$

where repair_{α} is the contrary-to-duty obligation associated to the violation of the obligation to refrain from doing α. This means that if it is forbidden to perform α, then after α it is obligatory to perform a repairing action repair_{α}.

We have then used this framework to provide a semantics for an agent communication language based on FIPA-ACL but relaxing its widely criticised too constraining feasibility conditions, and adding permission preconditions. This way, agents can choose to perform forbidden speech acts but would then be liable to sanctions in the corresponding institution. Our ACL semantics also includes new speech acts (commissives and declaratives). It generalises existing approaches by unifying the intentional and institutional dimensions in one single framework, while strongly distinguishing them; moreover it allows to consider various kinds of institutional contexts; finally it provides action laws taking both dimensions into account.

In future work we intend to improve the institutional and intentional semantics of speech acts by accounting for deadlines. Various researchers [4,12,9] have shown that an important feature of obligations to perform an action is the deadline before which this action must be performed, that is essential to be able to assess the violation or fulfillment of such obligations. Though for the sake of simplicity we have omitted deadlines in this paper. An idea to manage them in future work could be to use existing formalisations of norms with deadlines, or to ground on linear temporal logic with until and since operators [5].

Finally we would like to mention that our framework for the institutional interpretation of speech acts has been successfully implemented into institutional agents that have been used in a prototype of industrial application: a multi-agent mediation platform for automated business to business exchanges [2].

References

1. Boella, G., Damiano, R., Hulstijn, J., van der Torre, L.: Role-based semantics for agent communication: embedding of the 'mental attitudes' and 'social commitments' semantics. In: AAMAS 2006, Hakodate, Hokkado, Japon (2006)
2. Bourge, F., Picant, S., Adam, C., Louis, V.: A multi-agent mediation platform for automatic b2b exchanges. In: ESAW 2008 (2008) (demonstration)
3. Broersen, J., Dastani, M., van der Torre, L.: Beliefs, obligations, intentions, and desires as components in an agent architecture. International Journal of Intelligent Systems 20(9), 893–919 (2005)
4. Broersen, J., Dignum, F., Dignum, V., Meyer, J.-J.C.: Designing a deontic logic of deadlines. In: Lomuscio, A., Nute, D. (eds.) DEON 2004. LNCS (LNAI), vol. 3065, pp. 43–56. Springer, Heidelberg (2004)
5. Burgess, J.: Logic and time. The Journal of Symbolic Logic 44(4), 566–582 (1979)
6. Burgess, J.: Basic tense logic. In: Handbook of philosophical logic, 2nd edn., vol. 7, pp. 1–42. Kluwer Academic Publishers, Dordrecht (2002)
7. Castelfranchi, C.: Commitments: From individual intentions to groups and organizations. In: ICMAS 1995, San Francisco, pp. 41–48 (1995)
8. Chaib-draa, B., Labrie, M.-A., Bergeron, M., Pasquier, P.: Diagal: An agent communication language based on dialogues games and sustained by social commitments. Journal of Autonomous Agent and Multi-Agent Systems 13, 61–95 (2006)
9. Demolombe, R., Bretier, P., Louis, V.: Norms with deadlines in dynamic deontic logic. In: ECAI 2006, pp. 751–752 (2006)

10. Demolombe, R., Louis, V.: Norms, institutional power and roles: towards a logical framework. In: Esposito, F., Raś, Z.W., Malerba, D., Semeraro, G. (eds.) ISMIS 2006. LNCS (LNAI), vol. 4203, pp. 514–523. Springer, Heidelberg (2006)
11. Demolombe, R., Louis, V.: Speech acts with institutional effects in agent societies. In: Goble, L., Meyer, J.-J.C. (eds.) DEON 2006. LNCS (LNAI), vol. 4048, pp. 101–114. Springer, Heidelberg (2006)
12. Dignum, F., Kuiper, R.: Obligations and dense time for specifying deadlines. In: HICSS, vol. 5, pp. 186–195. IEEE Computer Society, Los Alamitos (1998)
13. Dignum, F., Morley, D., Sonenberg, E., Cavedon, L.: Towards socially sophisticated bdi agents. In: ICMAS 2000, pp. 111–118 (2000)
14. Dignum, F., Weigand, H.: Communication and deontic logic. In: Wieringa, R., Feenstra, R. (eds.) Information Systems, correctness and reusability, pp. 242–260. World Scientific, Singapore (1995)
15. Finin, T., Fritzson, R., McKay, D., McEntire, R.: KQML as an agent communication language. In: Int. conf. Information and knowledge management (1994)
16. FIPA. The foundation for intelligent physical agents, http://www.fipa.org
17. Fornara, N., Colombetti, M.: A commitment-based approach to agent communication. Applied Artificial Intelligence 18(9-10), 853–866 (2004)
18. Gaudou, B., Herzig, A., Longin, D., Nickles, M.: A new semantics for the fipa agent communication language based on social attitudes. In: Brewka, G., Coradeschi, S., Perini, A., Traverso, P. (eds.) ECAI, pp. 245–249. IOS Press, Amsterdam (2006)
19. Gaudou, B., Longin, D., Lorini, E., Tummolini, L.: Anchoring institutions in agents' attitudes: towards a logical framework for autonomous multi-agent systems. In: AAMAS 2008 (2008)
20. Jones, A., Carmo, J.: Deontic Logic and Contrary-to-duties. In: Handbook of philosophical logic, vol. 8, pp. 265–343. Kluwer Academic Publishers, Dordrecht (2002)
21. Jones, A., Sergot, M.: A formal characterisation of institutionalised power. Journal of the interest group in pure and applied logics 4(3) (1996)
22. Lorini, E., Longin, D., Gaudou, B.: The institutional dimension of speech acts: a logical approach based on the concept of acceptance. Research report, IRIT (2008)
23. Rao, A., Georgeff, M.: Modeling rational agents within a BDI-architecture. In: KR 1991 (1991)
24. Sadek, D.: A study in the logic of intention. In: KR 1992 (1992)
25. Searle, J.R.: Speech acts: An essay in the philosophy of language. Cambridge Univ. Press, Cambridge (1969)
26. Singh, M.: An ontology for commitments in multiagent systems: Towards a unification of normative concepts (1999)
27. Singh, M.P.: A social semantics for agent communication languages. In: Dignum, F.P.M., Greaves, M. (eds.) Issues in Agent Communication. LNCS, vol. 1916, pp. 31–45. Springer, Heidelberg (2000)
28. Wooldridge, M.: Reasoning about rational agents. MIT Press, Cambridge (2000)

Tableaux for Acceptance Logic

Mathijs de Boer[1], Andreas Herzig[2], Tiago de Lima[3], and Emiliano Lorini[2]

[1] University of Luxembourg, Luxembourg
[2] IRIT, University of Toulouse 3, France
[3] Eindhoven University of Technology, The Netherlands

Abstract. We continue the work initiated in [1,2,3], where the acceptance logic, a modal logic for modelling individual and collective acceptances was introduced. This logic is aimed at capturing the concept of acceptance *qua* member of an institution as the kind of attitude that agents are committed to when they are "functioning as members of an institution". Acceptance logic can also be used to model judgement aggregation: it deals with how a collective acceptance of the members of an institution about a certain fact φ is created from the individual acceptances of the members of the institution. The contribution of this paper is to present a tableau method for the logic of acceptance. The method automatically decides whether a formula of the logic of acceptance is satisfiable thereby providing an automated reasoning procedure for judgement aggregation in the logic of acceptance.

Keywords: Semantic tableaux method, acceptance logic, judgement aggregation, discursive dilemma.

1 Introduction

The notion of 'acceptance' has been extensively studied in philosophy and social sciences where several authors have distinguished it from the classical notion of belief (see [4,5,6] for instance). Other authors have been interested in studying the foundational role of acceptance in the existence and in the dynamics of groups and institutions. It has been stressed in [7] (see also [8]) that the existence and the dynamics of an institution depend on the acceptances of the norms and the rules of the institution by the members of the institution. For example, for a certain norm to be a norm of institution x, all members of institution x must accept such norm to be valid. This relationship between acceptance and institutions was already emphasised in the philosophical doctrine of Legal Positivism [9]. According to Hart, the foundations of a normative system or institution consist of adherence to, or acceptance of, an ultimate rule of recognition by which the validity of any rule of the institution may be evaluated.[1]

In some recent works [1,2,3] we have presented a logical framework in which such relationship between acceptances and institutions can be formally studied.

[1] In Hart's theory, the rule of recognition is the rule which specifies the ultimate criteria of validity in a legal system.

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 85–100, 2010.

We conceive institutions as rule-governed social practices on the background of which the agents reason. For example, take the case of a game like Cluedo. The institutional context is the rule-governed social practice which the agents conform to in order to be competent players and on the background of which agents reason. In the context of Cluedo, an agent accepts that something has happened *qua* player of Cluedo (e.g., the agent accepts that Mrs. Red is the murderer *qua* player of Cluedo). Our logic is aimed at capturing the state of acceptance *qua* member of an institution as the kind of acceptance one is committed to when one is "functioning as a member of the institution" [7]. Moreover, it enables to formalise the concept of 'collective acceptance' of groups of agents. Following [10,7], we conceive a collective acceptance held by a set of agents G *qua* members of a certain institution x as the kind of acceptance the agents in G are committed to when they are "functioning together as members of this institution". For example, in the context of Greenpeace agents (collectively) accept that their mission is to protect the Earth *qua* members of Greenpeace. The state of acceptance *qua* members of Greenpeace is the kind of acceptance these agents are committed to when they are functioning together as members of Greenpeace. Thus, in our logical framework a collective acceptance by a set of agents G is based on the identification of the agents in G as members of a certain institution (or group, team, organisation, etc.) and on the fact that the agents in G recognise each other as members of the same institution (or group, team, organisation, etc.).

More recently [11], we have shown that our logic of acceptance can also be applied to modelling some interesting aspects of judgement aggregation. In the logic of acceptance the problem of judgement aggregation is a particular case of the problem of explaining how collective acceptance of the members of a certain group about a certain fact φ is created from the individual acceptances of the members of this group.

The contribution of this article is to present a tableau method for the logic of acceptance we introduced in [1,2]. The method automatically decides whether a formula of the logic of acceptance is satisfiable thereby providing an automated reasoning procedure for making judgement aggregation in modal logic.

The remainder of the paper is organised as follows. First, in Section 2 we briefly present acceptance logic. Then, in Section 3 we present our tableau method. In Section 4 we apply it to a classical scenario in judgment aggregation, the so-called Discursive Dilemma [12,13]. And finally, Section 5 concludes.

2 Acceptance Logic

The logic AL (*Acceptance Logic*) was introduced in [1,2]. It allows to express that some agents identify themselves as members of a certain institution and what (groups of) agents accept while functioning together as members of an institution. The principles of AL clarify the relationships between individual acceptance (acceptances of individual agents) and collective acceptance (acceptances of groups of agents).

2.1 Syntax

Assume a finite non-empty set X of labels denoting institutional contexts, a finite non-empty set N of labels denoting agents and a countable set P of atomic formulae. We use $2^{N\star}$ to denote the set $2^N \setminus \emptyset$.

The language $\mathcal{L}_{\mathrm{AL}}$ of acceptance logic is the set of formulae φ defined by the following BNF:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathrm{A}_{G:x}\varphi$$

where p ranges over P, G ranges over $2^{N\star}$, and x ranges over X.

The other classical Boolean connectives $\wedge$, $\rightarrow$, $\leftrightarrow$, $\top$ (tautology) and $\bot$ (contradiction) are defined using $\vee$ and $\neg$ in the usual manner. And for simplicity, we write $i{:}x$ instead of $\{i\}{:}x$.

The formula $\mathrm{A}_{G:x}\varphi$ reads 'the agents in G accept that φ while functioning together as members of institution x'. For example, $\mathrm{A}_{G:Greenpeace}\mathit{protectEarth}$ expresses that the agents in G accept that the mission of Greenpeace is to protect the Earth while functioning as activists in the context of Greenpeace; and $\mathrm{A}_{i:Catholic}\mathit{PopeInfallible}$ expresses that agent i accepts that the Pope is infallible while functioning as a Catholic in the context of the Catholic Church.

The same agent may accept contradictory propositions in two different contexts. For example, while functioning as a Catholic, agent i accepts that killing is forbidden, and while functioning as a soldier i accepts that killing is allowed.

The formula $\mathrm{A}_{G:x}\bot$ has to be read 'agents in G are not functioning together as members of institution x'. This means that we assume that functioning as a member of an institution is, at least in this minimal sense, a rational activity. Conversely, $\neg\mathrm{A}_{G:x}\bot$ has to be read 'agents in G are functioning together as members of institution x'. Thus, $\neg\mathrm{A}_{G:x}\bot \wedge \mathrm{A}_{G:x}\varphi$ stands for 'agents in G are functioning together as members of institution x and they accept that φ while functioning together as members of x' or simply 'agents in G accept that φ *qua* members of institution x'. This is a case of group acceptance. For the individual case, the formula $\neg\mathrm{A}_{i:x}\bot \wedge \mathrm{A}_{i:x}\varphi$ has to be read 'agent i accepts that φ *qua* member of institution x'.

2.2 Semantics and Axiomatisation

We use a standard possible worlds semantics. Let the set of all pairs of non-empty sets of agents and institutional contexts be $\Delta = \{G{:}x \mid G \in 2^{N\star} \text{ and } x \in X\}$. An acceptance model is a triple $\langle W, \mathcal{A}, \mathcal{V}\rangle$ where: W is a non-empty set of possible worlds, $\mathcal{A} : \Delta \rightarrow W \times W$ maps every $G{:}x \in \Delta$ to a relation $\mathcal{A}(G{:}x)$ between possible worlds in W and $\mathcal{V} : P \rightarrow 2^W$ is valuation function associating a set of possible worlds $\mathcal{V}(p) \subseteq W$ to each atomic formula p of P.

Instead of $\mathcal{A}(G{:}x)$ we write $\mathcal{A}_{G:x}$, and we use $\mathcal{A}_{G:x}(w)$ to denote the set $\{w' \mid \langle w, w'\rangle \in \mathcal{A}_{G:x}\}$. $\mathcal{A}_{G:x}(w)$ is the set of worlds that is acceptable by the agents in G while functioning together as members of institution x.

Given $M = \langle W, \mathcal{A}, \mathcal{V}\rangle$ and $w \in W$, the pair $\langle M, w\rangle$ is a pointed acceptance model. The satisfaction relation $\models$ between formulae of $\mathcal{L}_{\mathrm{AL}}$ and pointed

acceptance models $\langle M, w\rangle$ is defined as usual for atomic propositions, negation and disjunction. The satisfaction relation for acceptance operators is the following:

$$M, w \models \mathrm{A}_{G:x}\varphi \quad \text{iff} \quad M, w' \models \varphi \text{ for all } w' \in \mathcal{A}_{G:x}(w)$$

Validity of a formula φ (noted: $\models \varphi$) is defined as usual.

The axiomatisation of AL is presented in Figure 1. As usual, the K-principles are the axioms and inference rules of the basic modal logic K.

(K)	All K-principles for the operators $\mathrm{A}_{G:x}$	
(4∗)	$\mathrm{A}_{G:x}\varphi \rightarrow \mathrm{A}_{H:y}\mathrm{A}_{G:x}\varphi$	(if $H \subseteq G$)
(5∗)	$\neg\mathrm{A}_{G:x}\varphi \rightarrow \mathrm{A}_{H:y}\neg\mathrm{A}_{G:x}\varphi$	(if $H \subseteq G$)
(Inc)	$(\neg\mathrm{A}_{G:x}\bot \wedge \mathrm{A}_{G:x}\varphi) \rightarrow \mathrm{A}_{H:x}\varphi$	(if $H \subseteq G$)
(Una)	$\mathrm{A}_{G:x}(\bigwedge_{i\in G} \mathrm{A}_{i:x}\varphi \rightarrow \varphi)$	

Fig. 1. Axiomatisation of acceptance logic

Axioms **4∗** and **5∗** are introspection axioms: when the agents in a set G function together as members of institution x then, for all $y \in X$ and all $H \subseteq G$, the agents in H have access to all the facts that are accepted (or that are not accepted) by the agents in G. In particular, if the agents in G (do not) accept that φ while functioning together as members of institution x then, while functioning together as members of institution x, the agents of every subset H of G accept that agents in G (do not) accept that φ.

Axiom **Inc** says that, if the agents in G accept that φ *qua* members of institution x then every subset H of G accepts φ while functioning together as members of institution x. This means that what is accepted by the agents in G *qua* members of institution x are necessarily accepted by agents in all of G's subsets with respect to the same institutional context x. Axiom **Inc** describes the *top down* process leading from G's collective acceptance to the individual acceptances of G's members.

Axiom **Una** expresses a unanimity principle according to which the agents in G, while functioning together as members of institution x, accept that if each of them individually accepts that φ while functioning as member of x, then φ is the case. This axiom describes the *bottom up* process leading from individual acceptances of the members of G to the collective acceptance of the group G.

In order to make our axioms valid we impose the following constraints on acceptance models, for any world $w \in W$, institutional context $x \in X$, and groups $G, H \in 2^{N\star}$ such that $H \subseteq G$:

(**C.4∗**)	if $w_2 \in \mathcal{A}_{H:y}(w_1)$ and $w_3 \in \mathcal{A}_{G:x}(w_2)$ then $w_3 \in \mathcal{A}_{G:x}(w_1)$;
(**C.5∗**)	if $w_2 \in \mathcal{A}_{H:y}(w_1)$ and $w_3 \in \mathcal{A}_{G:x}(w_1)$ then $w_3 \in \mathcal{A}_{G:x}(w_2)$;
(**C.Inc**)	if $\mathcal{A}_{G:x}(w) \neq \emptyset$ then $\mathcal{A}_{H:x}(w) \subseteq \mathcal{A}_{G:x}(w)$;
(**C.Una**)	if $w_2 \in \mathcal{A}_{G:x}(w_1)$ then $w_2 \in \bigcup_{i\in G} \mathcal{A}_{i:x}(w_2)$.

Axiom **4∗** corresponds to semantic constraint **C.4∗**, Axiom **5∗** corresponds to **C.5∗**, Axiom **Inc** to **C.Inc**, and **Una** to **C.Una** (in the sense of correspondence theory). We also note that **C.4∗** and **C.5∗** together are equivalent to the following semantic constraint: if $w_2 \in \mathcal{A}_{H:y}(w_1)$ then $\mathcal{A}_{G:x}(w_1) = \mathcal{A}_{G:x}(w_2)$. Thus, the acceptance models considered here are exactly the same as proposed in [1,2,3]. The theorem below has been shown in [2].

Theorem 1. *The axiomatisation in Figure 1 is sound and complete with respect to the class of acceptance models satisfying constraints* **C.4∗***,* **C.5∗***,* **C.Inc** *and* **C.Una***.*

2.3 Discussion about the Monotonicity Principle

One may think that the following principle would be desirable in Acceptance Logic:

$$\textbf{(Mon)} \quad \neg \mathrm{A}_{G:x}\bot \rightarrow \neg \mathrm{A}_{H:x}\bot \quad (\text{if } H \subseteq G)$$

which corresponds to semantic constraint: if $\mathcal{A}_{G:x}(w) \neq \emptyset$ then $\mathcal{A}_{H:x}(w) \neq \emptyset$. Principle **Mon** expresses a property of monotonicity about institution membership. It was also discussed from a different perspective in our previous works on Acceptance Logic [1,2].

We prefer not including **Mon** in the current version of AL because we are interested in strong notions of 'constituted group' and 'group identification' which are formally expressed by constructions $\neg \mathrm{A}_{G:x}\bot$. As we said above, $\neg \mathrm{A}_{G:x}\bot$ means "the agents in G are functioning together as members of the institution x" or, stated differently, "G constitutes a group of members of the institution x". We suppose here that the latter sentences just express that: every agent in G identifies himself as a member of institution x and recognizes G as a *group of members* of institution x. Under this assumption, **Mon** is not valid. The following example illustrates this point. Imagine that the eleven agents in $\{1, 2, \ldots, 11\}$ constitute a football team (i.e. $\neg \mathrm{A}_{\{1,2,\ldots,11\}:team}\bot$). This means that every agent in $\{1, 2, \ldots, 11\}$ identifies himself as a member of the football team and recognizes $\{1, 2, \ldots, 11\}$ as a football team. This does not entail that $\{1, 2, \ldots, 10\}$ constitute a football team (i.e. $\neg \mathrm{A}_{\{1,2,\ldots,10\}:team}\bot$). Indeed, it is not the case that every agent in $\{1, 2, \ldots, 10\}$ recognizes $\{1, 2, \ldots, 10\}$ as a football team. (Only ten players do not constitute a football team!).

It worth noting that **Mon** becomes a reasonable principle under a different reading of the construction $\neg \mathrm{A}_{G:x}\bot$. Namely, suppose that $\neg \mathrm{A}_{G:x}\bot$ just means: every agent in G identifies himself as a member of institution x and recognises every agent in the set of agents G as a member of institution x. Under this assumption, $\neg \mathrm{A}_{G:x}\bot$ should imply $\neg \mathrm{A}_{H:x}\bot$, for $H \subseteq G$.

3 The Tableau Method

In this section we present a proof method for AL that uses semantic tableaux. As a typical tableaux method, given a formula φ, it systematically tries to construct a model for it. When it fails, φ is inconsistent and thus, its negation is valid.

Each formula in the tableau is prefixed by a natural number that stands for a possible world in the model under construction, similar to the notion used by Fitting ([14, Chapter 8]).

Definition 1 (Labelled formula). *A* labelled formula *is a pair of the form* $\langle n, \varphi \rangle$ *such that* $n \in \mathbb{N}$ *and* $\varphi \in \mathcal{L}_{\mathrm{AL}}$.

Our method also builds the relations $\mathcal{A}_{G:x}$ between possible worlds that form the model. These relations are represented in tableau by means of arrows between possible worlds, which are represented as triples of the form $\langle G{:}x, n, n' \rangle$. That is, the tableau contains a set of labelled formulae and also a set of such triples, as defined in the sequel.

Definition 2 (Branch). *A* branch *is a pair of the form* $\langle L, S \rangle$ *such that* L *is a set of labelled formulae and* $S \subseteq (\Delta \times \mathbb{N} \times \mathbb{N})$.

Definition 3 (Tableau). *Let* $\varphi \in \mathcal{L}_{\mathrm{AL}}$. *A tableau for* φ *is a set of branches* T *inductively defined as follows:*

- $T = \{\langle \{\langle 0, \varphi \rangle\}, \emptyset \rangle\}$. *This is called the* initial tableau for φ.
- $T' = (T \setminus \{\langle L, S \rangle\}) \cup \{\langle L'_1, S'_1 \rangle, \langle L'_2, S'_2 \rangle, \ldots, \langle L'_n, S'_n \rangle\}$, *where* T *is a tableau for* φ *containing the branch* $\langle L, S \rangle$ *and each* $\langle L'_i, S'_i \rangle$ *is a branch generated by one of the* tableau rules *defined below: (A more standard presentation of some of these tableau rules is given in Figure 2.)*
  - **(R.¬)** *If* $\langle n, \neg\neg\varphi \rangle \in L$ *then generate* $L'_1 = L \cup \{\langle n, \varphi \rangle\}$ *and* $S'_1 = S$.
  - **(R.∧)** *If* $\langle n, \varphi_1 \wedge \varphi_2 \rangle \in L$ *then generate* $L'_1 = L \cup \{\langle n, \varphi_1 \rangle, \langle n, \varphi_2 \rangle\}$ *and* $S'_1 = S$.
  - **(R.∨)** *If* $\langle n, \neg(\varphi_1 \wedge \varphi_2) \rangle \in L$ *then generate* $L'_1 = L \cup \{\langle n, \neg\varphi_1 \rangle\}$ *and* $S'_1 = S$; *and also* $L'_2 = L \cup \{\langle n, \neg\varphi_2 \rangle\}$ *and* $S'_2 = S$.
  - **(R.□)** *if* $\langle n, \mathrm{A}_{G:x}\varphi \rangle \in L$ *and* $\langle G{:}x, n, n' \rangle \in S$ *then generate* $L'_1 = L \cup \{\langle n', \varphi \rangle\}$ *and* $S'_1 = S$.
  - **(R.◇)** *if* $\langle n, \neg\mathrm{A}_{G:x}\varphi \rangle \in L$ *then generate* $L'_1 = L \cup \{\langle n', \neg\varphi \rangle\}$ *and* $S'_1 = S \cup \{\langle G{:}x, n, n' \rangle\}$, *for some* n' *that does not occur in* L.
  - **(R.4∗)** *if* $H \subseteq G$ *and* $\langle H{:}y, n, n' \rangle, \langle G{:}x, n', n'' \rangle \in S$ *then generate* $L'_1 = L$ *and* $S'_1 = S \cup \{\langle G{:}x, n, n'' \rangle\}$.
  - **(R.5∗)** *If* $H \subseteq G$ *and* $\langle H{:}y, n, n' \rangle, \langle G{:}x, n, n'' \rangle \in S$ *then generate* $L'_1 = L$ *and* $S'_1 = S \cup \{\langle G{:}x, n', n'' \rangle\}$.
  - **(R.Inc)** *If* $H \subseteq G$ *and* $\langle H{:}x, n, n' \rangle \in S$ *then generate* $L'_1 = L$ *and* $S'_1 = S \cup \{\langle G{:}x, n, n' \rangle\}$; *and also* $L'_2 = L \cup \{\langle n, \mathrm{A}_{G:x}\bot \rangle\}$ *and* $S'_2 = S$.
  - **(R.Una)** *If* $G = \{i_1, \ldots, i_k\}$ *and* $\langle G{:}x, n, n' \rangle \in S$ *then generate* $L'_1 = L$ *and* $S'_1 = S \cup \{\langle \{i_1\}{:}x, n', n' \rangle\}$; $L'_2 = L$ *and* $S'_2 = S \cup \{\langle \{i_2\}{:}x, n', n' \rangle\}$; $\ldots$; $L'_k = L$ *and* $S'_k = S \cup \{\langle \{i_k\}{:}x, n', n' \rangle\}$.

Rules **R.**¬, **R.**∧ and **R.**∨ work exactly as for classical propositional logic. Rules **R.**□ and **R.**◇ work exactly as for modal logic K, as proposed, e.g., in [14]. The other rules are meant to ensure that the model to be created will be an acceptance model. Tableau Rule **R.4∗** implements semantic constraint **C.4∗**, Tableau Rule **R.5∗** implements constraint **C.5∗**, Rule **R.Inc** implements **C.Inc** and **R.Una** implements **C.Una**.

$$(\mathbf{R}.\Box)\quad \frac{\langle n, \mathrm{A}_{G:x}\varphi\rangle; \langle G{:}x, n, n'\rangle}{\langle n', \varphi\rangle;}$$

$$(\mathbf{R}.\Diamond)\quad \frac{\langle n, \neg\mathrm{A}_{G:x}\varphi\rangle;}{\langle n', \neg\varphi\rangle; \langle G{:}x, n, n'\rangle} \quad \text{for a new } n'$$

$$(\mathbf{R}.\mathbf{4}*)\quad \frac{; \langle H{:}y, n, n'\rangle, \langle G{:}x, n', n''\rangle}{; \langle G{:}x, n, n''\rangle} \quad \text{where } H \subseteq G$$

$$(\mathbf{R}.\mathbf{5}*)\quad \frac{; \langle H{:}y, n, n'\rangle, \langle G{:}x, n, n''\rangle}{; \langle G{:}x, n', n''\rangle} \quad \text{where } H \subseteq G$$

$$(\mathbf{R}.\mathbf{Inc})\quad \frac{; \langle H{:}x, n, n'\rangle}{; \langle G{:}x, n, n'\rangle \,|\, \langle n, \mathrm{A}_{G:x}\bot\rangle;} \quad \text{where } H \subseteq G$$

$$(\mathbf{R}.\mathbf{Una})\quad \frac{; \langle G{:}x, n, n'\rangle}{; \langle \{i_1\}{:}x, n', n'\rangle \,|\, \ldots \,|\, ; \langle \{i_k\}{:}x, n', n'\rangle} \quad \text{where } G = \{i_1, \ldots, i_k\}$$

Fig. 2. Tableau rules

Definition 4 (Closed tableau). *The set of labelled formulae L is* closed *if and only if $\{\langle n, \varphi\rangle, \langle n, \neg\varphi\rangle\} \subseteq L$, for some n and φ. A branch is closed if and only if its set of labelled formulae is closed. A tableau is closed if and only if all its branches are closed. A tableau is* open *if and only if it is not closed.*

Example 1. Now, let us see how the method can be used to show that the formula: $(\mathrm{A}_{ij:x}\mathrm{A}_{i:x}p \wedge \mathrm{A}_{ij:x}\mathrm{A}_{j:x}p) \rightarrow \mathrm{A}_{ij:x}p$ is valid in acceptance logic. As we will see, if there is a closed tableau for φ, then no model satisfies φ, which means that $\neg\varphi$ is valid. Therefore, if we provide a closed tableau for the negation of our formula above, we show its validity.

Such a tableau is given in Figure 3. Each line of the figure displays either a labelled formula, an arrow or both, which are the elements of the tableau branches. The number in parentheses on the left is used to identify the line. On the right, also in parentheses, we find the rule that generated that line, and what lines has been used in the application of such rule. For example, the labelled formula and arrow in line 6 have been generated by the application of $\mathbf{R}.\Diamond$, using the labelled formula in line 3. Also, lines 9 and 11 have been generated by the application of **R.Una**, using the arrow in line 6.

The input formula is in line 1 (we spelt out the abbreviations). The construction of the closed tableau started with the initial tableau for the input formula. The latter corresponds to line 1 alone. Then, a new tableau has been generated by the application of $\mathbf{R}.\wedge$ using the labelled formula in line 1. The latter corresponds to lines 1 and 2 together, and so on. When **R.Una** has been applied

(1) $0, (\mathrm{A}_{ij:x}\mathrm{A}_{i:x}p \wedge \mathrm{A}_{ij:x}\mathrm{A}_{j:x}p) \wedge \neg\mathrm{A}_{ij:x}p$
(2) $0, \mathrm{A}_{ij:x}\mathrm{A}_{i:x}p \wedge \mathrm{A}_{ij:x}\mathrm{A}_{j:x}p$ (**R**.$\wedge$: 1)
(3) $0, \neg\mathrm{A}_{ij:x}p$ (**R**.$\wedge$: 1)
(4) $0, \mathrm{A}_{ij:x}\mathrm{A}_{i:x}p$ (**R**.$\wedge$: 2)
(5) $0, \mathrm{A}_{ij:x}\mathrm{A}_{j:x}p$ (**R**.$\wedge$: 2)
(6) $1, \neg p$ $ij{:}x, 0, 1$ (**R**.$\Diamond$: 3)
(7) $1, \mathrm{A}_{i:x}p$ (**R**.$\Box$: 4, 6)
(8) $1, \mathrm{A}_{j:x}p$ (**R**.$\Box$: 5, 6)

(9) $i{:}x, 1, 1$ (**R**.**Una** : 6)	(11) $j{:}x, 1, 1$ (**R**.**Una** : 6)
(10) $1, p$ (**R**.$\Box$: 7, 9)	(12) $1, p$ (**R**.$\Box$: 8, 11)
closed (6, 10)	closed (6, 12)

Fig. 3. A tableau for Example 1

(1) $0, \neg\mathrm{A}_{i:x}p \wedge \mathrm{A}_{ij:x}\neg\mathrm{A}_{i:x}p$
(2) $0, \neg\mathrm{A}_{i:x}p$ (**R**.$\wedge$: 1)
(3) $0, \mathrm{A}_{ij:x}\neg\mathrm{A}_{i:x}p$ (**R**.$\wedge$: 1)
(4) $1, \neg p$ $i{:}x, 0, 1$ (**R**.$\Diamond$: 2)

(5) $ij{:}x, 0, 1$ (**R**.**Inc** : 4)		(16) $0, \mathrm{A}_{ij:x}\bot$ (**R**.**Inc** : 4)
(6) $1, \neg\mathrm{A}_{i:x}p$ (**R**.$\Box$: 3, 5)		
(7) $ij{:}x, 1, 1$ (**R**.**5**$*$: 5)		
(8) $2, \neg p$ $i{:}x, 1, 2$ (**R**.$\Diamond$: 6)		
(9) $ij{:}x, 1, 2$ (**R**.**Inc** : 8)	(14) $1, \mathrm{A}_{ij:x}\bot$ (**R**.**Inc** : 8)	
(10) $ij{:}x, 0, 2$ (**R**.**4**$*$: 4, 9)	(15) $1, \bot$ (**R**.$\Box$: 14, 7)	
(11) $2, \neg\mathrm{A}_{i:x}p$ (**R**.$\Box$: 3, 10)	closed (15)	
(12) $ij{:}x, 2, 2$ (**R**.**5**$*$: 9)		
(13) $3, \neg p$ $i{:}x, 2, 3$ (**R**.$\Diamond$: 11)		
$\vdots$		

Fig. 4. A tableau for Example 2

using the arrow in line 6, it generated two branches. This is represented in the figure by the vertical line dividing the tableau in two parts after line 8.

Example 2. On the other hand, if no closed tableau for φ exists, then φ is satisfiable. Let us see what happens when we try to generate a closed tableau for the formula: $\neg\mathrm{A}_{i:x}p \wedge \mathrm{A}_{ij:x}\neg\mathrm{A}_{i:x}p$ which is satisfiable. This is done in Figure 4. Note that one of the branches is closed. On the other hand, no rule can be applied in the rightmost branch, which means that this tableau will remain open. On the leftmost branch we have a rather different phenomenon. We can continue applying the same set of rules indefinitely. This will generate more branches that can be closed, but we can never close all of them. This means that we can also consider such a branch as an open one.

We proceed by proving soundness of the method. But first we need yet another definition.

Definition 5 (Satisfiable branch). *The branch $b = \langle L, S\rangle$ is* satisfiable *if and only if there exists an acceptance model $M = \langle W, \mathcal{A}, \mathcal{V}\rangle$ and a function $f : \mathbb{N} \to W$ such that:*

1. *$\langle f(n), f(n')\rangle \in \mathcal{A}_{G:x}$, for all $\langle G{:}x, n, n'\rangle \in S$; and*
2. *$M, f(n) \models \varphi$, for all $\langle n, \varphi\rangle \in L$.*

Theorem 2 (Soundness). *If there is a closed tableau for $\neg\varphi$ then φ is valid.*

Proof. We show that if φ is satisfiable then there is no closed tableau for φ. It is enough to show that all tableau rules preserve satisfiability. That is, it is enough to show that: if the branch $b = \langle L, S\rangle$ is satisfiable then the set of branches $B = \{b'_1, \ldots, b'_k\}$ generated by any tableau rule contains a satisfiable branch.

Indeed, and to see that it is enough, suppose that b'_i is satisfiable and closed. Then L'_i contains two labelled formulae $\langle n, \varphi\rangle$ and $\langle n, \neg\varphi\rangle$. Because b'_i is satisfiable, there exists an acceptance model M and a function f such that $M, f(n) \models \varphi$ and $M, f(n) \models \neg\varphi$, which is a contradiction.

Now, suppose that the branch b is satisfiable. The proof that the rules **R**.$\neg$ and **R**.$\wedge$ preserve satisfiability is straightforward and thus, left as an exercise to the reader. We proceed by showing that the modal rules preserve satisfiability.

R.$\Box$: $M, f(n) \models \mathrm{A}_{G:x}\varphi$ and $\langle f(n), f(n')\rangle \in \mathcal{A}_{G:x}$ (by hypothesis). Then $M, f(n') \models \varphi$ (by definition).

R.$\Diamond$: $M, f(n) \models \neg\mathrm{A}_{G:x}\varphi$ (by hypothesis). Then there exists $w' \in \mathcal{A}_{G:x}(f(n))$ such that $M, w' \models \neg\varphi$ (again, by definition). Now, consider the function $f' : \mathbb{N} \to W$ such that $f'(n) = f(n)$, for all n occurring in L, and $f'(n') = w'$. Then $M, f'(n'') \models \varphi''$, for all $\langle n'', \varphi''\rangle \in L$ (because n' does not occur in L), and $M, f'(n') \models \neg\varphi$.

R.**4***: Let $H \subseteq G$. $\langle f(n), f(n')\rangle \in \mathcal{A}_{H:y}$ and $\langle f(n'), f(n'')\rangle \in \mathcal{A}_{G:x}$ (by hypothesis). Then $\langle f(n), f(n'')\rangle \in \mathcal{A}_{G:x}$, since M is an acceptance model respecting **C**.**4***. Then, the branch $\langle L'_1, S'_1\rangle$ is satisfiable.

R.**5***: Let $H \subseteq G$. $\langle f(n), f(n')\rangle \in \mathcal{A}_{H:x}$ and $\langle f(n), f(n'')\rangle \in \mathcal{A}_{G:x}$ (by hypothesis). Then, $\langle f(n'), f(n'')\rangle \in \mathcal{A}_{G:x}$, since M is an acceptance model respecting **C**.**5***. Then, the branch $\langle L'_1, S'_1\rangle$ is satisfiable.

R.**Inc**: Let $H \subseteq G$. $M, f(n) \models \mathrm{A}_{G:x}\varphi$ and $\langle f(n), f(n')\rangle \in \mathcal{A}_{H:x}$ (by hypothesis). Note that $M, f(n') \not\models \bot$ (because we assume that the branch is satisfiable) then $M, f(n) \models \neg\mathrm{A}_{H:x}\bot$. The latter implies $\langle f(n), f(n')\rangle \in \mathcal{A}_{G:x}$ or $\mathcal{A}_{G:x}(f(n)) = \emptyset$, since M is an acceptance model respecting **C**.**Inc**. Therefore, one of the branches generated by **R**.**Inc** is satisfiable.

R.**Una**: Let $G = \{i_1, \ldots, i_k\}$. $\langle f(n), f(n')\rangle \in \mathcal{A}_{G:x}$ (by hypothesis). Then, $\langle f(n'), f(n')\rangle \in \mathcal{A}_{i_j:x}$ for some $1 \leq j \leq k$, since M is an acceptance model respecting **C**.**Una**. Therefore, one of the branches generated by **R**.**Una** is satisfiable. □

In the sequel we prove completeness. First though, we need another auxiliary definition.

Definition 6 (Saturated tableau). *Let T be a tableau for φ containing the branch $b = \langle L, S\rangle$. The branch b is 'saturated under the tableau rule ρ' if and*

only if L contains L'_i and S contains S'_i for some branch $\langle L'_i, S'_i\rangle$ generated by the application of the rule ρ to b. A branch is (simply) 'saturated' if and only if it is saturated under all tableau rules. A tableau is saturated if and only if all its branches are saturated.

Theorem 3 (Completeness). *If φ is valid then there exists a closed tableau for $\neg\varphi$.*

Proof. Let $\langle L, S\rangle$ be an open and saturated branch of the tableau. We build a model $M = \langle W, \mathcal{A}, \mathcal{V}\rangle$ such that $W = \{n \mid \langle n, \varphi\rangle \in L\}$, for some φ; $\mathcal{A}_{G:x}(n) = \{n' \mid \langle G{:}x, n, n'\rangle \in S\}$; and $\mathcal{V}(p) = \{n \mid \langle n, p\rangle \in L\}$.

Clearly, M is an acceptance model, because the branch is saturated under **R**.**4**$*$, **R**.**5**$*$, **R**.**Inc** and **R**.**Una**.

Now, we show that for all $\langle n, \varphi\rangle \in L$, $M, n \models \varphi$. It is done by induction on the structure of φ.

There are two cases in the induction base. (1) $\varphi = p$, i.e., $\langle n, p\rangle \in L$. Then $n \in \mathcal{V}(p)$, iff $M, n \models p$ (by definition). (2) $\varphi = \neg p$, i.e., $\langle n, \neg p\rangle \in L$. Then $\langle n, p\rangle \notin L$, because L is open. Then $n \notin \mathcal{V}(p)$, iff $M, n \models \neg p$ (by definition).

There are five cases in the induction step. (1) $\varphi = \neg\neg\varphi_1$, i.e., $\langle n, \neg\neg\varphi_1\rangle \in L$. Then $\langle n, \varphi_1\rangle \in L$, because L is saturated under **R**.$\neg$. Then $M, n \models \varphi_1$, by induction hypothesis, iff $M, n \models \neg\neg\varphi_1$, by definition. Cases (2) $\varphi = \varphi_1 \wedge \varphi_2$ and (3) $\varphi = \neg(\varphi_1 \wedge \varphi_2)$, are shown analogously as case (1) using rules **R**.$\wedge$ and **R**.$\vee$, respectively. They are left as an exercise to the rader. (4) $\langle n, \mathrm{A}_{G:x}\varphi\rangle \in L$. Then for all $n' \in W$, if $\langle G{:}x, n, n'\rangle \in S$, then $\langle n', \varphi\rangle \in L$ (because L is saturated under rule **R**.$\Box$). Then for all $n' \in W$, if $n' \in \mathcal{A}_{G:x}(n)$, then $M, n' \models \varphi$, by induction hypothesis. Therefore, $M, n \models \mathrm{A}_{G:x}\varphi$. (5) $\langle n, \neg\mathrm{A}_{G:x}\varphi\rangle \in L$. Then there is $n' \in W$ such that $\langle G{:}x, n, n'\rangle \in S$ and $\langle n', \neg\varphi\rangle \in L$ (because L is saturated under rule **R**.$\Diamond$). Then there is $n' \in W$ such that $n' \in \mathcal{A}_{G:x}(n)$ and $M, n' \models \neg\varphi$, by induction hypothesis. Therefore, $M, n \models \neg\mathrm{A}_{G:x}\varphi$. □

Theorem 4 (Termination). *There exists an implementation of the tableau method that halts for every input formula φ.*

Proof (Sketch). We assume an implementation of the tableau method that employs a 'loop-test'. That is, a procedure that, once the latest generated tableau is saturated under all rules but **R**.$\Diamond$, verifies whether the application of the latter rule to the witness formula ψ will generate a world such that the set of labelled formulae having this world as label and the set of arrows involving this world will be included in the respective sets for a different world already present in the branch. If it is the case, then ψ is marked and **R**.$\Diamond$ will not be applied using this formula any more.

Then, the argument for termination goes as for logic S4 (as used, e.g., in [15,16]): the formulae generated by the rules are in a finite set S, and therefore only a finite number of different nodes can be generated by the tableaux procedure. The only difference is that here the finite set S is not just the set of sub-formulae of the initial formula φ as for S4, but its closure, which is the set of sub-formulae of φ union the set of $\mathrm{A}_{G:x}\bot$ such that G and x occur in φ (due to rule **R**.**Inc**). Moreover, the set of labels of every arrow is finite. □

4 An Example: The Discursive Dilemma

In the recent years many researchers in philosophy, computer science and political sciences have been working on the issue of judgement aggregation (e.g., [17,18,19,20,21,22]). The problem is: How can a group of individuals aggregate the group members' individual judgements on some interconnected propositions into corresponding collective judgements on these propositions? Such problems occur in different social and legal contexts like committees, legislatures, judiciaries and expert panels.

Our logic of acceptance is a formal framework in which some important aspects of judgement aggregation can be modelled. Moreover, the tableau method for the logic of acceptance presented in Section 3 provides an interesting solution for making automated reasoning about judgement aggregation.

In the logic of acceptance the problem of judgement aggregation is a particular case of the problem of explaining how collective acceptance of the members of a certain group in an institutional context x about a certain fact φ is created from the individual acceptances in x of the members of the same group.

We here consider a well-known problem in judgement aggregation called 'doctrinal paradox' or 'discursive dilemma' [12,13]. The scenario of the discursive dilemma is a three-member court which has to judge whether a defendant is liable for a breach of contract. According to the legal doctrine, the defendant is liable (lia) if and only if he did a certain action (act) and he had a contractual obligation not to do this action (obl). This is expressed in propositional logic by the connection rule $lia \leftrightarrow (act \wedge obl)$. The three judges use majority rule to decide on this issue. The opinions of the three judges are given in Table 1.

Table 1. Discursive dilemma

	act	obl	$lia \leftrightarrow (act \wedge obl)$	lia
Judge 1	yes	yes	yes	yes
Judge 2	yes	no	yes	no
Judge 3	no	yes	yes	no
Majority	yes	yes	yes	no

It is supposed that all the judges accept the rule $lia \leftrightarrow (act \wedge obl)$. Judge 1 accepts both act and obl and, by the connection rule, he accepts lia. Judge 2 accepts act and rejects obl and, by the connection rule, he rejects lia. Finally, judge 3 rejects act and accepts obl and, by the connection rule, he rejects lia. If the three judges apply a majority rule on each proposition then they face a paradox. Indeed, a majority accepts act, a majority accepts obl, a majority accepts the connection rule $lia \leftrightarrow (act \wedge obl)$. But the majority rejects lia. Thus, when majority voting is applied to each single proposition it yields an inconsistent collective set of judgements (see the last row in Table 1). Note that this inconsistency occurs even though the sets of judgements of the individual judges are all consistent.

Let us now show how the discursive dilemma can be formalised in the logic of acceptance.

We first suppose that 1, 2 and 3 *qua* judges of the court accept the connection rule:

$$\neg \mathrm{A}_{123:c} \bot \tag{1}$$

$$\mathrm{A}_{123:c}(\mathit{lia} \leftrightarrow (\mathit{act} \wedge \mathit{obl})) \tag{2}$$

Then, we have that judge 1 announces that, *qua* judge of the court, he accepts $\mathit{act} \wedge \mathit{obl}$. Judge 2 announces that, *qua* judge of the court, he accepts $\mathit{act} \wedge \neg \mathit{obl}$. Judge 3 announces that, *qua* judge of the court, he accepts $\neg \mathit{act} \wedge \mathit{obl}$. This has the following effect:

$$\mathrm{A}_{123:c}\mathrm{A}_{1:c}(\mathit{act} \wedge \mathit{obl}) \tag{3}$$

$$\mathrm{A}_{123:c}\mathrm{A}_{2:c}(\mathit{act} \wedge \neg \mathit{obl}) \tag{4}$$

$$\mathrm{A}_{123:c}\mathrm{A}_{3:c}(\neg \mathit{act} \wedge \mathit{obl}) \tag{5}$$

Finally, the three judges use a majority principle for each proposition *act*, *obl* and *lia*. This majority principle is formally expressed by the following six hypotheses:

$$\begin{aligned}
\mathbf{Maj} = \{\ & \mathrm{A}_{123:c} \bigwedge_{i,j \in \{1,2,3\}, i \neq j} ((\mathrm{A}_{i:c}\mathit{act} \wedge \mathrm{A}_{j:c}\mathit{act}) \rightarrow \mathit{act}), \\
& \mathrm{A}_{123:c} \bigwedge_{i,j \in \{1,2,3\}, i \neq j} ((\mathrm{A}_{i:c}\neg\mathit{act} \wedge \mathrm{A}_{j:c}\neg\mathit{act}) \rightarrow \neg\mathit{act}), \\
& \mathrm{A}_{123:c} \bigwedge_{i,j \in \{1,2,3\}, i \neq j} ((\mathrm{A}_{i:c}\mathit{obl} \wedge \mathrm{A}_{j:c}\mathit{obl}) \rightarrow \mathit{obl}), \\
& \mathrm{A}_{123:c} \bigwedge_{i,j \in \{1,2,3\}, i \neq j} ((\mathrm{A}_{i:c}\neg\mathit{obl} \wedge \mathrm{A}_{j:c}\neg\mathit{obl}) \rightarrow \neg\mathit{obl}), \\
& \mathrm{A}_{123:c} \bigwedge_{i,j \in \{1,2,3\}, i \neq j} ((\mathrm{A}_{i:c}\mathit{lia} \wedge \mathrm{A}_{j:c}\mathit{lia}) \rightarrow \mathit{lia}), \\
& \mathrm{A}_{123:c} \bigwedge_{i,j \in \{1,2,3\}, i \neq j} ((\mathrm{A}_{i:c}\neg\mathit{lia} \wedge \mathrm{A}_{j:c}\neg\mathit{lia}) \rightarrow \neg\mathit{lia})\ \}
\end{aligned}$$

It is possible to prove that 1, 2, 3, 4 and 5 together with **Maj** lead to a contradiction using the axiomatisation of acceptance logic. Indeed, from hypotheses 3, 4 and 5 we infer

$$\mathrm{A}_{123:c}(\mathrm{A}_{1:c}\mathit{act} \wedge \mathrm{A}_{2:c}\mathit{act}) \wedge \mathrm{A}_{123:c}(\mathrm{A}_{1:c}\mathit{obl} \wedge \mathrm{A}_{3:c}\mathit{obl}).$$

By the first and third hypotheses in **Maj**, the latter implies

$$\mathrm{A}_{123:c}\mathit{act} \wedge \mathrm{A}_{123:c}\mathit{obl}$$

and by hypothesis 2 and standard modal principles the latter implies

$$\mathrm{A}_{123:c}\mathit{lia}.$$

From hypotheses 1 and 2, by axioms **4∗** and **5∗**, we can infer

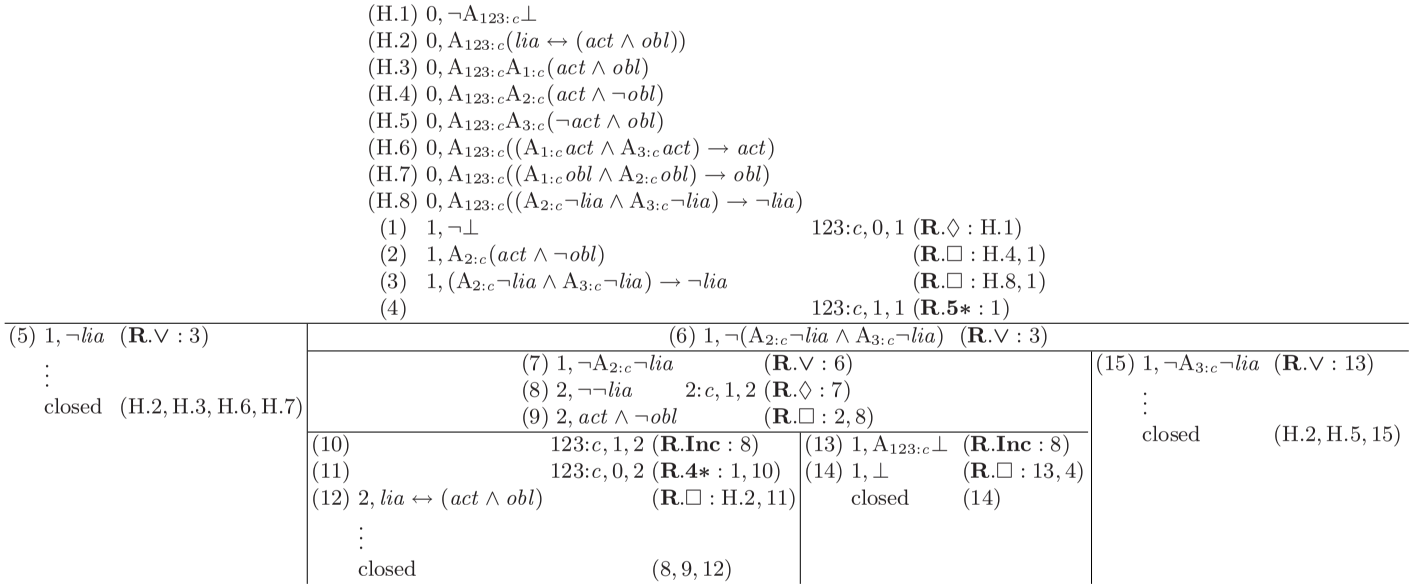

Fig. 5. A closed tableau for the discursive dilemma

$$\mathrm{A}_{123:c}(\mathrm{A}_{123:c}(lia \leftrightarrow (act \wedge obl)) \wedge \neg \mathrm{A}_{123:c}\bot).$$

By Axiom **Inc** and standard modal principles, the latter implies

$$\mathrm{A}_{123:c}(\mathrm{A}_{1:c}(lia \leftrightarrow (act \wedge obl)) \wedge \mathrm{A}_{2:c}(lia \leftrightarrow (act \wedge obl)) \wedge \mathrm{A}_{3:c}(lia \leftrightarrow (act \wedge obl))).$$

From this, by hypotheses 3, 4 and 5 and standard modal principles, we can infer

$$\mathrm{A}_{123:c}(\mathrm{A}_{1:c}lia \wedge \mathrm{A}_{2:c}\neg lia \wedge \mathrm{A}_{3:c}\neg lia)$$

and by the sixth hypothesis in **Maj** the latter implies

$$\mathrm{A}_{123:c}\neg lia.$$

Thus, we have $\mathrm{A}_{123:c}\bot$, and by hypothesis 1 we can infer $\bot$.

However, if one slightly changes the hypotheses, the proof may change completely. This means that the axiomatisation does not provide a straightforward way to automatise the process of finding such kind of inconsistency. On the other hand, a tableau method is meant to provide a systematic way to search for models and, thereby, an easy way to automatise the process of deciding whether a formula is inconsistent or not.

As an illustration, we show a closed tableau for this example (schematically) in Figure 5. On that figure, the lines identified by H.1–5 correspond to hypotheses 1–5 above, respectively. The line identified by H.6 corresponds to the first hypothesis in **Maj**, H.7 to the third hypothesis, and H.8 to the sixth. The branch containing lines 13 and 14 is already closed. The other branches are not completely displayed. But it is easy to see that all branches generated from line 12 on can be closed, since lines 8, 9 and 12 together are inconsistent in propositional logic. The branches generated from line 15 on can be closed in an analogous way as the latter. And the branches generated from line 5 on need more effort, but they can be closed too, in a similar way as the latter by using lines H.2, H.3, H.6 and H.7.

5 Conclusion

The contribution of this paper is a semantic tableau method for acceptance logic. The method consists in a procedure to check satisfiability of formulae, that can be easily automatised. Given that acceptance logic can be used to formalise some aspects of judgement aggregation, our method also provides an automated reasoning procedure for making judgement aggregation in modal logic.

It is to be noted that, differently from [21,22], in which logical approaches specialised for judgement aggregation have been proposed, in the logic of acceptance judgement aggregation is just an application. We have shown in [2,3] that the logic of acceptance is a much general formal framework in which the static and dynamic aspects of institutions can be studied (e.g., static and dynamic aspects of social roles, norms and rules).

Moreover, this method uses a rather different set of rules. For instance, rules for axioms **4∗** and **5∗** (which are nothing but a variation of the usual axioms

4 and 5) are similar to the so-called "structural rules" proposed in [23]. Our tableau rules are also modular. This means that it is possible to provide a sound, complete and terminating satisfiability checking method for a logic without one of the semantic constraints, by just removing the corresponding tableau rule. The addition of rules is also possible. For example, one could add the tableau rule:

$$(\mathbf{R}.\mathbf{Mon}) \quad \frac{;\langle G{:}x, n, n'\rangle}{;\langle H{:}x, n, n'\rangle}$$

which corresponds to Axiom **Mon**, thus, obtaining a tableau method for the acceptance logic with Axiom **Mon** proposed in [1,2].

As possible future works, we intend to investigate computational complexity, and also possible extensions of our method able to address acceptance logic with dynamic operators, such as the logic proposed in [24].

Acknowledgements

The contribution by Tiago de Lima is part of the research program Moral Responsibility in R&D Networks, supported by the Netherlands Organisation for Scientific Research (NWO), under grant number 360-20-160.

References

1. Gaudou, B., Longin, D., Lorini, E., Tummolini, L.: Anchoring institutions in agents' attitudes: Towards a logical framework for autonomous MAS. In: Padgham, L., Parkes, D.C. (eds.) Proceedings of AAMAS 2008., pp. 728–735 (2008)
2. Lorini, E., Longin, D., Gaudou, B., Herzig, A.: The logic of acceptance: Grounding institutions on agents' attitudes. Journal of Logic and Computation (2009), doi:10.1093/logcom/exn103
3. Lorini, E., Longin, D.: A logical account of institutions: From acceptances to norms via legislators. In: Brewka, G., Lang, J. (eds.) Proceedings of KR 2008, pp. 38–48 (2008)
4. Bratman, M.E.: Practical reasoning and acceptance in context. Mind 101(401), 1–15 (1992)
5. Cohen, L.J.: An essay on belief and acceptance. Oxford University Press, New York (1992)
6. Tuomela, R.: Belief versus acceptance. Philosophical Explorations 2, 122–137 (2000)
7. Tuomela, R.: The Philosophy of Social Practices: A Collective Acceptance View. Cambridge University Press, Cambridge (2002)
8. Boella, G., van der Torre, L.: Norm negotiation in multiagent systems. International Journal of Cooperative Information Systems 16(1), 97–122 (2007)
9. Hart, H.L.A.: The concept of law, new edn. Clarendon Press, Oxford (1992)
10. Gilbert, M.: On Social Facts. Routledge, London (1989)
11. Herzig, A., de Lima, T., Lorini, E.: On the dynamics of institutional agreements (manuscript, 2009)
12. Pettit, P.: Deliberative democracy and the discursive dilemma. Philosophical Issues 11, 268–299 (2001)

13. Kornhauser, L.A., Sager, L.G.: Unpacking the court. Yale Law Journal 96, 82–117 (1986)
14. Fitting, M.: Proof Methods for Modal and Intuitionistic Logics. Springer, Heidelberg (1983)
15. Halpern, J., Moses, Y.: A guide to completeness and complexity for modal logics of knowledge and belief. Artificial Intelligence 54, 311–379 (1992)
16. Goré, R.: Tableau methods for modal and temporal logics. In: D'Agostino, M., Gabbay, D.M., Hahnle, R., Posegga, J. (eds.) Handbook of Tableau Methods, pp. 297–396. Springer, Heidelberg (1999)
17. Pauly, M., van Hees, M.: Logical constraints on judgment aggregation. Journal of Philosophical Logic 35(6), 569–585 (2006)
18. List, C., Pettit, P.: Aggregating sets of judgments: An impossibility result. Economics and Philosophy 18, 89–110 (2002)
19. Goldman, A.: Group knowledge versus group rationality: Two approaches to social epistemology. Episteme 1(1), 11–22 (2004)
20. List, C.: Group knowledge and group rationality: A judgment aggregation perspective. Episteme 2(1), 25–38 (2005)
21. Ågotnes, T., van der Hoek, W., Wooldridge, M.: Reasoning about judgment and preference aggregation. In: Proceedings of AAMAS 2007, pp. 566–573 (2007)
22. Pauly, M.: Axiomatizing collective judgment sets in a minimal logical language. Synthese 158, 233–250 (2007)
23. Castilho, M.A., Fariñas del Cerro, L., Gasquet, O., Herzig, A.: Modal tableaux with propagation rules and structural rules. Fundamenta Informaticae 20, 1–17 (1998)
24. Herzig, A., de Lima, T., Lorini, E.: What do we accept after an announcement? In: Meyer, J.-J.C., Broersen, J. (eds.) Pre-proceedings of the KR'08-Workshop KRAMAS, pp. 81–94 (2008), http://www.cs.uu.nl/events/kramas2008/PreProceedingsKRAMAS2008.pdf

Ontology and Time Evolution of Obligations and Prohibitions Using Semantic Web Technology

Nicoletta Fornara[1] and Marco Colombetti[1,2]

[1] Università della Svizzera italiana, via G. Buffi 13, 6900 Lugano, Switzerland
{nicoletta.fornara,marco.colombetti}@usi.ch
[2] Politecnico di Milano, piazza Leonardo Da Vinci 32, Milano, Italy
marco.colombetti@polimi.it

Abstract. The specification and monitoring of conditional obligations and prohibitions with starting points and deadlines is a crucial aspect in the design of open interaction systems. In this paper we regard such obligations and prohibitions as cases of social commitment, and propose to model them in OWL, the logical language recommended by the W3C for Semantic Web applications. In particular we propose an application-independent ontology of the notions of social commitment, temporal proposition, event, agent, role and norms that can be used in the specification of any open interaction system. We then delineate a hybrid solution that uses the OWL ontology, SWRL rules, and a Java program to dynamically monitor or simulate the temporal evolution of social commitments, due to the elapsing of time and to the actions performed by the agents interacting within the system.

1 Introduction

The specification of *open interaction systems*, where heterogeneous, autonomous, and self-interested agents can interact by entering and leaving dynamically the system, is widely recognized to be a crucial issue in the development of distributed applications on the Internet, like e-commerce applications, or collaborative applications for the automatic creation of virtual organizations. An important aspect of the specification of open systems is the possibility to define the actions that agents should or should not perform in a given interval of time, that is, the possibility to define social commitments with starting points and deadlines, and to monitor and react to their fulfilment or violation.

As we discussed in our previous works [11,12,10] in our OCeAN meta-model for the specification of artificial institutions, commitments for the interacting agents can be created by the activation of norms associated to the agents' roles, or by the performance of agent communicative acts, like promises. In this paper we explore how to use OWL (in its OWL 2 DL version[1]), the logical language recommended by W3C for Semantic Web applications, to specify the deontic part of the *OCeAN* meta-model. More precisely, we show how it is possible to

[1] http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 101–118, 2010.

specify social commitment to express conditioned obligations and prohibitions on time intervals, in OWL.

There are many advantages of using a decidable logical language like OWL to specify an open interaction system, and in particular that: (i) Semantic Web technologies are increasingly becoming a standard for Internet applications; (ii) the language is supported by reasoners (like Fact++[2], Pellet[3], and the rule reasoner of the Jena Semantic Web framework[4]) that are more efficient than available alternatives (like the Discrete Event Calculus Reasoner[5]); (iii) it is possible to achieve a high degree of interoperability of data and applications, which is indeed a crucial precondition for the development of open systems.

The idea of using OWL for modelling and monitoring the dynamic evolution of open artificial institutions can be developed following different approaches. A first option would be to implement an institutional model in a object oriented language like Java, and use OWL only to specify the ontology of the content of communicative acts and norms. As a result reasoning may be used to deduce, for example, that the performance of a certain act implies the performance of another act, and thus the fulfillment of a given commitment. An alternative approach, which we investigate in this paper, consists in using OWL to express, as far as possible, the normative component of the OCeAN meta-model. As we shall see, this requires the use of SWRL (Semantic Web Rule Language[6]) and Java code to overcome certain expressiveness limitations of OWL. Indeed, with both OWL 1 (the current standard) and OWL 2 there are at least two major problems:

- The treatment of time. OWL has no temporal operators; on some occasions it is possible to bypass the problem by using SWRL rules and built-ins for comparisons, but in any case this does not provide full temporal reasoning capabilities. Another possible solution would consist in using the OWL Time Ontology[7], but given that its axiomatization is very weak, this alternative presents limitations analogous to those previously discussed.
- The open-world assumption. In many applications, nor being able to infer that an action has been performed is sufficient evidence that the action has not been performed; one would then like to infer, for example, that an obligation to perform the action has been violated. As standard OWL reasoning is carried out under the open world assumption, inferences of this type cannot be drawn. However, it is often possible to simulate a closed world assumption by adding closure axioms to an ontology.

The main contribution of this paper, with respect to our previous works, is to show how obligations and prohibitions can be formalized in OWL and SWRL for

[2] http://owl.man.ac.uk/factplusplus/
[3] http://clarkparsia.com/pellet
[4] http://jena.sourceforge.net/inference/
[5] http://decreasoner.sourceforge.net
[6] http://www.w3.org/Submission/SWRL/
[7] http://www.w3.org/TR/owl-time/, http://www.w3.org/2006/time.rdf

monitoring and simulation purposes with significant performance improvements with respect to the solution based on the Event Calculus that we presented elsewhere [10]. Another contribution of this work is a hybrid solution of the problem of monitoring the temporal evolution of obligations and prohibition, based on application independent upper ontology of those concepts, extended including other ontologies specific for the domain of application, a set of SWRL rules, and a Java program implemented using suitable OWL libraries (like the Jena Semantic Web Framework for Java[8] or OWL API[9]).

The paper is organized as follows. In the next section we briefly introduce OWL and SWRL, that is, the Semantic Web languages that we use to formally specify the normative component of an open interaction system. In Section 3 we specify the algorithms that we plan to use to simulate or monitor the temporal evolution of an interaction system. Then in Section 4 we define the classes, properties, axioms, and rules that we take to underlie the normative specification of every interaction system. In Section 5 we present an actual example of a system specified using the proposed ontology. Finally in Section 6 we compare our approach with other proposals and draw some conclusions.

2 OWL and SWRL

OWL is a practical realization of a Description Logic system known as $\mathcal{SROIQ}(\mathcal{D})$. It allows one to define *classes* (also called *concepts* in the DL literature), *properties* (also called *roles*), and *individuals*. An OWL ontology consists of: a set of class axioms to describe classes, which constitute the *Terminological Box* (*TBox*); a set of property axioms to describe properties, which constitute a *Role Box* (*RBox*); and a collection of assertions to describe individuals, which constitute an *Assertion Box* (*ABox*).

Classes can be viewed as formal descriptions of sets of objects (taken from a nonempty universe), and individuals can be viewed as names of objects of the universe. Properties can be either *object properties* or *data properties*. The former describe binary relations between objects of the universe; the latter, binary relationships between objects and data values (taken from XML Schema datatypes).

A class is either a *basic class* (i.e., an atomic class name) or a *complex class* build through a number of available *constructors* that express Boolean operations and different types of restrictions on the members of the class.

Through *class axioms* one may specify subclass or equivalence relationships between classes, and that certain classes are disjoint. *Property axioms* allow one to specify that a given property is a subproperty of another property, that a property is the inverse of another property, or that a property is functional or transitive. Finally, *assertions* allow one to specify that an individual belongs to a class, that an individual is related to another individual through an object

[8] http://jena.sourceforge.net/
[9] http://owlapi.sourceforge.net/

property, that an individual is related to a data value through a data property, or that two individuals are equal or different.

OWL can be regarded as a decidable fragment of First Order Logic (FOL). The price one pays for decidability, which is considered as an essential preconditions for exploiting reasoning in practical applications, is limited expressiveness. Even in OWL 2 (the more expressive version currently under specification) certain useful first-order statements cannot be formalized.

Recently certain OWL reasoners, like Pellet, have been extended to deal with SWRL rules. SWRL is a Datalog-like language, in which certain universally quantified conditional axioms (called *rules*) can be stated. To preserve decidability, however, rules have to be used in the *safe mode*, which means that before being exploited in a reasoning process all their variables must be instantiated by pre-existing individuals. An important aspect of SWRL is the possibility of including *built-ins*, that is, Boolean functions that perform operations on data values and return a truth value.

Conventions

In what follows we use the notation $p : C \rightarrow_O D$ to specify an *object property* p (not necessarily a function) with class C as domain and class D as range, and the notation $q : C \rightarrow_D T$ to specify a *data property* q with class C as domain and the datatype T as range. We use capital initials for classes, and lower case initials for properties and individuals.

3 Specification and Simulation or Monitoring of an Open Interaction System

Our approach is to model an open interaction system using one or more artificial institutions. The definition of a specific artificial institution consists of: (i) a first component, called meta-model, which includes the definition of basic entities common to the specification of every institution, like the concepts of temporal proposition, commitment, institutional power, role, and norm, and the actions necessary for exchanging messages; (ii) a second component, pertaining to the institution in exam, which includes the definition of specific powers and norms that apply to the agents playing roles in the institution, and the definition of the concepts pertaining to the domain of the interaction (for example the actions of paying or delivering a product, bidding in an auction, etc.).

We start from the specification of a system, formalized as an application-independent OWL ontology (including a TBox, an RBox, and an ABox as detailed in Section 4). We then add an application-dependent ontology (as exemplified in Section 5) and use a Java program to let such ABox evolve in time on the basis of the events, with the goal of *monitoring* the fulfilment or violation of obligations and prohibitions. Those events could be actual events that happen during the run time of the system mainly due to the interaction of the agents or events registered in a possible history of the system for simulation purposes.

In particular, when the system is used for run time monitoring, a Java program updates the state of the system, that is, it updates the ABox with new assertions

to model the elapsing of time, to allow for closed-world reasoning on certain classes, and to model the actions performed by the interacting agents. When such updating is completed, a reasoner can be used to deduce the state of obligations and prohibitions. After that, when the ontology has reached a stable state (in the sense that all closed-world reasoning has been completed), the agents may perform queries to know what are their pending obligations or prohibitions or to react to their violation or fulfillment. We assume that the events or actions that happen between two phases of update (that is, between two discrete instant of time) are queued in the data structure `ActionQueue` for being managed by the Java program subsequently.

When the system is used for simulation, the set of events that happen at run-time are known since the beginning, and are represented in the initial version of the ABox. In such a case the Java program simply updates the state of the system to represent the elapsing of time and to allow closed-world reasoning on certain classes; then the reasoner deduces the state of obligations and prohibitions at each time instant.

Temporal evolution of the ontology

An external Java program is used to update the ABox to model the elapsing of time, the actions performed by the interacting agents at run-time (in the monitoring usage), and to allow for closed-world reasoning on certain classes (see Section 4.1 for details). It is important to underline that the external program is used to update the ABox of the ontology and only to add new knowledge acquired due to time elapsing, it is never used to remove knowledge.

The program performs the following operations:

1. initialize the simulation/monitoring time t equal to 0 and close the extensions of the classes C, on which it is necessary to perform closed-world reasoning, by asserting that the class KC is equivalent to the enumeration of all individuals that can be proved to be members of the class C retrieved with the `retrieve(C)` query command;
2. insert in the ABox the assertion $happensAt(elapse, t)$;
3. insert in the ABox the events or actions that happen in the system between $t-1$ and t and that are cached in the `ActionQueue` queue (this involves creating new individuals of the class *Event*);
4. run a reasoner (more specifically, Pellet 2.0) to deduce all assertions that can be inferred (truth values of temporal propositions, states of commitments, etc.);
5. update the closure of the relevant classes C;
6. increment the time of simulation t by 1 and go to the point 2.

After point 5, given that the ontology has reached a stable state it is possible to let agents perform queries about pending, fulfilled, or violated commitments in order to plan their subsequent actions and to apply sanctions or rewards. When the ontology is used for monitoring purposes, and given that internal time (i.e., the time as represented in the ontology) is discrete, it is necessary to wait the actual number of seconds that elapse between two internal instants.

The corresponding Java pseudo code is as follows:

```
t=0
for each class C that has to be closed
   assert KC ≡ {i_1, ...i_n} with {i_1, ...i_n} = retrieve(C)
while t<timeSimulation {
   assert happensAt(elapse,t)
   for each event  e_n  in ActionQueue
      assert happensAt(e_n,t)
   run Pellet reasoner
   for each class C that has to be closed
      remove equivalent class axioms of class KC
      assert KC ≡ {i_1, ...i_n} with {i_1, ...i_n} = retrieve(C)
   run agents queries
   t=t+1
}
```

4 The Ontology of Obligations and Prohibitions

In this section we present the TBox, the RBox, and part of the Abox that have to be included in the ontology of any interaction system modelled using the OCeAN concepts of temporal proposition, commitment, role, and norm. In particular we specify the classes, the properties and the axioms for modelling those concepts and introduce some SWRL rules to deduce the truth value of temporal propositions. Social commitments are a crucial concept in our approach because they are used to model obligations and prohibitions due either to the activation of norms or created by the performance of communicative acts, like promises. Thanks to their evolution in time, commitments can be used to monitor the behavior of autonomous agents by detecting their *violation* or *fulfilment*, as a precondition for reacting with suitable *passive or active sanctions* or with a *reward* [10].

Some general classes of our ontology are used as domain or range of the properties used to describe temporal propositions and commitments; they are class *Event*, class *Action* and class *Agent*. In particular, an event may have as a property its *time* of occurrence. Class *Action* is a subclass of *Event*, and has a further property used to represent the *actor* of the action. Such properties are defined as follows:

$Event \sqcap Agent \sqsubseteq \bot$; $Action \sqsubseteq Event$;
$hasActor : Action \rightarrow_O Agent$;
$happensAt : Event \rightarrow_D integer$;

To represent the elapsing of time we introduce in the ABox the individual *elapse*, that is asserted to be a member of class *Event*: *Event(elapse)*.

4.1 Temporal Propositions

Temporal propositions are used to represent the *content* and *condition* of social commitments. They are a construct used to relate in two different ways a *proposition* to an *interval of time*. In the current OWL specification, we distinguish between *positive temporal propositions* used in commitments to represent

obligations (when an action has to be performed within a given interval of time), and *negative temporal propositions* used to model prohibitions (when an action must not be performed during a predefined interval of time).

The classes necessary to model temporal propositions are *TemporalProp*, with the two subclasses *TPPos* and *TPNeg* used to distinguish between positive and negative temporal propositions. The classes *IsTrue* and *IsFalse* are used to model the truth values of temporal propositions. All this is specified by the following axioms:

$TemporalProp \sqcap Agent \sqsubseteq \bot$; $TemporalProp \sqcap Event \sqsubseteq \bot$;
$TPPos \sqsubseteq TemporalProp$; $TPNeg \sqsubseteq TemporalProp$;
$TPPos \sqcap TPNeg \sqsubseteq \bot$;
$TemporalProp \equiv TPPos \sqcup TPNeg$;
$IsTrue \sqsubseteq TemporalProp$; $IsFalse \sqsubseteq TemporalProp$;
$IsTrue \sqcap IsFalse \sqsubseteq \bot$;

The class *TemporalProp* is the domain of the following object and data properties further specified with a cardinality axiom:

$hasAction : TemporalProp \rightarrow_O Action$;
$hasStart : TemporalProp \rightarrow_D integer$;
$hasEnd : TemporalProp \rightarrow_D integer$;
$TemporalProp \sqsubseteq = 1\ hasAction \sqcap\ = 1\ hasStart \sqcap\ = 1\ hasEnd$

The classes $IsTrue$ and $IsFalse$ are used to keep track of the truth value of temporal propositions by means of two SWRL rules, that are different on the basis of the type of temporal proposition. A positive temporal proposition (i.e., a member of class *TPPos*) is used to represent an obligation to do something in a given interval of time, with starting points t_{start} and deadline t_{end}. We therefore introduce a rule that deduces that the truth value of the temporal proposition is true (i.e., the temporal proposition becomes member of the class $IsTrue$) if the action associated to the temporal proposition is performed between the t_{start} (inclusive) and the t_{end} (exclusive) of interval of time associated to the same proposition. In the following SWRL rule we use two built-ins to compare the current time with the interval of time associated to the temporal proposition:

RuleTPPos1
happensAt(elapse,?t) $\wedge$ *happensAt(?a,?t)* $\wedge$ $TPPos$*(?tp)* $\wedge$ *hasAction(?tp,?a)* $\wedge$ *hasStart(?tp,?ts)* $\wedge$ *hasEnd(?tp,?te)* $\wedge$ *swrlb:lessThanOrEqual(?ts,?t)* $\wedge$ *swrlb:lessThan(?t,?te)* $\rightarrow$ *IsTrue(?tp)*

We then have to define a rule that, when the time t_{end} of a positive temporal proposition elapses, and such a temporal proposition is not true, deduces that the temporal proposition is member of the class $IsFalse$. Here closed-world reasoning comes into play, because we cannot assume the ABox to contain an explicit assertion that an action has not been performed: rather, we want to deduce that an action has not been performed by the lack of an assertion that it has been performed. Clearly, an SWRL rule like

happensAt(elapse,?te) $\wedge$ *hasEnd(?tp,?te)* $\wedge$ *TPPos(?tp)* $\wedge$(*not* $IsTrue$)(*?tp*)
$\rightarrow$ *IsFalse(?tp)*

would not work, given that OWL/SWRL reasoners operate under the open world assumption. This means that the conclusion that a temporal proposition is false can only be reached for those propositions that can be definitely proved not to be members of $IsTrue$. On the contrary, if a temporal proposition is not deduced to be $IsTrue$ by RuleTp1, even if its deadline has been reached it will not be deduced to be $IsFalse$.

To solve this problem we first assume that our ABox contains complete information on the actions performed before the current time of the system. This allows us to adopt a closed-world perspective as far as the performance of actions is concerned. More specifically, we assume that the program specified in Section 3 will always update the ABox when an event has happened (i.e. the program can only inset in the ABox the information that an event has happened at current time t); we then want to deduce that all temporal propositions, that cannot any longer become true because their deadline has elapsed, are false.

To get this result we need to perform some form of closed world reasoning on class $IsTrue$. As stated in [17] "the DL $\mathcal{ALCK}$ [7] adds a non-monotonic **K** operator (which is a kind of necessity operator) to the DL $\mathcal{ALC}$ to provide the ability to "turn on" the Closed World Assumption (CWA) when needed. The reasoning support for $\mathcal{ALCK}$ language has been implemented in Pellet to answer CWA queries that use the **K** operator". However, our ontology uses a more expressive DL than $\mathcal{ALC}$; moreover, the use of the **K** operator in SWRL rules is not supported.

We therefore take a different approach, based on an explicit *closure* of class $IsTrue$. More precisely, we introduce a new class, $KIsTrue$, which is meant to contain all temporal propositions that, at a given time, are known to be true. Class $KIsTrue$ therefore represents, at any given instant, the explicit closure of class $IsTrue$. Given its intended meaning, class $KIsTrue$ has to be a subclass of $IsTrue$ (and, as a consequence, of $TemporalProp$):

$$KIsTrue \sqsubseteq IsTrue$$

To maintain class $KIsTrue$ as the closure of class $IsTrue$, we define it periodically as equivalent to the enumeration of all individuals that can be proved to be members of $IsTrue$. This can be done by the Java program used to update the ABox to keep track of the elapsing of time (described in Section 3) by executing the operations described in the following pseudo-code:

$$\texttt{assert KIsTrue} \equiv \{tp_1, ...tp_n\} \texttt{ with } \{tp_1, ...tp_n\} = \texttt{retrieve(IsTrue)}$$

We now introduce a new class, $NotKIsTrue$, which is intended to contain all temporal propositions whose deadline is elapsed, and that are not members of $KIsTrue$. Such a class is defined as the difference between the set of all individuals that belong to $TemporalProp$, and the set of all those individuals that are members of $KIsTrue$:

$$NotKIsTrue \equiv TemporalProp \sqcap \neg KIsTrue$$

We are now ready to write a rule to deduce that the truth value of a positive temporal proposition is false if the deadline of the temporal proposition has elapsed, and it is not known that the associated action has been performed:

RuleTPPos2
happensAt(elapse,?te) $\wedge$ *hasEnd(?tp,?te)* $\wedge$ *TPPos(?tp)* $\wedge$ $NotKIsTrue(?tp)$ $\rightarrow$ *IsFalse(?tp)*

We now turn to negative temporal propositions, that is, temporal propositions that are members of the class *TPNeg* and are used to represent the prohibition to do something in a given interval of time. Such propositions belong to class $IsFalse$ when the associated action is performed in the interval between t_{start} (inclusive) and t_{end} (exclusive). This can be deduced by the following rule:

RuleTPNeg1
happensAt(elapse,?t) $\wedge$ *happensAt(?a,?t)* $\wedge$ *TPNeg(?tp)* $\wedge$ *hasAction(?tp,?a)* $\wedge$ *hasStart(?tp,?ts)* $\wedge$ *hasEnd(?tp,?te)* $\wedge$ *swrlb:lessThanOrEqual(?ts,?t)* $\wedge$ *swrlb:lessThan(?t,?te)* $\rightarrow$ *IsFalse(?tp)*

Similarly to what we did for RuleTPPos2, we now use the closure of class $IsFalse$, that we call $KIsFalse$, to deduce that a negative temporal proposition $IsTrue$ when its t_{end} has been reached and it has not yet been deduced that the proposition $IsFalse$:

$KIsFalse \sqsubseteq IsFalse$
$NotKIsFalse \equiv TemporalProp \sqcap \neg KIsFalse$

RuleTPNeg2
happensAt(elapse, ?te) $\wedge$ *hasEnd(?tp,?te)* $\wedge$ *TPNeg(?tp)* $\wedge$ $NotKIsFalse(?tp)$ $\rightarrow$ *IsTrue(?tp)*

4.2 Commitment

In the *OCeAN* meta-model of artificial institutions, commitments are used to model a social relation between a *debtor* a *creditor*, about a certain *content* and under a *condition*. Our idea is that by means of the performance of communicative acts, or due to the activation of norms, certain agents become committed with respect to another agent to perform a certain action within a given deadline (an *obligation*), or not to perform a given action during a given interval of time (a *prohibition*). Such commitments can be conditional on the truth of some proposition. In our model we assume that if an action is neither obligatory nor prohibited, then it is *permitted*.

In order to detect and react to commitment violation and fulfilment we need to deduce a commitments *state* (in our previous works [10] we also introduced precommitments to define the semantics of requests, but this is not relevant in the current work). We introduce in the ontology the class *Commitment*, disjoint from *Event*, *Agent* and *TemporalProp*.

$Commitment \sqcap Agent \sqsubseteq \bot$; $Commitment \sqcap Event \sqsubseteq \bot$;
$Commitment \sqcap TemporalProp \sqsubseteq \bot$;

The *Commitment* class is the domain of the following object properties:

$hasDebtor : Commitment \rightarrow_O Agent$;
$hasCreditor : Commitment \rightarrow_O Agent$;
$hasContent : Commitment \rightarrow_O TemporalProp$;
$hasCondition : Commitment \rightarrow_O TemporalProp$;
$hasSource : Commitment \rightarrow_O Norm$;
$Commitment \sqsubseteq \exists hasDebtor \sqcap \exists hasCreditor \sqcap =1 hasContent \sqcap =1 hasCondition$;

The *hasSource* property is used to keep track of the norm that generated a commitment, as explained in Section 4.3. Obviously the debtor of a commitment has to be the actor of the action to which it is committed, as expressed by the following axiom:

$$hasContent \circ hasAction \circ hasActor \sqsubseteq hasDebtor$$

In some situations it is necessary to create unconditional commitments. To avoid writing different rules for conditional and for unconditional commitments, we introduce a temporal proposition individual, $tpTrue$, whose truth value is initially true; that is, we assert: $IsTrue(tpTrue)$. An unconditional commitment is then defined as a conditional commitment whose condition is $tpTrue$.

Our next problem is deducing whether a given commitment is:

- *pending*, when its condition is satisfied but its content is not known to be $IsTrue$ or to be $IsFalse$;
- $fulfilled$, when is content is known to be $IsTrue$;
- *violated*, when its content is known to be $IsFalse$ and its condition is known to be $IsTrue$.

Knowing the state of a commitment may be important for the interacting agents to plan their actions on the basis of the advantages of fulfilling certain commitments. We therefore introduce classes $IsPending$, $IsFulfilled$, and $IsViolated$, defined by the following axioms:

$IsFulfilled \sqcap IsViolated \sqsubseteq \bot$;
$IsPending \sqsubseteq Commitment$; $IsFulfilled \sqsubseteq Commitment$;
$IsViolated \sqsubseteq Commitment$;

We define the following axiom to deduce that a commitment is member of the class $IsPending$:

Axiom1
$IsPending \equiv (\exists\, hasContent.NotKIsTrue) \sqcap (\exists\, hasContent.NotKIsFalse) \sqcap (\exists\, hasCondition.IsTrue))$

Note that as classes $NotKIsTrue$ and $NotKIsFalse$ are updated after running the reasoner, as soon as the *content* of a commitment becomes true the commitment is member of both class $IsPending$ and class $IsFulfilled$.

Lists of fulfilled and of violated commitments can be obtained by retrieving the individuals that are respectively members of class $IsFulfilled$ or $IsViolated$, defined by the following axioms:

Axiom2
$IsFulfilled \equiv \exists\, hasContent.IsTrue$

Axiom3
$IsViolated \equiv (\exists\, hasContent.IsFalse) \sqcap (\exists\, hasCondition.IsTrue)$

4.3 Norms and Roles

In OCeAN, norms are introduced to model obligations and prohibitions that, contrary to those created at run time by the performance of communicative acts, are implied by an institutional setting and can be specified at design time. For example, norms can be used to state the rules of an interaction protocol, like the protocol of a specific type of auction, or the rules of a seller-buyer interaction. Given that norms are usually specified at design time, when it is impossible to know which agents will actually interact in the system, one of their distinctive features is that they have to be expressed in term of the *roles* played by the agents. Therefore at run-time, when a norm becomes *active* (i.e., when its activating event happens), the actual debtor and creditor of the obligation or prohibition generated by the norm have to be computed on the basis of the roles played by the agents in the system at that moment.

Another important aspect of norms is that to enforce their fulfillment in an open system, it must be possible to specify *sanctions* or *rewards*. In [9] we suggested that a satisfactory model of sanctions has to distinguish between two different type of actions: the action that the violator of a norm has to perform to extinguish its violation (which we call *active sanction*), and the action that the agent in charge of norm enforcement may perform to deter agents from violating the norm (which we call *passive sanction*). Active sanctions can be represented in our model through a temporal proposition, whereas passive sanctions can be represented as new specific powers that the agent entitled to enforce the norm acquires when a norm is violated. As far as passive sanctions are concerned, another norm (that in [16]) is called *enforcement norm*) may oblige the enforcer to punish the violation. Due to space limitations, in this paper we do not model the notion of *power*; thus passive sanctions are not treated in this paper. An obligation or prohibition generated by a norm can in turn violated; it will therefore be necessary to monitor the fulfillment or violation of such obligations or prohibition t punish the violation.

Role

Typically, artificial institutions provide for different roles. In a run of an auction, for example, we may have the roles of auctioneer and of participant; in a company, like an auction house, we may have the roles of boss or employee; and so on. More generally, also the debtor and the creditor of a commitment may be regarded as roles. Coherently with these examples, a role is identified by a label (like *auctioneer*, *participant*, etc.) and by the institutional entity that provides for the role. Such an institutional entity may be an organization (like an auction house), an institutional activity (like a run of an auction), or an institutional

relationship (like a commitment). For example an agent may be the *auctioneer* of run 01 of a given auction, or an *employee* of IBM, or the *creditor* of a specific commitment.

We introduce class *Role* to represent the set of possible labels that representing roles and class *InstEntity* to represent the institutional entity within which a given role is played. Elements of class *AgentInRole* are used to reify the fact that an agent plays a given role in a given institutional entity. Those classes are related by the following object properties:

$isPlayedBy : AgentInRole \rightarrow_O Agent$;
$hasRole : AgentInRole \rightarrow_O Role$;
$isIn : AgentInRole \rightarrow_O InstEntity$;

Norm

Summarizing, a norm has: a *content* and a *condition*, modelled using temporal propositions; a *debtor* and a *creditor*, expressed in term of roles; an *activating event*; and a collection of *active* and *passive sanctions*. Norms are represented in our ontology using class *Norm* and the following object properties:

$hasRoleDebtor : Norm \rightarrow_O Role$; $hasRoleCreditor : Norm \rightarrow_O Role$;
$hasNContent : Norm \rightarrow_O TemporalProp$;
$hasNCondition : Norm \rightarrow_O TemporalProp$;
$hasActivation : Norm \rightarrow_O Event$;
$hasASanction : Norm \rightarrow_O TemporalProp$;
$hasPSanction : Norm \rightarrow_O Power$;

When a norm is activated it is necessary to create as many commitments as there are agents playing the role associated to the debtor property of the norm. For example, the activation of a norm that applies to all the agents playing the role of *participant* of an auction, creates a commitment for each participant currently taking part to the auction. The creditors of these commitments are the agents that play the role reported in the creditor property of the norm. All these commitments have to be related by the *hasSource* object property (defined in Section 4.2) to the norm that generated them; this is important to know which norm generated a commitment and what sanctions apply for the violation of such commitment.

As every commitment is an individual of the ontology, the activation of a norm involves the generation of new individuals. However, the creation of new individuals in an *ABox* cannot be performed using OWL or SWRL. There are at least two possible solutions to this problem, which we plan to investigate in our future work. The first consists in defining a set of axioms in the ontology that allows the reasoner to deduce the existence of those commitments as anonymous objects with certain properties. With this solution, an agent that needs to know its pending commitments instead of simply retrieving the corresponding individuals will have to retrieve their contents, conditions and debtors. Another possible solution consists in defining a new built-in that makes it possible for SWRL rules to create new individuals as members of certain classes and with given properties. A similar problem will have to be solved to manage the creation of a sanctioning

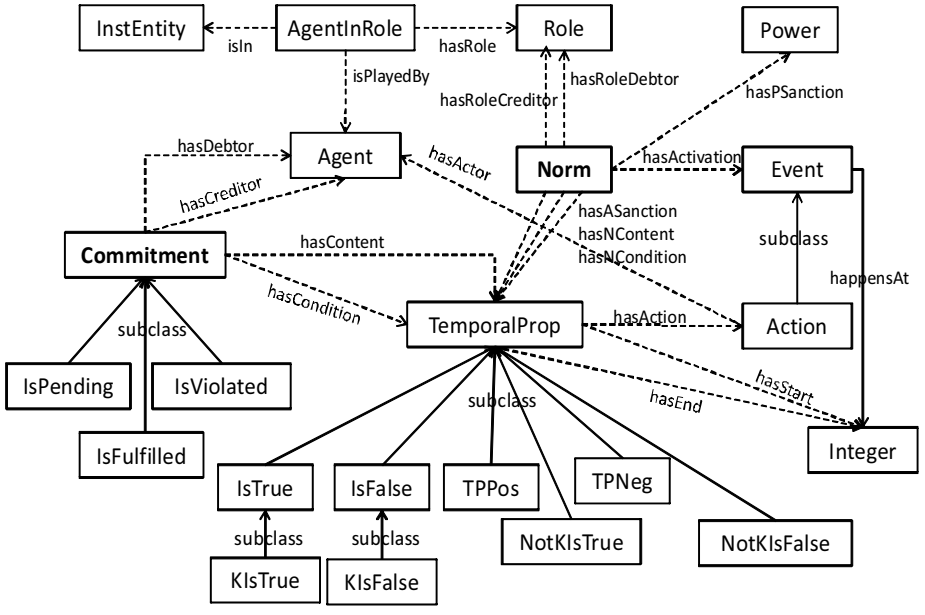

Fig. 1. Graphical representation of the ontology

commitment generated by the violation of a commitment related to a norm, which has as content the temporal proposition associated to the active sanction of the norm.

In Figure 1 classes, subclasses, and properties (dotted lines) of the ontology described in this section are graphically represented.

5 Example

In this section we show how it is possible to specify the state of an interaction system and to simulate or monitor its evolution in time. To do so it is necessary to integrate the ontology defined in the previous sections with an application-dependent ontology, and to insert a set of individuals for representing commitments and temporal propositions in the ABox. In a real application these commitments and their temporal propositions will be created by the performance of communicative acts (defined in the *OCeAN* agent communication library [10]) or by the activation of norms. If the system is used for monitoring purposes, we assume that there is a way of mapping the actions that are actually executed onto their counterparts in the ontology.

Here we describe an example of interaction where a buyer agent, Ann, promises to pay a certain amount of money for a product (a book) to a seller agent, Bob, on condition that the seller agent delivers the product to Ann. We also represent the

prohibition for the seller to deliver a different product (a CD). Different possible evolution of the state of the interaction are possible on the basis of the agents' actions.

The ontology described in the previous sections has to be integrated with the action for paying an amount of money and for delivering a product. In a realistic application they would be described in a detailed domain-dependent ontology introducing the class *Pay* and the class *Delivery* as subclass of the class *Action* described in our application independent ontology. Both of those type of action have a *receiver* and an *object*, the pay action type also has an *amount of money*. For simplicity in this example we represent the action of payment of the book and its delivery with individuals inserted in the ABox.

The agents are represented with the following assertions:

$Agent(ann); Agent(bob); \neq (ann, bob);$

The actions that we are interested to model in the ontology are represented by the following assertions:

$Action(payBook1);\ Action(deliverBook1);\ Action(deliverCD1);$
$hasActor(payBook1, ann);\ hasActor(deliverBook1, bob);$
$hasActor(deliverCD1, bob);$
$\neq (payBook1, deliverBook1, deliverCD1, elapse);$

Temporal propositions are represented by the following assertions:

$TPPos(tpPayBook1); TPPos(tpDeliverBook1); TPNeg(tpNotDeliverCD1);$
$hasAction(tpPayBook1, payBook1);\ hasStart(tpPayBook1, 1);$
$hasEnd(tpPayBook1, 3);$
$hasAction(tpDeliverBook1, deliverBook1);\ hasStart(tpDeliverBook1, 1);$
$hasEnd(tpDeliverBook1, 2);$
$hasAction(tpNotDeliverCD1, deliverCD1); hasStart(tpNotDeliverCD1, 0);$
$hasEnd(tpNotDeliverCD1, 3);$
$\neq (tpPayBook1, tpDeliverBook1, tpTrue);\ \neq (tpNotDeliverCD1, tpTrue);$

Commitments are represented by the following assertions:

$Commitment(c1);\ Commitment(c2);\ Commitment(c3);$
$hasDebtor(c1, ann);\ hasCreditor(c1, bob);$
$hasContent(c1, tpPayBook1);\ hasCondition(c1, tpDeliverBook1);$
$hasDebtor(c2, bob);\ hasCreditor(c2, ann);$
$hasContent(c2, tpDeliverBook1);\ hasCondition(c2, tpTrue);$
$hasDebtor(c3, bob);\ hasCreditor(c3, ann);$
$hasContent(c3, tpNotDeliverCD1);\ hasCondition(c3, tpTrue);$
$\neq (c1, c2, c3);$

The history of the system is represented by the following assertions (the action happens at time 1):

$happensAt(deliverBook1, 1)$

We created the ontology of the interaction system with the free, open source ontology editor Protege 4.0 beta[10]. As this version of Protege does not support the

[10] http://protege.stanford.edu/

editing of SWRL rules, we created them with Protege 3.4 and inserted their RDF/XML code in the ontology file. We implemented the Java program described in Section 3 using OWL-API library to operate on the ontology and the source code of Pellet 2.0 to reason and query it[11].

In Table 1 we report the evolution of the ontology ABox in time, with particular regard to the truth value of the temporal propositions and the state of commitments. As the extension of classes $KIsTrue$ and $KIsFalse$ are computed by an external program, when the reasoner runs their extensions are specified in the axiom relative to the previous state. In the table we abbreviate the assertion $happensAt(elapse, n)$ with the expression $t = n$.

Table 1. Dynamic evolution of the state of the system

$time$	$t=0$	$t=1$	$t=2$	$t=3$
$tpPayBook1[1,3]$				$IsFalse$
$tpDeliverBook1[1,2]$		$IsTrue$	$IsTrue$	$IsTrue$
$tpNotDeliverCD1[0,3]$				$IsTrue$
$c1(ann, bob,$ $tpPayBook1,$ $tpDeliverBook1)$		$IsPending$	$IsPending$	$IsViolated$
$c2(bob, ann,$ $tpDeliverBook1,$ $tpTrue)$	$IsPending$	$IsFulfilled$	$IsFulfilled$	$IsFulfilled$
$c3(bob, ann,$ $tpNotDeliverCD1,$ $tpTrue),$	$IsPending$	$IsPending$	$IsPending$	$IsFulfilled$

Classes updated by the external program

$KIsTrue$	$\{tpTrue\}$	$\{tpTrue,$ $tpDeliverBook1\}$	$\{tpTrue,$ $tpDeliverBook1\}$	$\{tpTrue,$ $tpDeliverBook1,$ $tpNotDeliverCD1\}$
$KIsFalse$	$nothing$	$nothing$	$nothing$	$\{tpPayBook1\}$

6 Conclusions and Related Works

The main contributions of this paper, with respect to our previous works and with respect to other approaches, are as follows. We show how conditional obligations and prohibitions with stating points and deadlines may be specified and monitored using OWL and SWRL with significant advantages with respect to other approach that use other formal languages. Moreover we propose a hybrid solution, based on an OWL ontology, SWRL rules, and a Java program, to the problem of monitoring the time evolution of obligations and prohibitions.

In particular if we compare this specification with another one that we presented elsewhere based on Event Calculus [10] we observe significant

[11] The ontology file, its representation in DL, the Java program and its output that represent the time evolution of the state of the system as depicted in Table 1 can be found at http://www.people.lu.unisi.ch/fornaran/ontology/DALT09Ontology.html

improvement in performance (even if a complete comparison will be possible only when the complete *OCeAN* meta-model will be formalized with Semantic Web Technology). Moreover semantic web technologies are becoming an international standard for web applications and numerous tools, reasoners, and libraries are available to support the development and usage of ontologies. This, in spite of the drawbacks on time reasoning and due to the limits of OWL language expressivity, is a crucial advantage with respect to other languages used in the multiagent community for the specification of norms and organizations, like as we already mentioned the Event Calculus [19,1], or other specific formal languages like the one required by the rule engine Jess [13,5], or a variant of Propositional Dynamic Logic (PDL) used to specify and verify liveness and safety properties of multi-agent system programs with norms [6], or Process Compliance Language (PCL) [14].

In literature there are few approaches that use semantic web languages for the specification of multiagent systems and in particular of obligations and prohibitions. One of the most interesting one is the approach for policy specification and management presented in the KAoS framework [18]. Even if in English the word *norm* and *policy* have different meaning and also in informatics literature they could be referred to two different concepts [3], in the MAS community they may have very close meanings. In KAoS a policy could be a positive or negative authorization to perform an action (that is a permission or a prohibition) or it can be an obligation. Like in our approach in KAoS policies are specified using a set of concepts defined in an OWL *core ontology* that could be extended with application dependent ontologies. A crucial difference between KAoS approach and the approach presented in this paper is in the methods used for monitoring and enforcement of policies or norms. In KAoS policies are usually regimented (as far as is possible given that it is almost impossible to regiment obligations [9]) by means of "guards" and are monitored by means of platform specific mechanisms. Differently in our proposal norms are enforced by means of sanctions or rewards and are monitored by deducing their fulfillment or violation with an OWL reasoner (we use Pellet but other OWL 2 reasoners could be used) and an external Java program.

Another example is the one presented in [15] where prohibited, obliged and permitted actions are represented as object properties from agents to actions. But without the reification of the notion of obligation and prohibition that we propose here, it is very difficult to find a feasible solution to express conditional commitments with deadlines and it is impossible to detect what norms and how many time were fulfilled or violated. Moreover the approach proposed for detecting violations is based on the external performance of SPARQL queries and on the update of the ABox to register that an obligation/prohibition resulted violated; however SPARQL queries do not exploit the semantics specified by the ontology, moreover it is necessary to write different queries for every possible different action that has to be monitored and for the execution of SPARQL queries it is necessary to use a proper additional tool.

In [2] a hybrid approach is presented: they define a communication acts ontology using OWL and express the semantics of those acts through social commitments that are formalized in the Event Calculus. This work is complementary with respect to our approach, in fact we specify also the semantics of social commitments using semantic web technologies. Semantic web technologies in multiagent systems can be used also to specify domain specific ontologies used in the content of norms like in [8].

Another interesting contribution of this work is due also to the exemplification of a solution to the problem to performing closed world reasoning on certain classes in OWL. Another work that tackles a similar problem in a different domain, the ontology of software models, is [4].

Indeed this model is still incomplete e we plan to investigate how it is possible to manage the creation of commitments to model norm activations, and to model active sanctions, moreover we plan to study how to formalize the notion of power to express the semantics of declarative communicative acts and of passive sanctions.

References

1. Artikis, A., Sergot, M., Pitt, J.: Animated Specifications of Computational Societies. In: Castelfranchi, C., Johnson, W.L. (eds.) Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002), pp. 535–542. ACM Press, New York (2002)
2. Berges, I., Bermúdez, J., Goñi, A., Illarramendi, A.: Semantic web technology for agent communication protocols. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 5–18. Springer, Heidelberg (2008)
3. Bradshaw, J., Beautment, P., Breedy, M., Bunch, L., Drakunov, S., Feltovich, P., Hoffman, R., Jeffers, R., Johnson, M., Kulkarni, S., Lott, J., Raj, A., Suri, N., Uszok, A.: Making agents acceptable to people, pp. 355–400. Springer, Heidelberg (2004)
4. Bräuer, M., Lochmann, H.: An ontology for software models and its practical implications for semantic web reasoning. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 34–48. Springer, Heidelberg (2008)
5. da Silva1, V.T.: From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behavior of agents. Autonomous Agents and Multi-Agent Systems 17(1), 113–155 (2008)
6. Dastani, M., Grossi, D., Meyer, J.-J., Tinnemeier, N.: Normative multi-agent programs and their logics. In: Boella, G., Noriega, P., Pigozzi, G., Verhagen, H. (eds.) Normative Multi-Agent Systems, Germany. Dagstuhl Seminar Proceedings, vol. 09121. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Dagstuhl (2009)
7. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A., Nutt, W.: An epistemic operator for description logics. Artificial Intelligence 200(1-2), 225–274 (1998)

8. Felicissimo, C., Briot, J.-P., Chopinaud, C., Lucena, C.: How to concretize norms in NMAS? An operational normative approach presented with a case study from the television domain. In: International Workshop on Coordination, Organization, Institutions and Norms in Agent Systems (COIN@AAAI 2008), 23rd AAAI Conference on Artificial Intelligence, Chicago, IL, Etats-Unis. AAAI Press, Menlo Park (2008)
9. Fornara, N., Colombetti, M.: Specifying and enforcing norms in artificial institutions. In: Baldoni, M., Son, T.C., van Riemsdijk, M.B., Winikoff, M. (eds.) DALT 2008. LNCS (LNAI), vol. 5397, pp. 1–17. Springer, Heidelberg (2009)
10. Fornara, N., Colombetti, M.: Specifying Artificial Institutions in the Event Calculus. In: Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models of Information science reference, ch. XIV, pp. 335–366. IGI Global (2009)
11. Fornara, N., Viganò, F., Colombetti, M.: Agent communication and artificial institutions. Autonomous Agents and Multi-Agent Systems 14(2), 121–142 (2007)
12. Fornara, N., Viganò, F., Verdicchio, M., Colombetti, M.: Artificial institutions: A model of institutional reality for open multiagent systems. Artificial Intelligence and Law 16(1), 89–105 (2008)
13. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: Constraint rule-based programming of norms for electronic institutions. Autonomous Agents and Multi-Agent Systems 18(1), 186–217 (2009)
14. Governatori, G., Rotolo, A.: How do agents comply with norms? In: Boella, G., Noriega, P., Pigozzi, G., Verhagen, H. (eds.) Normative Multi-Agent Systems, Dagstuhl, Germany. Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2009)
15. Lam, J.S.-C., Guerin, F., Vasconcelos, W., Norman, T.J.: Representing and reasoning about norm-governed organisations with semantic web languages. In: Sixth European Workshop on Multi-Agent Systems Bath, UK, December 18-19 (2008)
16. López, F., López, Luck, M., d'Inverno, M.: A Normative Framework for Agent-Based Systems. In: Proceedings of the First International Symposium on Normative Multi-Agent Systems, Hatfield (2005)
17. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Web Semantics: Science, Services and Agents on the World Wide Web 5(2), 51–53 (2007)
18. Uszok, A., Bradshaw, J.M., Lott, J., Breedy, M., Bunch, L., Feltovich, P., Johnson, M., Jung, H.: New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of kaos. In: IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 145–152 (2008)
19. Yolum, P., Singh, M.: Reasoning about commitment in the event calculus: An approach for specifying and executing protocols. Annals of Mathematics and Artificial Intelligence 42, 227–253 (2004)

Prioritized Goals and Subgoals in a Logical Account of Goal Change – A Preliminary Report

Shakil M. Khan and Yves Lespérance

Department of Computer Science and Engineering
York University, Toronto, ON, Canada
{skhan,lesperan}@cse.yorku.ca

Abstract. Most previous logical accounts of goal change do not deal with prioritized goals and do not handle subgoals and their dynamics properly. Many are restricted to achievement goals. In this paper, we develop a logical account of goal change that addresses these deficiencies. In our account, we do not drop lower priority goals permanently when they become inconsistent with other goals and the agent's knowledge; rather, we make such goals inactive. We ensure that the agent's chosen goals/intentions are consistent with each other and the agent's knowledge. When the world changes, the agent recomputes her chosen goals and some inactive goals may become active again. This ensures that our agent maximizes her utility. We also propose an approach for handling subgoals and their dynamics. We prove that the proposed account has some intuitively desirable properties.

1 Introduction

There has been much work on modeling agent's mental states, beliefs, goals, and intentions, and how they interact and lead to rational decisions about action. As well, there has been a lot of work on modeling belief change. But the dynamics of motivational attitudes has received much less attention. Most formal models of goal and goal change [1,2,3,4,5,6] assume that all goals are equally important and many only deal with achievement goals (one exception to this is the model of prioritized goals in [7]). Moreover, most of these frameworks do not guarantee that an agent's goals will properly evolve when an action/event occurs, e.g. when the agent's beliefs/knowledge changes or a goal is adopted or dropped. Also, they do not model the dependencies between goals and the subgoals and plans adopted to achieve these goals. For instance, subgoals and plans adopted to bring about a goal should be dropped when the parent goal becomes impossible, is achieved, or is dropped. Dealing with these issues is important for developing effective models of rational agency. It is also important for work on BDI agent programming languages, where handling declarative goals is an active research topic [8,9].

In this paper, we present a formal model of prioritized goals and their dynamics that addresses some of these issues. Specifically, we propose a framework, where an agent can have multiple goals at different priority levels, possibly inconsistent with each other. We define intentions as the maximal set of highest priority goals that is consistent given the agent's knowledge. Our model supports the specification of general

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 119–136, 2010.

temporally extended goals, not just achievement goals, and handles subgoals and their dynamics.

We start with a (possibly inconsistent) initial set of *prioritized goals* or desires that are totally ordered according to priority, and specify how these goals evolve when actions/events occur and the agent's knowledge changes. We define the agent's *chosen goals* or intentions in terms of this goal hierarchy. Our agents maximize their utility; they will abandon a chosen goal ϕ if an opportunity to commit to a higher priority but inconsistent with ϕ goal arises. To this end, we keep all prioritized goals in the goal base unless they are explicitly dropped. At every step, we compute an optimal set of chosen goals given the hierarchy of prioritized goals, preferring higher priority goals, such that chosen goals are consistent with each other and with the agent's knowledge. Thus at any given time, some goals in the hierarchy are active, i.e. chosen, while others are inactive. Some of these inactive goals may later become active, e.g. if a higher priority active goal that is currently blocking an inactive goal becomes impossible. We also show how the dependencies between goals and subgoals can be modeled. Finally, we prove some interesting properties about the dynamics of chosen goals.

As mentioned above, our formalization of prioritized goals ensures that the agent always tries to maximize her utility, and as such a limitation of our framework is that it displays an idealized form of rationality. In Section 5, we discuss how this relates to Bratman's theory of practical reasoning [10]. We use an action theory based on the situation calculus [11] along with our formalization of paths in the situation calculus as our base formalism.

The paper is organized as follows: in the next section, we outline our base framework. In Section 3, we formalize *paths* in the situation calculus to support modeling temporally extended goals. In Section 4, we present our model of prioritized goals. In section 5, we present our formalization of goal dynamics and discuss some of its properties. In Section 6, we discuss what it means for an agent to have a subgoal and how subgoals change as a result of changes to their parent goals. Then in the last section, we summarize our results, discuss previous work in this area, and point to possible future work.

2 Action and Knowledge

Our base framework for modeling goal change is the situation calculus [11] as formalized in [12]. In this framework, a possible state of the domain is represented by a situation. There is a set of initial situations corresponding to the ways the agents believe the domain might be initially, i.e. situations in which no actions have yet occurred. Init(s) means that s is an initial situation. The actual initial state is represented by a special constant S_0. There is a distinguished binary function symbol do where $do(a, s)$ denotes the successor situation to s resulting from performing the action a. Thus the situations can be viewed as a set of trees, where the root of each tree is an initial situation and the arcs represent actions. Relations (and functions) whose truth values vary from situation to situation, are called relational (functional, resp.) fluents, and are denoted by predicate (function, resp.) symbols taking a situation term as their last argument. There is a special predicate Poss(a, s) used to state that action a is executable in situation s.

Our framework uses a theory D_{basic} that includes the following set of axioms:[1] (1) action precondition axioms, one per action a characterizing $\text{Poss}(a, s)$, (2) successor state axioms (SSA), one per fluent, that succinctly encode both effect and frame axioms and specify exactly when the fluent changes [12], (3) initial state axioms describing what is true initially including the mental states of the agents, (4) unique name axioms for actions, and (5) domain-independent foundational axioms describing the structure of situations [14].

Following [15,16], we model knowledge using a possible worlds account adapted to the situation calculus. $K(s', s)$ is used to denote that in situation s, the agent thinks that she could be in situation s'. Using K, the knowledge of an agent is defined as:[2] $\text{Know}(\Phi, s) \stackrel{\text{def}}{=} \forall s'.\ K(s', s) \supset \Phi(s')$, i.e. the agent knows Φ in s if Φ holds in all of her K-accessible situations in s. K is constrained to be reflexive, transitive, and Euclidean in the initial situation to capture the fact that agents' knowledge is true, and that agents have positive and negative introspection. As shown in [16], these constraints then continue to hold after any sequence of actions since they are preserved by the successor state axiom for K. We also assume that all actions are public, i.e. whenever an action (including exogenous events) occurs, the agent learns that it has happened. Note that, we work with knowledge rather than belief. Although much of our formalization should extend to the latter, we leave this for future work.

3 Paths in the Situation Calculus

To support modeling temporally extended goals, we introduce a new sort of *paths*, with (possibly sub/super-scripted) variables p ranging over paths. A path is essentially an infinite sequence of situations, where each successor situation along the path can be reached by performing some *executable* action in the preceding situation. We introduce a predicate $\text{OnPath}(p, s)$, meaning that the situation s is on the path p. Also, $\text{Starts}(p, s)$ means that s is the starting situation of path p. A path p starts with the situation s iff s is the earliest situation on p:[3]

Axiom 1

$$\text{Starts}(p, s) \equiv \text{OnPath}(p, s) \wedge \forall s'.\ \text{OnPath}(p, s') \supset s \leq s'.$$

In the standard situation calculus, paths are implicitly there, and a path can be viewed as a pair (s, F) consisting of a situation s representing the starting situation of the path, and a function F from situations to actions (called *Action Selection Functions* (ASF) or

[1] We will be quantifying over formulae, and thus we assume D_{basic} includes axioms for encoding formulae as first order terms, as in [13]. We will also be using lists of integers, and assume that D_{basic} includes axiomatizations of integers and lists.

[2] A state formula $\Phi(s)$ takes a single situation as argument and is evaluated with respect to that situation. Φ may contain a placeholder constant *now* that stands for the situation in which Φ must hold. $\Phi(s)$ is the formula that results from replacing *now* by s. Where the intended meaning is clear, we sometimes suppress the placeholder.

[3] In the following, $s < s'$ means that s' can be reached from s by performing a sequence of executable actions. $s \leq s'$ is an abbreviation for $s < s' \vee s = s'$.

strategies in [5]), such that from the starting situation s, F defines an infinite sequence of situations by specifying an action for every situation starting from s. Thus, one way of axiomatizing paths is by making them correspond to such pairs (s, F):

Axiom 2

$$\forall p.\ \mathrm{Starts}(p, s) \supset (\exists F.\ \mathrm{Executable}(F, s) \wedge \forall s'.\ \mathrm{OnPath}(p, s') \equiv \mathrm{OnPathASF}(F, s, s')),$$
$$\forall F, s.\ \mathrm{Executable}(F, s) \supset \exists p.\ \mathrm{Starts}(p, s) \wedge \forall s'.\ \mathrm{OnPathASF}(F, s, s') \equiv \mathrm{OnPath}(p, s').$$

This says that for every path there is an executable ASF that produces exactly the sequence of situations on the path from its starting situation. Also, for every executable ASF and situation, there is a path that corresponds to the sequence of situations produced by the ASF starting from that situation.

$$\mathrm{OnPathASF}(F, s, s') \stackrel{\mathrm{def}}{=} s \leq s' \wedge \forall a, s^*.\ s < do(a, s^*) \leq s' \supset F(s^*) = a,$$
$$\mathrm{Executable}(F, s) \stackrel{\mathrm{def}}{=} \forall s'.\ \mathrm{OnPathASF}(F, s, s') \supset \mathrm{Poss}(F(s'), s').$$

Here, $\mathrm{OnPathASF}(F, s, s')$ [6] means that the situation sequence defined by (s, F) includes the situation s'. Also, the situation sequence encoded by a strategy F and a starting situation s is executable iff for all situations s' on this sequence, the action selected by F in s' is executable in s'.

In our framework, we will use both state and path formulae. A state formula is a formula that has a free situation variable in it, whereas a path formula is one that has a free path variable. State formulae are used in the context of knowledge while path formulae are used in that of goals. We use $\Phi(s), \Psi(s), \cdots$ and $\phi(p), \psi(p), \cdots$ possibly with decorations to represent state and path formulae, respectively. Note that, by incorporating infinite paths in our framework, we can evaluate goals over these and handle arbitrary temporally extended goals; thus, unlike some other situation calculus based accounts where goal formulae are evaluated w.r.t. finite paths (e.g. [7]), we can handle for example unbounded maintenance goals.

We next define some useful constructs. A state formula Φ *eventually holds* over the path p if Φ holds in some situation that is on p, i.e. $\Diamond\Phi(p) \stackrel{\mathrm{def}}{=} \exists s'.\ \mathrm{OnPath}(p, s') \wedge \Phi(s')$. Other Temporal Logic operators can be defined similarly, e.g. always Φ: $\Box\Phi(p)$.

An agent *knows* in s that ϕ has become *inevitable* if ϕ holds over all paths that starts with a K-accessible situation in s:

$$\mathrm{KInevitable}(\phi, s) \stackrel{\mathrm{def}}{=} \forall p.\ \mathrm{Starts}(p, s') \wedge K(s', s) \supset \phi(p).$$

An agent knows in s that ϕ is impossible if she knows that $\neg\phi$ is inevitable in s, i.e. $\mathrm{KImpossible}(\phi, s) \stackrel{\mathrm{def}}{=} \mathrm{KInevitable}(\neg\phi, s)$.

Thirdly, we define what it means for a path p' to be a suffix of another path p w.r.t. a situation s:

$$\mathrm{Suffix}(p', p, s) \stackrel{\mathrm{def}}{=} \mathrm{OnPath}(p, s) \wedge \mathrm{Starts}(p', s)$$
$$\wedge \forall s'.\ s \leq s' \supset \mathrm{OnPath}(p, s') \equiv \mathrm{OnPath}(p', s').$$

That is, a path p' is a suffix of another path p w.r.t. a situation s iff s is on p, and p', which starts with s, is exactly the same as the subpath of p that starts with s.

Fourthly, SameHistory(s_1, s_2) means that the situations s_1 and s_2 share the same history of actions, but perhaps starting from different initial situations:

Axiom 3

$$\text{SameHistory}(s_1, s_2) \equiv (\text{Init}(s_1) \wedge \text{Init}(s_2)) \\ \vee (\exists a, s_1', s_2'.\ s_1 = do(a, s_1') \wedge s_2 = do(a, s_2') \wedge \text{SameHistory}(s_1', s_2')).$$

Thus, if s_1 can be reached from some initial situation by performing a sequence of actions σ, then s_2 can be reached from a (possibly different) initial situation by executing σ.

Finally, we say that ϕ has become *inevitable* in s if ϕ holds over all paths that starts with a situation that has the same action history as s:

$$\text{Inevitable}(\phi, s) \stackrel{\text{def}}{=} \forall p, s'.\ \text{Starts}(p, s') \wedge \text{SameHistory}(s', s) \supset \phi(p).$$

4 Prioritized Goals

Most work on formalizing goals only deals with static goal semantics and not their dynamics. There are two main categories of motivational attitudes, namely goal [1,17] (AKA choice [2], wish [10] or preference), and intention. While goals are sometimes allowed to be inconsistent [10], intentions are mostly required to be consistent. Another difference is that agents are committed to their intentions, but not necessarily to their goals [10]. Intention is sometimes primitive [17,3] and sometimes a defined concept, specified in terms of goals [1,2,4]. In this section, we formalize goals or desires with different priorities, which we call *prioritized goals* (p-goals, henceforth). These p-goals are not required to be mutually consistent and need not be actively pursued by the agent. In terms of these, we define the consistent set of *chosen goals* or intentions (c-goals, henceforth) that the agent is committed to. In the next section, we formalize goal dynamics by providing a SSA for p-goals. The agent's c-goals are automatically updated when her p-goals change. We deal with subgoals and their dynamics in Section 6.

Not all of the agent's goals are equally important to her. Thus, it is useful to support a priority ordering over goals. This information can be used to decide which of the agent's c-goals should no longer be actively pursued in case they become mutually inconsistent. Following [6], we specify each p-goal by its own accessibility relation/fluent G. A path p is G-accessible at priority level n in situation s (denoted by $G(p, n, s)$) if all the goals of the agent at level n are satisfied over this path and if it starts with a situation that has the same history (in terms of the actions performed so far) as s. The latter requirement ensures that the agent's p-goal-accessible paths reflect the actions that have been performed so far. A smaller n represents higher priority, and the highest priority level is 0. Thus in this framework, we assume that the set of p-goals are totally ordered according to priority. We say that an agent has the p-goal that ϕ at level n in situation s iff ϕ holds over all paths that are G-accessible at n in s:

$$\text{PGoal}(\phi, n, s) \stackrel{\text{def}}{=} \forall p.\ G(p, n, s) \supset \phi(p).$$

To be able to refer to all the p-goals of the agent at some given priority level, we also define *only p-goals*.

$$\text{OPGoal}(\phi, n, s) \stackrel{\text{def}}{=} \text{PGoal}(\phi, n, s) \wedge (\forall p.\ \phi(p) \supset G(p, n, s)).$$

An agent has the *only p-goal* that ϕ at level n in situation s iff ϕ is a p-goal at n in s, and any path over which ϕ holds is G-accessible at n in s.

A domain theory for our framework D includes the axioms of a theory D_{basic} as in the previous section, the axiomatization of paths i.e. axioms 1-3, domain dependent initial goal axioms (see below), the domain independent axioms 4-7 and the definitions that appear in this section and the next. The modeler must provide initial goal axioms of the following form:

Initial Goal Axioms

$$\begin{aligned}
&\text{(a) } \text{Init}(s) \supset ((G(p, 0, s) \equiv \text{Starts}(p, s') \wedge \text{Init}(s') \wedge \phi_0(p)) \\
&\qquad \wedge (G(p, 1, s) \equiv \text{Starts}(p, s') \wedge \text{Init}(s') \wedge \phi_1(p)) \wedge \cdots \\
&\qquad \wedge (G(p, k-1, s) \equiv \text{Starts}(p, s') \wedge \text{Init}(s') \wedge \phi_{k-1}(p))), \\
&\text{(b) } \forall n, p, s.\ \text{Init}(s) \wedge n \geq k \supset (G(p, n, s) \equiv \text{Starts}(p, s') \wedge \text{Init}(s')), \\
&\text{(c) } \text{Init}(s) \supset \mathit{NPGoals}(s) = k.
\end{aligned}$$

The p-goals $\phi_0, \phi_1, \cdots, \phi_{k-1}$ (from highest to lowest priority) of the agent in the initial situations are specified by the Initial Goal Axiom (a); each of them defines a set of initial goal paths for a given priority level, and must be consistent. We assume that the agent has a finite number k of initial p-goals. For $n \geq k$, we make $G(p, n, s)$ true for every path p that starts with an initial situation in (b). Thus at levels $n \geq k$, the agent has the trivial p-goal that she be in an initial situation. We also have a distinguished functional fluent $\mathit{NPGoals}(s)$ that represents the number of prioritized goals that the agent has (i.e. the location of the first empty slot after the last p-goal). Initially $\mathit{NPGoals}$ is set to k in (c). Later, we will specify the dynamics of p-goals by giving SSAs for G and $\mathit{NPGoals}$.

We use the following as a running example. We have an agent who initially has the following three p-goals: $\phi_0 = \Box\text{BeRich}$, $\phi_1 = \Diamond\text{GetPhD}$, and $\phi_2 = \Box\text{BeHappy}$ at level 0, 1, and 2, respectively (see second column of Table 1). Assume that while initially the agent knows that all of her p-goals are individually achievable, she knows that her p-goal $\Diamond$GetPhD is inconsistent with her highest priority p-goal $\Box$BeRich as well as with her p-goal $\Box$BeHappy, while the latter are consistent with each other. It is straightforward to specify a domain action theory such that it entails this. Thus in our example, we have $\text{OPGoal}(\phi_i(p) \wedge \text{Starts}(p, s) \wedge \text{Init}(s), i, S_0)$, for $i = 0, 1, 2$. Also, for any $n \geq 3$, we have $\text{OPGoal}(\text{Starts}(p, s) \wedge \text{Init}(s), n, S_0)$.

Table 1. Example of an Agent's PGoals and their Dynamics

G-Level	S_0, S_1'	S_1	S_2	S_3
4	TRUE	TRUE	$\Box$BeRich $\wedge$ $\Box$WorkHard $\wedge$ $\Box$BeEnergetic	TRUE
3	TRUE	$\Box$BeRich $\wedge$ $\Box$WorkHard	$\Box$BeRich $\wedge$ $\Box$WorkHard	TRUE
2	$\Box$BeHappy	$\Box$BeHappy	$\Box$BeHappy	$\Box$BeHappy
1	$\Diamond$GetPhD	$\Diamond$GetPhD	$\Diamond$GetPhD	$\Diamond$GetPhD
0	$\Box$BeRich	$\Box$BeRich	$\Box$BeRich	$\Box$BeRich

While p-goals or desires are allowed to be known to be impossible to achieve, an agent's c-goals or intentions must be realistic. Not all of the G-accessible paths are realistic in the sense that they start with a K-accessible situation. To filter these out, we define *realistic* p-goal accessible paths:

$$G_R(p, n, s) \stackrel{\text{def}}{=} G(p, n, s) \wedge \text{Starts}(p, s') \wedge K(s', s),$$

i.e., a path p is G_R-accessible at level n in situation s if it is G-accessible at n in s, and if p starts with a situation that is K-accessible in s. Thus G_R prunes out the paths from G that are known to be impossible, and since we define c-goals in terms of realistic p-goals, this ensures that c-goals are realistic. We say that an agent has the *realistic p-goal* that ϕ at level n in situation s iff ϕ holds over all paths that are G_R-accessible at n in s:

$$\text{RPGoal}(\phi, n, s) \stackrel{\text{def}}{=} \forall p.\ G_R(p, n, s) \supset \phi(p).$$

Using realistic p-goals, we next define c-goals. The idea of how we compute c-goal-accessible paths is as follows: the set of G_R-accessibility relations represents a set of prioritized temporal propositions that are candidates for the agent's c-goals. Given G_R, in each situation we want to compute the agent's c-goals such that it is the *maximal consistent* set of higher priority realistic p-goals. We do this iteratively starting with the set of all possible paths (i.e. paths that starts with a K-accessible situation). At each iteration we compute the intersection of this set with the next highest priority set of G_R-accessible paths. If the intersection is not empty, we thus obtain a new chosen set of paths at level i. We call a p-goal chosen by this process an *active* p-goal. If on the other hand, the intersection is empty, then it must be the case that the p-goal represented by this level is either in conflict with another active higher priority p-goal/a combination of two or more active higher priority p-goals, or is known to be impossible. In that case, that p-goal is ignored (i.e. marked as inactive), and the chosen set of paths at level i is the same as at level $i - 1$. We repeat this until we reach $i = NPGoals$. Axiom 4 "computes" this intersection:[4]

Axiom 4

$$\begin{aligned} G_\cap(p, n, s) \equiv\ & \textbf{if } (n = 0) \textbf{ then} \\ & \quad \textbf{if } \exists p'.\ G_R(p', n, s) \textbf{ then } G_R(p, n, s) \\ & \quad \textbf{else } \text{Starts}(p, s') \wedge K(s', s) \\ & \textbf{else} \\ & \quad \textbf{if } \exists p'.(G_R(p', n, s) \wedge G_\cap(p', n - 1, s)) \\ & \quad\ \textbf{then } (G_R(p, n, s) \wedge G_\cap(p, n - 1, s)) \\ & \quad \textbf{else } G_\cap(p, n - 1, s). \end{aligned}$$

C-goal accessible paths are the result of this intersection after all priority levels have been considered:

$$G_C(p, s) \stackrel{\text{def}}{=} G_\cap(p, NPGoals(s) - 1, s).$$

[4] **if** ϕ **then** ψ_1 **else** ψ_2 is an abbreviation for $(\phi \supset \psi_1) \wedge (\neg\phi \supset \psi_2)$.

We define an agent's c-goals in terms of the G_C-accessible paths:

$$\text{CGoal}(\phi, s) \stackrel{\text{def}}{=} \forall p.\ G_C(p, s) \supset \phi(p),$$

i.e., the agent has the c-goal that ϕ if ϕ holds over all of her G_C-accessible paths.

We also define what it means for an agent to have a c-goal at some level n:

$$\text{CGoal}(\phi, n, s) \stackrel{\text{def}}{=} \forall p.\ G_{\cap}(p, n, s) \supset \phi(p),$$

i.e. an agent has the c-goal at level n that ϕ if ϕ holds over all paths that are in the prioritized intersection of the set of G_R-accessible paths up to level n.

In our example, the agent's realistic p-goals are $\Box$BeRich, $\Diamond$GetPhD, and $\Box$BeHappy in order of priority. The $G_{\cap}$-accessible paths at level 0 in S_0 are the ones that start with a K-accessible situation and where $\Box$BeRich holds. The $G_{\cap}$-accessible paths at level 1 in S_0 are the same as at level 0, since there are no K-accessible paths over which both $\Diamond$GetPhD and $\Box$BeRich hold. Finally, the $G_{\cap}$-accessible paths at level 2 in S_0 and hence the G_C-accessible paths are those that start with a K-accessible situation and over which $\Box$BeRich $\wedge$ $\Box$BeHappy holds. Also, it can be shown that initially our example agent has the c-goals that $\Box$BeRich and $\Box$BeHappy, but not $\Diamond$GetPhD.

Note that by our definition of c-goals, the agent can have a c-goal that ϕ in situation s for various reasons: 1) ϕ is known to be inevitable in s; 2) ϕ is an active p-goal at some level n in s; 3) ϕ is a consequence of two or more active p-goals at different levels in s. To be able to refer to c-goals for which the agent has a primitive motivation, i.e. c-goals that result from a single active p-goal at some priority level n, in contrast to those that hold as a consequence of two or more active p-goals at different priority levels, we define *primary* c-goals:

$$\text{PrimCGoal}(\phi, s) \stackrel{\text{def}}{=} \exists n.\ \text{PGoal}(\phi, n, s) \wedge \exists p.\ G(p, n, s) \wedge G_{\cap}(p, n, s).$$

That is, an agent has the primary c-goal that ϕ in situation s, if ϕ is a p-goal at some level n in s, and if there is a G-accessible path p at n in s that is also in the prioritized intersection of G_R-accessible paths upto n in s. The last two conjucts are required to ensure that n is an active level. Thus if an agent has a primary c-goal that ϕ, then she also has the c-goal that ϕ, but not necessarily vice-versa. It can be shown that initially our example agent has the primary c-goals that $\Box$BeRich and $\Box$BeHappy, but not their conjunction. This shows that (strictly speaking) primary c-goals are not closed under logical consequence.

5 Goal Dynamics

An agent's goals change when her knowledge changes as a result of the occurrence of an action (including exogenous events), or when she adopts or drops a goal. We formalize this by specifying how p-goals change. C-goals are then computed using realistic p-goals in every new situation as explained above.

We introduce two actions for adopting and dropping a p-goal, $adopt(\phi)$ and $drop(\phi)$, and a third for adopting a subgoal ψ w.r.t. a supergoal ϕ, $adopt(\psi, \phi)$. The action precondition axioms for these are as follows:

Axiom 5

$$\begin{aligned}&\mathrm{Poss}(adopt(\phi), s) \equiv \neg\exists n.\ \mathrm{PGoal}(\phi, n, s),\\ &\mathrm{Poss}(adopt(\psi, \phi), s) \equiv \neg\exists n.\ \mathrm{PGoal}(\psi, n, s) \wedge \exists n'.\ \mathrm{PGoal}(\phi, n', s),\\ &\mathrm{Poss}(drop(\phi), s) \equiv \exists n.\ \mathrm{PGoal}(\phi, n, s).\end{aligned}$$

That is, an agent can adopt the p-goal that ϕ, if she does not already have ϕ as her p-goal at some level. An agent can adopt a subgoal ψ w.r.t. the parent goal that ϕ if she does not already have the p-goal that ψ at some level, and if at some level she currently has the parent goal that ϕ. The $drop(\phi)$ action is possible in s if ϕ is a p-goal at some level n in s.

In the following, we specify the dynamics of p-goals by giving the SSA for G and then discuss each case, one at a time:

Axiom 6 (SSA for G)

$$\begin{aligned}&G(p, n, do(a, s)) \equiv\\ &\quad \forall\phi, \psi.\ (a \neq adopt(\phi) \wedge a \neq adopt(\psi, \phi) \wedge a \neq drop(\phi) \wedge \mathrm{Progressed}(p, n, a, s))\\ &\quad \vee \exists\phi.\ (a = adopt(\phi) \wedge \mathrm{Adopted}(p, n, a, s, \phi))\\ &\quad \vee \exists\phi, \psi.\ (a = adopt(\psi, \phi) \wedge \mathrm{SubGoalAdopted}(p, n, a, s, \psi, \phi)\\ &\quad \vee \exists\phi.\ (a = drop(\phi) \wedge \mathrm{Dropped}(p, n, a, s, \phi)).\end{aligned}$$

The overall idea of the SSA for G is as follows. First of all, to handle the occurrence of a non-adopt/drop (i.e. regular) action a, we progress all G-accessible paths to reflect the fact that this action has just happened; this is done using the $\mathrm{Progressed}(p, n, a, s)$ construct, which replaces each G-accessible path p' with starting situation s', by its suffix p provided that it starts with $do(a, s')$:

$$\mathrm{Progressed}(p, n, a, s) \stackrel{\mathrm{def}}{=} \exists p'.\ G(p', n, s) \wedge \mathrm{Starts}(p', s') \wedge \mathrm{Suffix}(p, p', do(a, s')).$$

Any path over which the next action performed is not a is eliminated from the respective G accessibility level.

Secondly, to handle adoption of a p-goal ϕ, we add a new proposition containing the p-goal to the agent's goal hierarchy. We assume that the newly adopted p-goal ϕ has the lowest priority. Thus in addition to progressing the G-accessible paths at all levels as above, we eliminate the paths over which ϕ does not hold from the $NPGoals(s)$-th G-accessibility level, and the agent acquires the p-goal that ϕ at level $NPGoals(s)$:

$$\begin{aligned}\mathrm{Adopted}(p, n, a, s, \phi) \stackrel{\mathrm{def}}{=} &\textbf{ if } (n = NPGoals(s)) \textbf{ then } (\mathrm{Progressed}(p, n, a, s) \wedge \phi(p))\\ &\textbf{ else } \mathrm{Progressed}(p, n, a, s).\end{aligned}$$

The third case of subgoal adoption is discussed in the next section.

Finally, to handle dropping of a p-goal ϕ, we replace the propositions that imply the dropped goal in the agent's goal hierarchy by the "trivial" proposition that the history of actions in the current situation has occurred. Thus in addition to progressing all G-accessible paths as above, we add back all paths that share the same history with $do(a, s)$

to the existing G-accessibility levels where the agent has the p-goal that ϕ, and thus these G-accessibility levels now amount to the "trivial" p-goal that CorrectHist(s, $path$).[5]

$$\begin{aligned}&\text{Dropped}(p, n, a, s, \phi) \stackrel{\text{def}}{=}\\ &\quad \textbf{if } \text{PGoal}(\phi, n, s) \textbf{ then } \exists s'.\ \text{Starts}(p, s') \wedge \text{SameHistory}(s', do(a, s))\\ &\quad \textbf{else } \text{Progressed}(p, n, a, s).\end{aligned}$$

The SSA for $NPGoals(s)$ is as follows:

Axiom 7 (SSA for $NPGoals(s)$)

$$\begin{aligned}&NPGoals(do(a, s)) = k \equiv\\ &\quad \neg(\exists \phi.\ a = adopt(\phi)) \wedge \neg(\exists \psi, \phi.\ a = adopt(\psi, \phi)) \wedge NPGoals(s) = k \vee\\ &\quad \exists \phi.\ a = adopt(\phi) \wedge NPGoals(s) + 1 = k \vee\\ &\quad \exists \psi, \phi.\ a = adopt(\psi, \phi) \wedge \text{AdjustSubGoalAdopt}(\phi, s) = k.\end{aligned}$$

That is, when the agent adopts a p-goal, her current $NPGoals$ is incremented by one. We discuss the adjustment of $NPGoals$ required for subgoal adoption in the next section. Finally, $NPGoals$ is not affected by any other action.

Returning to our example, recall that our agent has the c-goals/active p-goals in S_0 that □BeRich and □BeHappy, but not ◇GetPhD, since the latter is inconsistent with her higher priority p-goal □BeRich. Assume that, after the action *goBankrupt* happens in S_0, the p-goal □BeRich becomes impossible. Then in $S_1' = do(goBankrupt, S_0)$, the agent has the c-goal that ◇GetPhD, but not □BeRich nor □BeHappy; □BeRich is excluded from the set of c-goals since it has become impossible to achieve (i.e. unrealistic). Also, since her higher priority p-goal ◇GetPhD is inconsistent with her p-goal □BeHappy, the agent will make □BeHappy inactive.

Note that, while it might be reasonable to drop a p-goal (e.g. ◇GetPhD) that is in conflict with another higher priority active p-goal (e.g. □BeRich), in our framework we keep such p-goals around. The reason for this is that although □BeRich is currently inconsistent with ◇GetPhD, the agent might later learn that □BeRich has become impossible to bring about (e.g. after *goBankrupt* occurs), and then might want to pursue ◇GetPhD. Thus, it is useful to keep these inactive p-goals since this allows the agent to maximize her utility (that of her chosen goals) by taking advantage of such opportunities. As mentioned earlier, c-goals are our analogue to intentions. Recall that Bratman's model of intentions limits the agent's practical reasoning – agents do not always optimize their utility and don't always reconsider all available options in order to allocate their reasoning effort wisely. In contrast to this, our c-goals are defined in terms of the p-goals, and at every step, we ensure that the agent's c-goals maximizes her utility so that these are the set of highest priority goals that are consistent given the agent's knowledge. Thus, our notion of c-goals is not as persistent as Bratman's notion of intention [10]. For instance as mentioned above, after the action *goBankrupt* happens in S_0, the agent will lose the c-goal that □BeHappy, although she did not drop it and it did not

[5] CorrectHist(s, $path$) is defined as Starts($path$, s') ∧ SameHistory(s', s); here $path$ is a placeholder that stands for a path and s represents the current situation.

become impossible or achieved. In this sense, our model is that of an idealized agent. There is a tradeoff between optimizing the agent's chosen set of prioritized goals and being committed to chosen goals. In our framework, chosen goals behave like intentions with an automatic filter-override mechanism [10] that forces the agent to drop her chosen goals when opportunities to commit to other higher priority goals arise. In the future, it would be interesting to develop a logical model that captures the pragmatics of intention reconsideration by supporting control over it.

We now show that our formalization of prioritized goals has some desirable properties. Some of these (e.g. Proposition 3a) are analogues of the AGM postulates; others (e.g. adopting logically equivalent goals has the same result, etc.) were left out for space reasons. First we show that c-goals are consistent:

Proposition 1 (Consistency)

$$D \models \forall s.\ \neg \mathrm{CGoal}(\mathrm{False}, s).$$

Thus, the agent cannot have both ϕ and $\neg\phi$ as c-goals in a situation s and there is a path that is G_C-accessible in s. Even if all of the agent's p-goals become known to be impossible, the set of G_C-accessible paths will be precisely those that starts with a K-accessible situation, and thus the agent will only choose the propositions that are known to be inevitable.

We also have the property of realism [1], i.e. if an agent knows that something has become inevitable, then she has this as a c-goal:

Proposition 2 (Realism)

$$D \models \forall \phi, s.\ \mathrm{KInevitable}(\phi, s) \supset \mathrm{CGoal}(\phi, s).$$

Note that this is not necessarily true for p-goals and primary c-goals – an agent may know that something has become inevitable and not have it as her p-goal/primary c-goal, which is intuitive. In fact, this is the reason why we define p-goals in terms of G-accessible paths rather than G_R. While the property of realism is often criticized, one should view these inevitable goals as something that holds in the worlds that the agent intends to bring about, rather than something that the agent is actively pursuing.

A consequence of Proposition 1 and 2 is that an agent does not have a c-goal that is known to be impossible, i.e. $D \models \forall \phi, s.\ \mathrm{CGoal}(\phi, s) \supset \neg \mathrm{KImpossible}(\phi, s)$.

We next discuss some properties of the framework w.r.t. goal change. Proposition 3 says that (a) an agent acquires the p-goal that ϕ at some level n after she adopts it, and (b) that she acquires the primary c-goal (and c-goal) that ϕ after she adopts it in s, provided that she does not have the c-goal in s that $\neg\phi$ next.

Proposition 3 (Adoption)

(a) $D \models \exists n.\ \mathrm{PGoal}(\phi, n, do(adopt(\phi), s))$,

(b) $D \models \neg \mathrm{CGoal}(\neg \exists s', p'.\ \mathrm{Starts}(s') \wedge \mathrm{Suffix}(p', do(adopt(\phi), s')) \wedge \phi(p'), s)$
$\supset \mathrm{PrimCGoal}(\phi, do(adopt(\phi), s))$.

We can also show that after dropping the p-goal that ϕ at n in s, an agent does not have the p-goal (and thus the primary c-goal) that the progression of ϕ at n, i.e. ProgressionOf$(\phi, drop(\phi), s)$, provided that ProgressionOf$(\phi, drop(\phi), s)$ is not inevitable in $do(drop(\phi), s)$.

Proposition 4 (Drop)

$$\begin{aligned} D \models\ & \text{PGoal}(\phi, n, s) \\ & \wedge \neg\text{Inevitable}(\text{ProgressionOf}(\phi, drop(\phi), s), do(drop(\phi), s)) \\ & \supset \neg\text{PGoal}(\text{ProgressionOf}(\phi, drop(\phi), s), n, do(drop(\phi), s)), \end{aligned}$$

where,

$$\text{ProgressionOf}(\phi, a, s) \stackrel{\text{def}}{=} \exists p', s'.\ \text{Starts}(p', s') \wedge \text{Suffix}(p', do(a, s')) \wedge \phi(p').$$

Note that, this does not hold for CGoal, as ϕ could still be a consequence of two or more of her remaining primary c-goals.

The next few properties concern the persistence of these motivational attitudes. First, we have a persistence property for achievement realistic p-goals:

Proposition 5 (Persistence of Achievement RPGoals)

$$D \models \text{RPGoal}(\Diamond\Phi, n, s) \wedge \text{Know}(\neg\Phi, s) \wedge \forall\psi.\ a \neq drop(\psi) \supset \text{RPGoal}(\Diamond\Phi, n, do(a, s)).$$

This says that if an agent has a realistic p-goal that $\Diamond\Phi$ in s, then she will retain this realisitc p-goal after some action a has been performed in s, provided that she knows that Φ has not yet been achieved, and a is not the action of dropping a p-goal. Note that, we do not need to ensure that $\Diamond\Phi$ is still known to be possible or consistent with higher priority active p-goals, since the SSA for G does not automatically drop such incompatible p-goals from the goal hierarchy.

For achievement chosen goals we have the following:

Proposition 6 (Persistence of Achievement Chosen Goals)

$$\begin{aligned} D \models\ & \text{OPGoal}(\Diamond\Phi \wedge \text{CorrectHist}(s), n, s) \wedge \text{CGoal}(\Diamond\Phi, s) \\ & \wedge \text{Know}(\neg\Phi, s) \wedge \forall\psi.\ a \neq drop(\psi) \wedge \neg\text{CGoal}(\neg\Diamond\Phi, n-1, do(a, s)) \\ & \supset \text{CGoal}(\Diamond\Phi, n, do(a, s)). \end{aligned}$$

Thus, in situation s, if an agent has the only p-goal at level n that $\Diamond\Phi$ and the correct history of actions in s has been performed, and if $\Diamond\Phi$ is also a chosen goal in s (and thus she has the primary c-goal that $\Diamond\Phi$), then she will retain the c-goal that $\Diamond\Phi$ at level n after some action a has been performed in s, provided that:

- she knows in s that Φ has not yet been achieved,
- that a is not the action of dropping a p-goal,
- and that at level $n-1$ the agent does not have the c-goal in $do(a, s)$ that $\neg\Diamond\Phi$, i.e. $\Diamond\Phi$ is consistent with higher priority c-goals after a has been performed in s.

Note that this property also follows if we replace the consequent with CGoal($\Diamond\Phi$, $do(a, s)$), and thus it deals with the persistence of c-goals. Note however that, it does not hold if we replace the OPGoal in the antecedent with PGoal; the reason for this is that the agent might have a p-goal at level n in s that ϕ and the c-goal in s that ϕ, but not have ϕ as a primary c-goal in s, e.g. n might be an inactive level because another p-goal at n has become impossible, and ϕ could be a c-goal in s because it is a consequence of two other primary c-goals. Thus even if $\neg\phi$ is not a c-goal after a has been performed in s, there is no guarantee that the level n will be active in $do(a, s)$ or that all the active p-goals that contributed to ϕ in s are still active.

We believe that the dropping of an unrelated p-goal will not affect persistence, and hence it should be possible to strengthen Proposition 5 and 6. Also, in the future we would like to generalize these two propositions to deal with arbitrary temporally extended goals.

6 Handling Subgoals

In this section, we deal with the dynamics of subgoals. As mentioned earlier, a subgoal must be dropped when the parent goal is dropped or becomes impossible. When adopting a subgoal ψ with respect to a supergoal ϕ, in addition to recording the newly adopted goal ψ, we need to model the fact that ψ is a subgoal of ϕ. This information can later be used to drop the subgoal when the parent goal is dropped. One way of modeling this is to ensure that the adoption of a subgoal ψ w.r.t. a parent goal ϕ adds new p-goals that contain *both this subgoal and this parent goal i.e.* $\psi \wedge \phi$ *at a lower priority than the parent goal* ϕ. This ensures that when the parent goal is dropped, the subgoal is also dropped. To see this, recall that to handle the dropping of a goal ϕ, we drop the p-goals at all G-accessibility levels that imply ϕ. Thus, if we drop the parent goal ϕ, it will also drop all of its subgoals including ψ, since the G-accessibility levels where the parent goal ϕ holds include the G-accessibility levels where the subgoal ψ holds. Note that, if there are more than one level where the supergoal ϕ is a p-goal, then we copy all these levels, i.e. for each level n where ϕ is a p-goal, we add a (lower priority) level to the goal hierarchy. As we will see, this ensures that the sub-subgoals and sub-sub-subgoals etc. are also properly dropped when the supergoal is dropped. Also, this means that dropping a subgoal does not necessarily drop the supergoal.

Before going over the formal details, let us mention some useful bookkeeping tools that we will use: Length(l) returns the number of elements in list l; Nth(l, i) returns the i-th element in list l, and -1 if $i >$ Length(l); Sort(l) returns a sorted version of list l. The part of the SSA for G that handles subgoal adoption is defined as follows:

$$\begin{aligned}\text{SubGoalAdopted}(p, n, a, s, \psi, \phi) \stackrel{\text{def}}{=} & (n < \mathit{NPGoals}(s) \wedge \text{Progressed}(p, n, a, s))) \vee \\ & (\mathit{NPGoals}(s) \leq n < \mathit{NPGoals}(s) + \text{Length}(\text{AddList}(\phi, s)) \\ & \quad \wedge \exists i, m.\ (n = \mathit{NPGoals}(s) + i \wedge m = \text{Nth}(\text{AddList}(\phi, s), i) \\ & \quad \wedge \text{Progressed}(p, m, a, s) \wedge \psi(p))) \vee \\ & (n \geq \mathit{NPGoals}(s) + \text{Length}(\text{AddList}(\phi, s)) \wedge \text{Progressed}(p, n, a, s)).\end{aligned}$$

That is, if the action involves the adoption of a subgoal ψ w.r.t. a supergoal ϕ, we adjust G to incorporate (possibly several) new p-goals. We will discuss each case in turn. First,

note that the existing p-goals are just carried over by progressing them; this is handled by the first disjunct.

Secondly, we adjust G starting at level $NPGoals(s)$. We add a number of new levels that include the conjunction of the only p-goal and the subgoal at a lower priority for all the current only p-goals that imply the parent goal ϕ. For example, say at level i we have an OPGoal that ϕ_i and it implies the parent goal that ϕ; then we add at a lower priority the conjoined goal of the progressed version of ϕ_i and the subgoal ψ. Our formalization of this uses the abbreviation AddList(ϕ, s) which is a sorted list of levels such that the parent goal is implied by the only p-goal at this level. AddList is defined as: AddList(ϕ, s) $\stackrel{\text{def}}{=}$ Sort($[n \mid \text{PGoal}(\phi, n, s)]$). The length of this list indicates the number of lower priority goals that needs to be added. As discussed above, this ensures that the agent will drop the subgoal when the parent goal is dropped (but not necessarily vice-versa). Note that if this process adds two or more new p-goals to the agent's goal hierarchy, we maintain the original ordering; e.g. suppose that the agent adopted ψ w.r.t. ϕ, that there are two G-accessibility levels m and n such that the agent has the only p-goal that ϕ_m at m and ϕ_n at n, that ϕ_m implies ϕ and ϕ_n implies ϕ, and that $n > m$. In that case, the SSA for G will add the p-goal $\phi_m \wedge \phi$ at level $NPGoals(s)$ and the p-goal $\phi_n \wedge \phi$ at level $NPGoals(s) + 1$.

Finally, all the remaining levels involving trivially true goals are just carried over by progressing them.

The part of the SSA for $NPGoals$ that handles subgoal adoption is defined as follows:

$$\text{AdjustSubGoalAdopt}(\phi, s) \stackrel{\text{def}}{=} NPGoals(s) + \text{Length}(\text{AddList}(\phi, s)).$$

That is, when the agent adopts a subgoal w.r.t. a parent goal, her current $NPGoals$ is incremented by the number of new p-goals adopted in this process.

Let us go back to our example. Suppose that the agent knows that one way of always being rich is to always work hard, which in turns can be fulfilled by always being energetic. Assume that with this in mind, our agent adopts the subgoal that □WorkHard w.r.t. the p-goal that □BeRich, and then adopts the sub-subgoal that □BeEnergetic w.r.t. the subgoal that □WorkHard starting in S_0. Then the agent's goal hierarchy in $S_1 = do(adopt(\text{□WorkHard}, \text{□BeRich}), S_0)$ should include the p-goal that □WorkHard and in $S_2 = do(adopt(\text{□BeEnergetic}, \text{□WorkHard}), S_1)$ should also include the p-goal that □BeEnergetic. According to the SSA for G, our agent's goal hierarchy in S_1 and in S_2 will be as in Table 1.[6] In S_0, the supergoal □BeRich holds at level 0 and thus AddList(□BeRich, S_0) = [0]. Similarly in S_1, the supergoal □WorkHard holds at level 3 and thus AddList(□WorkHard, S_1) = [3]. Now, suppose that in S_2 the agent wants to drop the p-goal that □WorkHard. Then in $S_3 = do(drop(\text{□WorkHard}), S_2)$, she should no longer have □BeEnergetic as a p-goal, but should retain the supergoal that □BeRich. After the agent drops the p-goal that □WorkHard, by the SSA for G we can see that all the G-accessible levels where □WorkHard holds will be replaced by the only p-goal that CorrectHist(S_2, $path$) (see S_3 in Table 1). This shows that dropping □WorkHard results in the dropping of all of its subgoals (in this case □BeEnergetic), but that its parent goal □BeRich is retained.

[6] For simplicity in Table 1, we only show the agent's relevant p-goals rather than its only p-goals (which in addition reflect the actions that have been performed so far, i.e. CorrectHist(s)).

We define the SubGoal relation as follows:

$$\begin{aligned}\text{SubGoal}(\psi, \phi, s) \stackrel{\text{def}}{=} \exists n.\ \text{PGoal}(\phi, n, s) \wedge \neg\text{PGoal}(\psi, n, s)\\ \wedge \forall n.\ \text{PGoal}(\psi, n, s) \supset \text{PGoal}(\phi, n, s).\end{aligned}$$

This says that ψ is a subgoal of ϕ in situation s iff there exists an G-accessibility level n in s such that ϕ is a p-goal at n while ψ is not, and for all G-accessibility levels in s where ψ is a p-goal, ϕ is also a p-goal. Note that, while our formalization of subgoal dynamics allows a subgoal to have multiple parents, in this definition we assume that a subgoal can't have more than one parent. In the future, we will work on relaxing this constraint.

We now discuss some properties concerning the dynamics of subgoals and the dependencies between a subgoal and its parent goal. Proposition 7 states that (a) an agent acquires the p-goal that ψ after she adopts it as a subgoal of another goal ϕ in s, provided that she has the p-goal at some level in s that ϕ, and (b) she also acquires the primary c-goal that ψ after she adopts it as a subgoal of ϕ in s, provided that she has the primary c-goal in s that ϕ, and that she does not have the c-goal in s that $\neg\psi$ next.

Proposition 7 (Subgoal Adoption)

(a) $D \models \exists m.\ \text{PGoal}(\phi, m, s) \supset \exists n.\ \text{PGoal}(\psi, n, do(adopt(\psi, \phi), s))$,

(b) $D \models \text{PrimCGoal}(\phi, s)$

$$\begin{aligned}&\wedge \neg\text{CGoal}(\neg\exists s', p'.\ \text{Starts}(s') \wedge \text{Suffix}(p', do(adopt(\psi, \phi), s')) \wedge \psi(p'), s)\\ &\supset \text{PrimCGoal}(\psi, do(adopt(\psi, \phi), s)).\end{aligned}$$

The next property says that after dropping the p-goal that ϕ in s, an agent does not have the p-goal (and thus the primary c-goal) that the progression of ψ, provided that ψ is a subgoal of ϕ in s, and that the progression of ψ is not inevitable in $do(drop(\phi), s)$.

Proposition 8 (Supergoal Drop)

$$\begin{aligned}&D \models \text{SubGoal}(\psi, \phi, s) \wedge \neg\text{Inevitable}(\text{ProgressionOf}(\psi, drop(\phi), s), do(drop(\phi), s))\\ &\quad \supset \neg\exists n.\ \text{PGoal}(\text{ProgressionOf}(\psi, drop(\phi), s), n, do(drop(\phi), s)).\end{aligned}$$

As with Proposition 4, this does not hold if we replace PGoal in the consequence with CGoal since ψ could be a consequence of a combination of other active p-goals.

The next two properties say that dropping a subgoal does not effect the parent goal.

Proposition 9 (Subgoal Drop)

(a) $D \models \text{SubGoal}(\psi, \phi, s)$

$$\supset \exists n.\ \text{PGoal}(\text{ProgressionOf}(\phi, drop(\psi), s), n, do(drop(\psi), s)),$$

(b) $D \models \text{SubGoal}(\psi, \phi, s) \wedge \text{PrimCGoal}(\phi, s)$

$$\supset \text{PrimCGoal}(\text{ProgressionOf}(\phi, drop(\psi), s), do(drop(\psi), s)).$$

That is, (a) an agent retains the p-goal that the progression of ϕ after she drops a subgoal ψ of ϕ, and (b) she also retains the primary c-goal that the progression of ϕ after she drops a subgoal ψ of ϕ in s, provided that she has the primary c-goal that ϕ in s.

Finally, it can be shown that the SubGoal relation is transitive, i.e. if ψ_1 is a subgoal of ϕ in s, and if ψ_2 is a subgoal of ψ_1 in s, then ψ_2 must also be a subgoal of ϕ in s.

7 Discussion and Future Work

In this paper, we presented a formalization of prioritized goals, subgoals, and their dynamics. Our formalization ensures that an agent's chosen goals are always consistent and that goals and subgoals properly evolve as a result of regular actions as well as of adopting and dropping goals. Although we made some simplifying assumptions, in this paper we have focused on developing an expressive framework that captures an idealized form of rationality without worrying about tractability. In would be desirable to study restricted fragments of the logic where reasoning is tractable. Also, before defining more limited forms of rationality, one should have a clear specification of what ideal rationality really is so that one understands what compromises are being made.

While in our account chosen goals are closed under logical consequence, primary c-goals are not. Thus, our formalization of primary c-goals is related to the non-normal modal formalizations of intentions found in the literature [3], and as such it does not suffer from the side-effect problem [1]. For instance, in our framework an agent can have the primary c-goal to get her teeth fixed and know that this always involves pain, but not have the primary c-goal to have pain.

Also, since we are using the situation calculus, we can easily represent procedural goals/plans, e.g. the goal to do a_1 and then a_2 can be written as: PGoal($\exists s, s_1$. Starts(s_1) $\wedge$ OnPath(s) $\wedge$ $s = do(a_2, do(a_1, s_1)), 0, S_0$). Golog [12] can be used to represent complex plans/programs. So we can model the adoption of plans as subgoals.

Recently, there have been a few proposals that deal with goal change. Shapiro *et al.* [18] present a situation calculus based framework where an agent adopts a goal when she is requested to do so, and remains committed to this goal unless the requester cancels this request; a goal is retained even if the agent learns that it has become impossible, and in this case the agent's goals become inconsistent. Shapiro and Brewka [7] modify this framework to ensure that goals are dropped when they are believed to be impossible or when they are achieved. Their account is similar to ours in the sense that they also assume a priority ordering over the set of (in their case, requested) goals, and in every situation they compute chosen goals by computing a maximal consistent goal set that is also compatible with the agent's beliefs. However, their model has some unintuitive properties: the agent's chosen goals in $do(a, s)$ may be quite different from her chosen goals in s, although a did not make any of her goals in s impossible or inconsistent with higher priority goals, because inconsistencies between goals at the same priority level are resolved differently. In their framework, this can happen because goals are only partially ordered. Note that, while one might argue that a partial order over goals might be more general, allowing this means that additional control information is required to obtain a single goal state after the agent's goals change. Also, we provide a more expressive formalization of prioritized goals – we model goals using infinite paths, and thus can model many types of goals that they cannot. Finally they model prioritized goals by treating the agent's p-goals as an arbitrary set of temporal formulae, and then defining the set of c-goals as a subset of the p-goals. But our possible world semantics has some advantages over this: it clearly defines when goals are consistent with each other and with what is known. One can easily specify how goals change when an action a occurs, e.g. the goal to do a next and then do b becomes the goal to do b next, the goal that $\Diamond\Phi \vee \Diamond\Psi$ becomes the goal that $\Diamond\Psi$ if a makes achieving Φ impossible, etc.

There has been much work on agent programming languages with declarative goals where the dynamics of goals and intentions and the dependencies between goals and subgoals are modeled (e.g. [19,20,9] and the references therein). However, most of these are not based on a formal theory of agency, and to the best of our knowledge, none maintains the consistency of (chosen) goals (e.g. when adopting a plan to achieve a goal, these frameworks do not ensure that this plan is consistent with the agent's other concurrent goals/plans). Also, most of these do not deal with temporally extended goals, and as a result they often need to accommodate inconsistent goal-bases to allow the agent to achieve conflicting states at different time points (e.g. the default logic based framework in [21]); chosen goals are required to be consistent. In [22], the authors formalized two semantics for representing conflicting goals, using propositional and default logic; they argued that even logically consistent goals can be conflicting, e.g. when multiple goals/plans are chosen to fulfill the same (super)goal. Unlike us however, they do not address how an agent chooses the goals that she will actively pursue. In [6], the authors present a situation calculus based agent programming language where the agent executes a program while maximizing the achievement of a set of prioritized goals. However, they do not formalize goal dynamics.

One limitation of our account is that we assume that the agent's p-goals are totally ordered in terms of priority. Also, newly adopted p-goals are assigned the lowest priority. A consequence of this is that an agent's c-goals depend on the adoption order of her p-goals. For instance, given a fixed starting situation, an agent can end up with two different sets of c-goals by adopting ϕ followed by ψ, and by adopting ψ followed by ϕ. This has very different results when ϕ and ψ conflict with each other. We would like to address this by incorporating the priority of the p-goal as an argument to the *adopt* action, and handling this in the framework. Finally, one could argue that our agent wastes resources trying to optimize her c-goals at every step. In the future, we would like to develop an account where the agent is strongly committed to her chosen goals, and where the filter override mechanism is only triggered under specific conditions.

References

1. Cohen, P.R., Levesque, H.J.: Intention is Choice with Commitment. Artificial Intelligence 42(2-3), 213–361 (1990)
2. Sadek, M.D.: A Study in the Logic of Intention. In: Third Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR&R 1992), Cambridge, MA, pp. 462–473 (1992)
3. Konolige, K., Pollack, M.E.: A Representationalist Theory of Intention. In: Thirteenth Intl. J. Conf. on Artificial Intelligence (IJCAI 1993), Chambéry, France, pp. 390–395 (1993)
4. Singh, M.P.: Multiagent Systems – A Theoretical Framework for Intentions, Know-How, and Communications. LNCS (LNAI), vol. 799. Springer, Heidelberg (1994)
5. Shapiro, S., Lespérance, Y., Levesque, H.J.: Goals and Rational Action in the Situation Calculus - A Preliminary Report. In: Working Notes of the AAAI Fall Symposium on Rational Agency: Concepts, Theories, Models, and Applications, Cambridge, MA, November 1995, pp. 117–122 (1995)
6. Sardina, S., Shapiro, S.: Rational Action in Agent Programs with Prioritized Goals. In: Second Intl. J. Conf. on Autonomous Agents and Multi-Agent Sys (AAMAS 2003), Melbourne, Australia, pp. 417–424 (2003)
7. Shapiro, S., Brewka, G.: Dynamic Interactions Between Goals and Beliefs. In: Twentieth Intl. J. Conf. on Artificial Intelligence (IJCAI 2007), India, pp. 2625–2630 (2007)

8. Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative and Procedural Goals in Intelligent Agent Systems. In: Eighth Intl. Conf. on Principles and Knowledge Representation and Reasoning (KR&R 2002), Toulouse, France, pp. 470–481 (2002)
9. Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E. (eds.): Multi-Agent Programming: Languages, Platforms and Applications. Springer, Heidelberg (2005)
10. Bratman, M.E.: Intentions, Plans, and Practical Reason. Harvard University Press, Cambridge (1987)
11. McCarthy, J., Hayes, P.J.: Some Philosophical Problems from the Standpoint of Artificial Intelligence. Machine Intelligence 4, 463–502 (1969)
12. Reiter, R.: Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press, Cambridge (2001)
13. DeGiacomo, G., Lespérance, Y., Levesque, H.J.: ConGolog, a Concurrent Programming Language Based on the Situation Calculus. Artificial Intelligence 121, 109–169 (2000)
14. Levesque, H.J., Pirri, F., Reiter, R.: Foundations for a Calculus of Situations. Electronic Transactions of AI (ETAI) 2(3-4), 159–178 (1998)
15. Moore, R.C.: A Formal Theory of Knowledge and Action. In: Hobbs, J.R., Moore, R.C. (eds.) Formal Theories of the Commonsense World, pp. 319–358. Ablex, Greenwich (1985)
16. Scherl, R., Levesque, H.: Knowledge, Action, and the Frame Problem. Artificial Intelligence 144(1-2), 1–39 (2003)
17. Rao, A.S., Georgeff, M.P.: Modeling Rational Agents with a BDI-Architecture. In: Fikes, R., Sandewall, E. (eds.) Second Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR&R 1991), San Mateo, CA, pp. 473–484. Morgan Kaufmann Publishers, San Francisco (1991)
18. Shapiro, S., Lespérance, Y., Levesque, H.J.: Goal Change in the Situation Calculus. J. of Logic and Computation 17(5), 983–1018 (2007)
19. Sardina, S.: deSilva, L., Padgham, L.: Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach. In: Fifth Intl. J. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 2006), Hakodate, Japan, pp. 1001–1008 (2006)
20. van Riemsdijk, M.B., Dastani, M., Dignum, F., Meyer, J.J.C.: Dynamics of Declarative Goals in Agent Programming. In: Leite, J., Omicini, A., Torroni, P., Yolum, p. (eds.) DALT 2004. LNCS (LNAI), vol. 3476, pp. 1–18. Springer, Heidelberg (2005)
21. van Riemsdijk, M.B., Dastani, M., Meyer, J.J.: Ch.: Semantics of Declarative Goals in Agent Programming. In: Fourth Int'l J. Conf. on Autonomous Agents and Multiagent Sys. (AAMAS 2005), pp. 133–140 (2005)
22. van Riemsdijk, M.B., Dastani, M., Meyer, J.J.Ch.: Goals in Conflict: Semantic Foundations of Goals in Agent Programming. International Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) 18(3), 471–500 (2009)

Declarative and Numerical Analysis of Edge Creation Process in Trust-Based Social Networks

Babak Khosravifar[1], Jamal Bentahar[2], and Maziar Gomrokchi[3]

[1] Department of Electrical and Computer Engineering, [2] Concordia Institute for Information System Engineering, [3] Department of Computer Science, Concordia University, Montreal, Canada
b_khosr@encs.concordia.ca, bentahar@ciise.concordia.ca, m_gomrok@encs.concordia.ca

Abstract. Online social networks are enjoying drastic increase in their population and connectivity. One of the fundamental issues in these networks is trust, which is an essential factor in quality of the connections among diverse nodes in the network. To address the efficiency in the interactions among nodes, we propose in this paper a trust-based architecture applicable to maintain interactions in multi-agent-based social networks. We provide a detailed discussion over the network formation by taking into account the edge creation factors classified as homophily, confounding and influence. We systematically inspire different involving factors to observe evolution of trust-based interconnections in a microscopic manner. We also provide declarative and numerical analysis of the proposed model and its assessment and discuss the system implementation, along with simulations obtained from a number of executions compared with the broadly known frameworks.

Keywords: Trust establishment, edge creation, agent communication, social networks.

1 Introduction

During the last ten years, online social networks have been drastically enlarged. Facebook, Flicker, Yahoo! Answers are among very popular social networks that are gaining a very high traffic in terms of the users and their connectivity. In general, the impact of the features of these networks and analysis on how they form the behavior of the users have been of a great interest during the very recent years. A number of theoretical and empirical works have been proposed analyzing the users' behavior in forming the connection among them. For example, the analysis on the edge creation process between network nodes (participants), which is related to the sociality of a node, led to observe the distribution of a heavy-traffic degree of popular nodes [6,10]. In [1], the authors address the source of the correlation among agents that led them to extend their activity and create edges. In [2], the correlation between agents are analyzed in an online large scale network. In fact, the relation among the agents that just joined the

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 137–157, 2010.

network and the agents that are already in the network is discussed. Clearly, all these proposals investigate the way the networks are formed and enlarged. However, the reasons behind the edge creations and extensions are not specifically discussed.

In this paper, we analyze the issue of edge creation from a different perspective over the social correlation among agents using a combination of declarative and numerical techniques. In fact, trust is the main issue agents consider when decide to create edges connecting them within the network. In edge creation process, we analyze diverse impacts upon the trust that is already established between two nodes and generalize the trust concept to analyze the socialization of agents that use different trust evaluation systems. There are some proposals in the literature developing frameworks to establish trust among agents. The purpose of this paper is not to develop a new trust framework, but to analyze how the trust can affect the status of an agent in a social network, which is captured by a set of decision rules in the framework. We first discuss the trust evaluation method and upon that, analyze the connectivity among agents in a microscopic approach. Some trust models in the literature consider the direct interaction of two parties [8,15,16]. Some models also rely, to some extent, on the suggested ratings provided by other agents [9,12,14]; and some others also consider the suggested ratings of the agent being evaluated [4,8]. Since agents are self-interested, it is hard to analyze an agent's likely behavior based on previous direct interactions given the fact that the collected information from other agents may be non-reliable and could lead to a non-accurate trust assessment. So far, these frameworks do not act properly if selfish agents tend to change their behaviors. Therefore, agents do not properly initiate a social activity in the sense that they cannot maintain a strong control on the environment.

In this paper, we use the model we proposed in [11] and discuss the social network related parameters (classified in [1]) such as homophily, confounding and influence on the edge creation process of the distributed agents. Homophily refers to the tendency for agents to have ties with agent who are similar to themselves. Confounding refers to the external influence where external factors correlate with the event that two agents become connected. For example, "two friends are likely to live in the same city, and therefore to post pictures of the same landmarks in an online photo sharing system" [1]. Influence is a parameter for the extent to which an agent is prompted to initiate a connection with another agent caused by an adjacent agent.

The objective of discussing these parameters is to elaborate the impacts that a proper trust adjustment framework has on the extension of agents' connectivity. Considering the cost that agents pay for edge creation, the accuracy on suitable extensions are crucial. In the proposed model, we provide an efficient assessment process in a twofold contribution. In the first contribution, agents evaluate the trust of other agents by combining their direct and indirect trust. In direct trust, the history of interactions is considered as a measure of honesty. In indirect trust, the suggested rates through some consulting agents are

considered in the measurement of trust. In the second contribution, the evaluator agent updates its belief set considering the difference/similarity between the proposed rates and the actual trustworthiness of the evaluated agent captured by its actual behavior. The update (so-called maintenance) considers the evaluated agent together with the consulting agents that have been suggested in support of the agent being evaluated. By updating the trust values, agents would recognize the adjacent agents that are worth to extend the connection with. On the other hand, agents would stay far from bad agents in terms of social connectivity. Doing so, the agents equipped with the maintenance framework, gradually recognize the more reliable interacting agents and thus, can quickly propagate the recent changes in the environment. The edge creations are done towards the active and accurate communities. Therefore, more efficient social correlation is formed among interacting agents.

The remainder of this paper is organized as follows. In Section 2, we briefly define our proposed framework as comprehensive trust assessment process, which is composed of evaluation and maintenance process. In Section 3, we define the social network parameters and environment where the interactions are taking place. In Section 4, we analyze and discuss these parameters in an experimental setting. We represent the testbed and compare our model results with two well-known trust models in terms of efficiency in trust assessment. Finally, Section 5 concludes the paper.

2 Trust Evaluation Approach

2.1 General Background

In this section, we combine declarative and numerical techniques to formalize trust and its assessment between interacting parties in the social network. As illustrative example, we consider a social network of customers and providers of some services used for service selection. In this paper, we focus on the trust assessment formulation and predefined rules that direct the edge extensions of agents interacting in the social network. Details about the used trust model are provided in [11]. To characterize the relationship between a trustor agent Ag_a (e.g. a customer) and a trustee agent Ag_b (e.g. a provider), three elements are used [3]: 1) how much the trustor agent trusts the trustee: $Tr_{Ag_a}^{Ag_b}$; 2) the number of past interactions: $NI_{Ag_a}^{Ag_b}$ or business transactions: $NT_{Ag_a}^{Ag_b}$; and 3) the time recency of the last transactions: $TiR_{Ag_a}^{Ag_b}$. Formally, we define a social network for service selection as follows:

Definition 1 (Social Network). A social network SN for service selection is a tuple $\langle C, P, \longrightarrow_{tc}, \longrightarrow_{tp} \rangle$ where C is a set of customers, P is a set of providers, $\longrightarrow_{tc} \subseteq C \times \mathbb{R}^3 \times C$ is a ternary relation (for labelled edges linking customers) and $\longrightarrow_{tp} \subseteq C \times \mathbb{R}^3 \times P$ is a ternary relation (for labelled edges linking customers to providers).

To simplify the notation for the labelled edges, if $c_i, c_j \in C$ and $v \in \mathbb{R}^3$, then $(c_i, v, c_j) \in \longrightarrow_{cc}$ is written as $R_{cc}(c_i, v, c_j)$. Likewise, we write $R_{cp}(c_i, v, p_k)$ instead of $(c_i, v, p_k) \in \longrightarrow_{cp}$ where $p_k \in P$. Our social network for service selection has two types of nodes: type 1 for customers and type 2 for providers and two types of edges: type 1 for edges between customers and type 2 for edges linking customers to providers. The edges of type 1 represent friendship relations in the network, while edges of type 2 capture business relationships. The existence of an edge of type 1 $R_{cc}(c_i, v, c_j)$ means that c_i knows (is friend of) c_j such that: $v = (Tr_{c_i}^{c_j}, NI_{c_i}^{c_j}, TiR_{c_i}^{c_j})$. The existence of an edge of type 2 $R_{cp}(c_i, v, p_k)$ means that c_i had transactions with p_k such that: $v = (Tr_{c_i}^{p_k}, NT_{c_i}^{p_k}, TiR_{c_i}^{p_k})$.

In general, each customer agent c_i is linked to a set of customers it knows and a set of providers it has interacted with in the past. A link (an edge) between two customers c_i and c_j is added to the social network when the number of interactions between them is large enough. In the same way, a link (an edge) between a customer c_i and a provider p_k is added to the social network when the number of transactions between them is large enough. The following two Prolog-like rules (having the form: $Head \leftarrow Body$: if $Body$ then $Head$) are used by c_i as decision rules to decide about adding links to the social networks where μ_1 and μ_2 are two predefined thresholds:

$$R_{cc}(c_i, v, c_j) \leftarrow v = (Tr_{c_i}^{c_j}, NI_{c_i}^{c_j}, TiR_{c_i}^{c_j}) \wedge NI_{c_i}^{c_j} > \mu_1$$

$$R_{cp}(c_i, v, p_k) \leftarrow v = (Tr_{c_i}^{p_k}, NT_{c_i}^{p_k}, TiR_{c_i}^{p_k}) \wedge NT_{c_i}^{p_k} > \mu_2$$

We note that there is no edges in this social network between providers. This does not mean that there is no social link between providers, but only the existing links (which could be collaborations or competitions) are not used in our framework. In fact, links between providers could be used to share information regarding clients trust. However, sharing such market information in a competitive setting requires incentives and other considerations such as coalition formation. Game theory and mechanism design tools could be used to analyze these considerations. However, this aspect is out of the scope of this paper.

A social link between two customers c_i and c_j (denoted by $SL(c_i, c_j)$) exists either because there is a link (an edge) between c_i and c_j ($R_{cc}(c_i, v, c_j)$) or because c_i is linked via an edge to another customer c_x, which is socially linked to c_j via a social link $SL(c_x, c_j)$. This aspect is specified using the following Prolog-like recursive rules:

$$SL(c_i, c_j) \leftarrow R_{cc}(c_i, v, c_j)$$

$$SL(c_i, c_j) \leftarrow R_{cc}(c_i, v', c_x) \wedge SL(c_x, c_j)$$

In the same way, we specify the social link between a customer c_i and a provider p_k as follows:

$$SL(c_i, p_k) \leftarrow R_{cp}(c_i, v, p_k)$$

$$SL(c_i, p_k) \leftarrow R_{cc}(c_i, v', c_x) \wedge SL(c_x, p_k)$$

2.2 Direct Trust (DTr)

Let $\mathcal{T}ran_{c_i}^{p_k}$ be the set of transactions between a customer c_i and a provider p_k. The direct evaluation of p_k by c_i is based on the ratings c_i gave to p_k for each past transaction ($r_l \in \mathcal{T}ran_{c_i}^{p_k}$) combined with the importance of that interaction (λ_l) and its time recency. Let n be the number of total transactions between c_i and p_k ($n = NT_{c_i}^{p_k} = |\mathcal{T}ran_{c_i}^{p_k}|$), equation 1 gives the formula to compute this evaluation.

$$DTr_{c_i}^{p_k} = \frac{\sum_{l=1}^{n}(\lambda_l.TiR_{c_i}^{p_k}.r_l)}{\sum_{l=1}^{n}(\lambda_l.TiR_{c_i}^{p_k})} \tag{1}$$

The direct interaction is considered reliable if the history of transactions is strong enough ($n > \mu_2$). If not, the evaluation should be done through consulting with some other agents. We refer to this evaluation as indirect evaluation $ITR_{c_i}^{p_k}$. In fact, if the time recency of the transactions is less than a predefined threshold μ_3, the transactions reflect old behaviors and thus may not reveal the accurate information. Likewise, if the number of transactions is not high enough to reflect a strong history, the evaluating agent could not rely on that. The following rule is a decision rule for c_i specifying the pre-conditions of using indirect trust denoted by $Use(ITR_{c_i}^{p_k})$:

$$Use(ITR_{c_i}^{p_k}) \leftarrow \neg R_{cp}(c_i, v, p_k) \vee TiR_{c_i}^{p_k} < \mu_3$$

2.3 Indirect Trust (ITr)

To perform the indirect evaluation, the customer c_i solicits information about the provider p_k from other customers, called consulting customers (denoted by the set $\mathcal{T}_{c_i}$), such that for all $c_j \in \mathcal{T}_{c_i}$ there is an edge $R_{cc}(c_i, v, c_j)$ in the social network. An agent c_j is added to $\mathcal{T}_{c_i}$ if the size of this set is less than a maximum size μ_4 and the overall trust value of c_j ($\alpha Tr_{c_i}^{c_j}$) is greater than a threshold μ_5 where $\alpha Tr_{c_i}^{c_j} = DTr_{c_i}^{c_j}.NI_{c_i}^{c_j}.TiR_{c_i}^{c_j}$. Thus, agents are added in $\mathcal{T}_{c_i}$ in the sense that the evaluating agent can rely on their provided information in support of the provider that is being evaluated. The following is the decision rule used to update $\mathcal{T}_{c_i}$:

$$\mathcal{T}_{c_i} = \mathcal{T}_{c_i} \cup \{c_j\} \leftarrow |\mathcal{T}_{c_i}| < \mu_4 \wedge \alpha Tr_{c_i}^{c_j} > \mu_5$$

To communicate and exchange trust information, agents use messages defined as follows:

Definition 2. A communication message is a tuple $\langle \alpha, \beta, c_i, c_j, M, t \rangle$, where α ($\alpha \in \{Req, Rep\}$) indicates whether it is a request or a reply communication message, β ($\beta \in \{Inf, Refuse, Not\ Have\}$) represents the type of the message as requesting information in case of initiating the communication (Inf), refusing to reveal information ($Refuse$), or not having the information in case of replying to a request message ($Not\ Have$). Agents c_i and c_j are respectively the sender and receiver of the message, M is the content of the message and finally t is the time at which the message is sent.

In order to obtain a trust value of a provider p_k ($Trust(p_k)$), the evaluator agent c_i initiates a communication message to get the information from the consulting agents. The reply from a consulting agent c_j consists of providing the trust value ($Inf(p_k)$), or informing that it does not have the required information($NotHave$), or refusing to answer($Refuse$). Formally, these possibilities are represented by the following rules:

$$\langle Rep, Inf, c_j, c_i, Inf(p_k), t_1\rangle \leftarrow \langle Req, Inf, c_i, c_j, Trust(p_k), t_0\rangle$$

$$\langle Rep, NotHave, c_j, c_i, *, t_1\rangle \leftarrow \langle Req, Inf, c_i, c_j, Trust(p_k), t_0\rangle$$

$$\langle Rep, Refuse, c_j, c_i, *, t_1\rangle \leftarrow \langle Req, Inf, c_i, c_j, Trust(p_k), t_0\rangle$$

Proposition 1. *Assume c_j is collaborating with c_i. c_j will reply by $Inf(p_k)$ using the first rule iff $SL(c_j, p_k)$.*

The consulting agents are also subject to check for their trust values, and indeed, the more the consulting agent is trustworthy, the more the evaluating agent can rely on the provided rating. The equation computing the indirect estimation is given by equation 2.

$$ITr_{c_i}^{p_k} = \frac{\sum_{c_j \in \mathcal{T}_{c_i}} \alpha Tr_{c_i}^{c_j}.DTr_{c_j}^{p_k}.TiR_{c_j}^{p_k}.NT_{c_j}^{p_k}}{\sum_{c_j \in \mathcal{T}_{c_i}} \alpha Tr_{c_i}^{c_j}.TiR_{c_j}^{p_k}.NT_{c_j}^{p_k}} \tag{2}$$

2.4 Total Trust (Tr)

To compute $Tr_{c_i}^{p_k}$, the direct and indirect evaluations are combined according to their proportional importance. The idea is that the customer relies, to some extent, on its own history (direct trust evaluation) and on consulting with its network (indirect trust evaluation). This merging method considers the proportional relevance of each trust assessment, rather than treating them separately. To this end, c_i assigns a contribution value for the trust assessment method (ω for direct trust evaluation and $1-\omega$ for indirect trust evaluation when $\omega < 1$). The value ω is obtained from equation 3.

$$\omega = \frac{\log(DTr_{c_i}^{p_k}.NT_{c_i}^{p_k}.TiR_{c_i}^{p_k})}{\sum_{c_j \in \mathcal{T}_{c_i}} \log(DTr_{c_i}^{c_j}.NI_{c_i}^{c_j}.TiR_{c_i}^{c_k})} \tag{3}$$

This value could exceed 1 in the case that the history is more informative than contribution of others. Basically, the contribution of each approach in the evaluation of p_k is defined regarding to: (1) how informative the history is in terms of the number of direct transactions between c_i and p_k ($NT_{c_i}^{p_k}$) and their time recency ($TiR_{c_i}^{p_k}$); and (2) how informative and reliable the consulting customers are from c_i's point of view ($DTr_{c_i}^{c_j}$). Therefore, consultation with other agents is less considered if the history represents a comparatively higher value for ω,

which reflects lower uncertainty. Respecting the contribution percentage of the trust assessments, c_i computes the trust value for p_k using the following rules:

$$Tr_{c_i}^{p_k} = \omega.DTr_{c_i}^{p_k} + (1-\omega).ITr_{c_i}^{p_k} \leftarrow \omega < 1$$

$$Tr_{c_i}^{p_k} = DTr_{c_i}^{p_k} \leftarrow \omega \geq 1$$

Generally, the merging method is used to obtain the most accurate trust assessment. According to the following rule, the customer agent c_i would initiate the transaction $Transact(c_i, p_k)$ with p_k if the evaluated trust is high enough (threshold μ_6), which means that the customer agent can expect a high quality of service.

$$Transact(c_i, p_k) \leftarrow Tr_{c_i}^{p_k} > \mu_6$$

2.5 Maintenance

After performing the transaction, customer c_i analyzes the quality of the received service regarding to what is expected (i.e. the evaluated trust $Tr_{c_i}^{p_k}$) and what is actually performed (so-called observed trust value $\widehat{Tr}_{c_i}^{p_k}$). To this end, an adjustment trust evaluation should be performed. When c_i decides, based on the previous rule, to transact with p_k, the number of transactions is incremented. Then the observed value is checked with the expected trust value. The corresponding update is applied on the assessed trust value depending on the difference between the observed and the evaluated values. The following rules specify the maintenance process where μ_7 is a predefined threshold and the value β is a small value in the sense that $1+\beta$ reflects an increase and $1-\beta$ reflects a decrease in the current value.

$$NT_{c_i}^{p_k} = NT_{c_i}^{p_k} + 1 \leftarrow Transact(c_i, p_k)$$

$$Tr_{c_i}^{p_k} = Tr_{c_i}^{p_k} \times (1+\beta) \leftarrow |\widehat{Tr}_{c_i}^{p_k} - Tr_{c_i}^{p_k}| < \mu_7$$

$$Tr_{c_i}^{p_k} = Tr_{c_i}^{p_k} \times (1-\beta) \leftarrow |\widehat{Tr}_{c_i}^{p_k} - Tr_{c_i}^{p_k}| \geq \mu_7$$

In general, the idea is to learn from gained experiences in the sense that observing the actual value, the agent that performed the evaluation and consulted with couple of other agents can adjust its trust in them regarding to the accuracy of information they provided. To this end, the consulting agents that provided bad trust values, which are far from the observed one, will be removed from the list of potential witnesses in the future. Likewise, the agents that reveal accurate information would be considered more trustworthy than before and would be potentially consulted in future. The evaluating agent c_i would check for each consulting agent the suggested trust value with the observed actual value (checking the difference with the threshold μ_8), and consequently updates the corresponding values.

$$Tr_{c_i}^{c_j} = Tr_{c_i}^{c_j} \times (1+\beta) \leftarrow |\widehat{Tr}_{c_i}^{p_k} - Tr_{c_j}^{p_k}| < \mu_8$$

$$Tr_{c_i}^{c_j} = Tr_{c_i}^{c_j} \times (1-\beta) \leftarrow |\widehat{Tr}_{c_i}^{p_k} - Tr_{c_j}^{p_k}| \geq \mu_8$$

$$\mathcal{T}_{c_i} = \mathcal{T}_{c_i} \setminus \{c_j\} \leftarrow |\widehat{Tr}_{c_i}^{p_k} - Tr_{c_j}^{p_k}| \geq \mu_8$$

In addition, over the recent interactions, high quality providers are recognized and thus distributed over the adjacent agents in the network. In general, using the maintenance process (for full description of algorithms, see [11]), correlated agents could increase their rate of influence to one another, which eventually would approach to a more active social network. This can be represented by an optimization problem as shown in equation 4. The minimization problem is actually inspired by the fact that each time the maintenance is progressed, there is an actual value to compare with the previously suggested trust values. To this end, the evaluating agent would learn to minimize the error such that the upcoming maintenances would be more accurate [13].

$$\min_{c_j \in \mathcal{T}_{c_i}} |\widehat{Tr}_{c_i}^{p_k} - Tr_{c_j}^{p_k}| \tag{4}$$

3 Social Network Representation

To analyze our social network for service selection, many parameters described in the literature about social networks could be considered. A detailed list of such parameters are presented in [5]. For space limit, we consider only the following parameters and provide equations to compute them in our context of trust for service selection. Without loss of generality, we would like to measure the probability (likelihood) of edge creation between a customer and a provider agent. The focus of this paper is on the study of edge-by-edge evaluation of the social network in microscopic manner. We compare the network formation of different types of agents that are using different trust establishment method and use different strategies. Hence, we effectively analyze the impact of different trust models in socializing a particular agent that joins a network and seeks to increase its overall outcome (so-called utility). We basically distinguish between different models based on their strategies of network formation in agent arrival, edge arrival and interaction maintenance process (how after-interaction parameters affect the strategies that are used in the further actions of agents).

3.1 Outdegree

Outdegree is a parameter for the extent to which an agent in the network conveys information regarding some other agents. Outdegree value from the customer's point of view, is to what extent a customer agent knows the providers. The idea is to reflect the fact that a customer that is connected to more reliable providers has a higher outdegree than a customer linked to less reliable ones.

In other words, the outdegree value reflects the extent to which an agent tries to set up and strengthen more edges connecting it to other agents. Equation 5 computes this parameter for an agent Ag, where $\alpha Tr_{Ag}^{c_j} = Tr_{Ag}^{c_j}.NI_{Ag}^{c_j}.TiR_{Ag}^{c_j}$ and $\alpha Tr_{Ag}^{p_k} = Tr_{Ag}^{p_k}.NT_{Ag}^{p_k}.TiR_{Ag}^{p_k}$.

$$D_{out}(Ag) = \sum_{c_j \in \mathcal{T}_{Ag}} \alpha Tr_{Ag}^{c_j} + \sum_{p_k \in \mathcal{T}'_{Ag}} \alpha Tr_{Ag}^{p_k} \tag{5}$$

where $\mathcal{T}'_{Ag} = \{p_k \in P \mid \exists v \in \mathbb{R}^3 \wedge R_{cp}(Ag, v, p_k)\}$

3.2 Indegree

Indegree is a parameter for the extent to which a customer in the network receives information regarding to a particular agent from some other agents. Indegree value from the customer's point of view, is the extent that the agent is known by the close agents in the network. The idea is to reflect the fact that a customer that is connected to more reliable providers has a higher indegree than a customer linked to less reliable ones. Indegree value from a provider's point of view, is the extent that a provider agent is popular in the social network that causes higher number of requests from the customer agents. In other words, the indegree value reflects the popularity of an agent in the sense that any agent would like to increase it and thus cares not to distract it. Equation 6 computes this parameter for a generalized agent Ag, that could be a customer or a provider agent.

$$D_{in}(Ag) = \sum_{c_j \in \mathcal{S}_{Ag}} \alpha Tr_{c_j}^{Ag} \tag{6}$$

where $\mathcal{S}_{Ag} = \{c_j \in C \mid \exists v \in \mathbb{R}^3 \wedge (R_{cc}(c_j, v, Ag) \vee R_{cp}(c_j, v, Ag))\}$

3.3 Homophily

Homophily is a parameter for the extent to which a customer in the network chooses to interact with a provider that is known and is already evaluated (this concept is derived from [1]). This basically raises to strengthen the correlation of adjacent agents. In the social network, agents that are known from previous interactions may tend to request for a service, which is expected to be satisfactory. This is the affect of being friend in a network. In general, it is likely that a customer agent re-selects a particular provider agent aiming to request for a new service. Thus, provider agents normally try to provide a quality service to keep their customers. The homophily of agents in the network is a factor that is not directly compared to other choices of the customer agent, that is seeking for a service. Basically it is the matter of how well-quality the provider agent would provide the new service. This means that, the customer agent's concern is to measure the probability of gaining the expected quality in the service given the fact that the provider agent has already provided a similar service to the same customer. This possibility measurement is mainly related to the indegree

value of the provider agent in the sense that a provider with high indegree value is known to be popular, so there is less chance of disturbing its popularity by providing a not promised service quality. In Section 4, we analyze this effect in more details showing that the trust models with the after-interaction policies could lead to a more accurate friendship evaluations.

Equation 7 computes the probability of selecting a provider p_k with $D_{in}(p_k)$ as indegree value. In this equation, we do not involve the trust measurement that the customer agent c_i performs for evaluating the provider agent p_k ($Tr_{c_i}^{p_k}$). The reason is that since the customer agent c_i is already in relation with the provider p_k, then based on the previous evaluation, could decide whether it worths to select this provider again. If by any chance, the previous history does not reflect the efficiency of the provider p_k, there is no point for investigating the probability of the provider's efficiency if being selected. In equation 7, the value ω is set to be the uncertainty factor (see equation 3) of the history between the customer agent c_i and the provider agent p_k. And the value β represents the coefficient set for the system inconsistency. In the trust models with after-interaction strategies, this value is dynamically modified reflecting the system accuracy level. However, without maintenance process, the value is set initially and remains fixed.

$$p(D_{in}(p_k)) = \frac{e^{\omega \log(D_{in}(p_k)+1)+\beta}}{1 + e^{\omega \log(D_{in}(p_k)+1)+\beta}} \tag{7}$$

In general, the customer c_i would request for transaction with providers that their total trust values are high enough and their homophily probability exceeds a predefined threshold μ_9. The new rule for initiating a transaction is given as follows:

$$Transact(c_i, p_k) \leftarrow Tr_{c_i}^{p_k} > \mu_6 \wedge p(D_{in}(p_k)) > \mu_9$$

3.4 Confounding

Confounding is a parameter for the extent to which a provider as an external agent influences a customer agent to request for a particular service (this concept is derived from [1]). This influence affects some close agents in the network to set up an edge with an unknown provider under the promising conditions that the provider defines. To this end, the provider that is looking for the customers requests to interact ($Intr$) and specifies some conditions that it promises to provide ($conditions(p_k)$). The customer agent analyzes the request and thus would may accept or refuse the interaction according to the following rules:

$$\langle Rep, accept, c_i, p_k, *, t_1\rangle \leftarrow \langle Req, Intr, p_k, c_i, conditions(p_k), t_0\rangle$$

$$\langle Rep, Refuse, c_i, p_k, *, t_1\rangle \leftarrow \langle Req, Intr, p_k, c_i, conditions(p_k), t_0\rangle$$

In general, the providers that join the network, seek for the agents that are more likely to request for their services. In other words, when a provider agent

is being activated, it tries to socialize itself in the network. Thus, starting from very close customer agents, the provider agent encourages them to request for its service. To this end, the provider at the beginning acts generously in order to attract the customers and gain more popularity. Moreover, upon high quality service, the customer agents may influence their adjacent agents to request for the same service. So, the provider agent takes the outdegree value of the customer agents into account and based on the interaction span of the customer agents, it provides high quality services.

In confounding factor, the probability of activating an agent c_i with a provider agent p_k is computed in equation 8. As it is assumed that the provider p_k is unknown to the customer c_i, the customer agent would evaluate the social trustworthiness value of the provider. Given the fact that the trust measurement requires some information from the other adjacent agents, the customer agent takes the entropy value into account in order to partially consider the indirect trust value ($ITr_{c_i}^{p_k}$) and the rest for the popularity of the provider agent. Thus, the customer c_i first evaluates the provider p_k and then considers the p_k's in-degree value together with the network inconsistency level. If the information obtained for evaluating p_k is not enough, the entropy value ω would be high, so that mostly the trust evaluation part would be considered. This would normally cause to lower the overall probability of activation.

$$p(c_i, D_{in}(p_k)) = \omega \times ITr_{c_i}^{p_k} + (1-\omega) \times \frac{e^{\log(D_{in}(p_k)+1)+\beta}}{1+e^{\log(D_{in}(p_k)+1)+\beta}} \tag{8}$$

The providers whose probability is greater than a predefined threshold μ_{10} would likely transact with the customer if the two conditions set in the previous rule (Section 3.3) are satisfied. The new rule to decide about initiating a transaction is given as follows:

$$Transact(c_i, p_k) \leftarrow Tr_{c_i}^{p_k} > \mu_6 \wedge p(D_{in}(p_k)) > \mu_9 \wedge p(c_i, D_{in}(p_k)) > \mu_{10}$$

3.5 Influence

Influence is a parameter for the extent to which an agent is prompted to initiate a request caused by an adjacent agent (this concept is derived from [1]). This could take place in a friendship of agents that distribute the idea of some services to be requested. When an agent needs to request a particular service from a provider, it may have already set up an edge with that provider, so the evaluation can be done, or may need to set up a new edge upon which could obtain the service. This is the affect of getting encouraged by a friend in a network. In general, it is likely that a person does action because his friend has already done it. Thus, it is the matter of activation of a new edge, which is set up between a customer agent and the provider agent, that has already been requested for a service by the customer's adjacent agent (friend).

In the confounding factor, we mentioned that when a typical provider agent advertises its service to a couple of adjacent customer agents, it considers that some of the customers may propagate its quality of service to their adjacent

agents, which could lead to more service requests. On the other hand, the customer agent that is being prompted to take a service produced by a particular provider, needs to evaluate both the advertising adjacent agent c_j ($DTr_{c_i}^{c_j}$) and the provider itself p_k ($ITr_{c_i}^{p_k}$). Equation 9 computes the influence-based probability of activation of a customer agent c_i regarding to taking the service produced by a provider agent p_k. In this equation, ω_{c_j} is the entropy value related to the information c_i has and thus could rely on, and ω_{p_k} is the entropy value related to the information that c_i has about the provider p_k.

$$p(c_i, c_j, D_{in}(p_k)) = \omega_{c_j} \times DTr_{c_i}^{c_j} + (1 - \omega_{c_j}) \times \Theta \tag{9}$$

where

$$\Theta = \omega_{p_k} \times ITr_{c_i}^{p_k} + (1 - \omega_{p_k}) \times \frac{e^{\log(D_{in}(p_k)+1)+\beta}}{1 + e^{\log(D_{in}(p_k)+1)+\beta}}$$

Finally, the decision about initiating a transaction with a provider can be given considering the previous conditions (Section 3.4) and the fact that the influence probability is greater than a predefined threshold μ_{11}. The final rule is then specified as follows:

$$Transact(c_i, p_k) \leftarrow$$
$$Tr_{c_i}^{p_k} > \mu_6 \wedge p(D_{in}(p_k)) > \mu_9 \wedge p(c_i, D_{in}(p_k)) > \mu_{10} \wedge p(c_i, c_j, D_{in}(p_k)) > \mu_{11}$$

4 Experimental Results and Related Work

In this section, we describe the implementation of proof of concept prototype. In the implemented prototype, agents are implemented as $Jadex^{©TM}$ agents. Like in [7], the testbed environment is populated with two agent types: (1) *service provider agents*; and (2) *service consumer agents*. The simulation consists of a number of consequent Runs in which 200 agents are activated and build their private knowledge, keep interacting with one another, and enhance their overall knowledge about the environment. Depending on the agent interactions, agent may extend their connections hoping to be more socialized. However, there is always the chance of investing on wrong agents that lead to no outcome. Here, we distinguish agents by the service (or information) quality that they provide. Table 1 represents four types of the service providers we consider in our simulation: good (15% of the population), ordinary (30% of the population), bad (15% of the population) and fickle (40% of the population). The first three provide the service regarding to the assigned mean value of quality with a small range of deviation. Fickle providers are more flexible as their range of service quality covers the whole possible outcomes. Upon interaction with service providers, service consumer agents obtain utilities and consequently rate the quality of the providers (for simplicity, we assume only the consumers are interconnected to the provider agents). In the simulation environment, agents are equipped with different trust models in the sense that their edge creation policies are different. In the proposed model, we try to establish a trust mechanism where an

agent, firstly can maintain an effective trust assessment process and secondly, accurately updates its belief set, which reflects the other agents likely accuracy. In order to observe the impact of each contribution, we compare the proposed model with other trust models in two perspectives. In the first comparison, we use the agents that only perform a direct trust assessment process. We refer to this group of agents as *Direct Trust Group* (DTG). In the second comparison, we use the agents that (in addition to the direct trust assessment mechanism), perform maintenance process for evaluating the consulting agents in order to increase their information accuracy. We refer to this group of agents as *Maintenance-based Trust Group* (MTG). The reason of decomposing the proposed model to two groups is to focus on the efficiency of each model, which enables us to analyze the impact of each contribution on the accuracy of the agent in edge creation process. In order to discuss the proposed model's overall performance, we compare it with BRS [1] [9] and Travos [2] [14] trust models. These models are similar to the proposed model in the sense that they do consider other agents' suggestions while evaluating the trust of some specific agents and discard inaccurate suggestions aiming to perform most efficient edge creation. The detailed description of these models is provided in [4]. Here, we basically distinguish [10] between different models based on their strategy of network formation in agent arrival, edge arrival and interaction maintenance process (how after-interaction parameters affect the strategies that are used in the further actions of agents). In the rest of this section, we discuss the impacts of efficient parameters in the edge extension of agents and elaborate how different trust mechanisms effectively deal with these impacts.

Table 1. Simulation summarization over the obtained measurements

Service provider type	Density in the network	Utility range	Utility SD
Good	15.0%	$]+5,+10]$	1.0
Ordinary	30.0%	$]-5,+5]$	2.0
Bad	15.0%	$]-10,-5]$	2.0
Fickle	40.0%	$]-10,+10]$	–

Provider Popularity. We start the discussion by the probability of selecting the providers over their different popularity values. As we discussed earlier, the indegree value of a node reflects their popularity in the social network. Thus, we could conclude that the chance of selection for a particular service provider agent would be proportionally relevant to its indegree value (ordinary selection attitude). However, the trust evaluation method together with its distribution process would affect this probability of selection. Illustrated in figure 1, the BRS

[1] BRS trust model collects the after-interaction ratings and estimates the trust using beta distribution method. This trust model ignores the ratings from such agents that deviate the most from the majority of the ratings.

[2] Travos trust model is similar to BRS in collecting the after-interaction ratings and estimating the trust using beta distribution method. But Travos ignores the ratings from agents that provide intermittent reports in the form of suggestions.

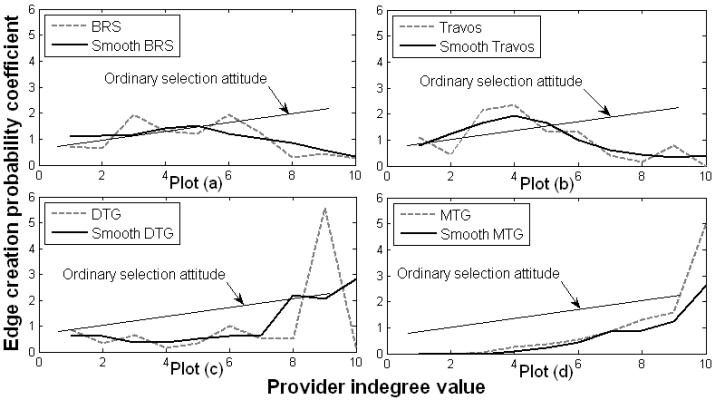

Fig. 1. Probability of edge creation with provider agent vs. the provider's indegree value

agents act independently of the mentioned probability as the BRS agents do not consider the popularity of the provider. Travos agents also do not consider such value. However, the probability of selection of the popular providers increase as they take less risk of changing their behaviors and thus perform satisfactory services, which would lead to their selection. In general, because of inaccuracy detection feature of Travos agents, the percentage of selection of provider agents with high indegree value increases in a gentle manner. At some certain point, the selection of popular providers are coming down (see *plot b*). This is explained by the fact that a popular provider has large number of recommenders that provide diverse range of information to the agent that is trying to evaluate the provider. This diversity would lead to confusion state due to high deviation of reports (the state that this system would generalize the majority of the information that is obtained and could be inaccurate), which in Travos would cause the drop of the suggestions and thus the selection would be less. The proposed model agents (DTG and MTG) follow the information propagation feature as the adjacent agents influence each other to select the high quality providers. There is a difference in the slope of selection graph in MTG and DTG models. This is explained by the fact that agents in the MTG group are characterized by the maintenance process that enable them to recognize high quality provider agents and thus their accuracy in influencing adjacent agents are more than regular DTG agents. In general, since the maintenance feature does not exist in DTG group, the customer agents loose the track of high quality provider agents, and thus the probability of selection would not increase so fast.

Interacting Agents Age. In general, in the defined testbed, the agents that are obtaining a high quality service are encouraged to distribute their experience to other adjacent agents (influence others). This activity of agents would basically get increased over the time, or say over the age of the agent. In figure 2, we have compared the activity of different groups of agents by comparing edge extension of the agents (outdegree value). Without loss of generality, the edge extension

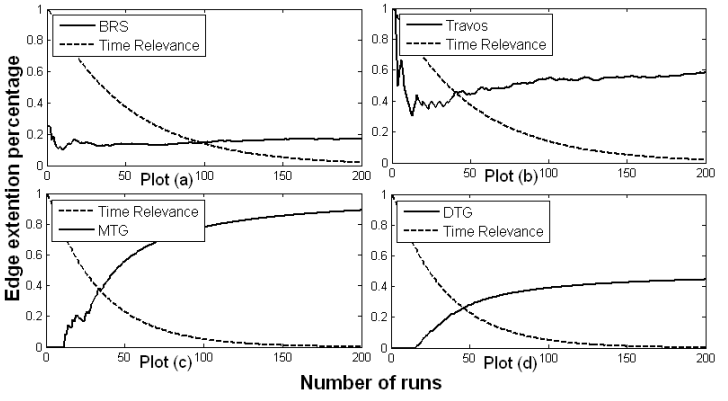

Fig. 2. Agent edge extension vs. the agents age

is proportionally related to the accuracy of agent in detecting the high quality providers. In BRS model, the extension over the time is not increasing as the agent gets involved with high number of adjacent agents and would be difficult to effectively extend the social activity, so more or less would be independent of the age of the agent. Travos and *DTG* models are increasing, however relatively with small slope. In *MTG* group, because of the maintenance process the agents would be encouraged to initiate a request to high quality service providers and thus extend their activity. In this graph, the slope is relatively large as over the time, the agent could manage to categorize the providers that could possibly act beneficially for the agent, and thus would enlarge his activity area. In figure 2, the second line represents how fast the agents would drop the previous data and use the recent data for their analysis. This dropping factor is also relevant to how active an agent is and thus, to what extent there would be available resource that agents could drop obsolete data. *DTG* and *MTG* group use the same dropping feature ($TiR(\Delta t^{Ag_b}_{Ag_a})$), which is derived in equation 10. Variable λ is an application-dependent coefficient. In some applications, recent interactions are more desirable to be considered (λ is set to relatively large number). In contrast, in some other applications, even the old interactions are still valuable source of information. In that case, a relatively smaller value to λ is used.

$$TiR(\Delta t^{Ag_b}_{Ag_a}) = e^{-\lambda \log(\Delta t^{Ag_b}_{Ag_a})} \quad \lambda \geq 0 \tag{10}$$

Homophily-Confounding-Influence. We would like to go further into the details of the selection history in terms of the microscopic social network affects (homophily, confounding, and influence) and illustrate them in figure 3. in this section, we observe the diverse impacts of homophily, confounding and influence features on each group in the sense that we would capture their edge creation reasons. Note that the edge creation is not the important issue, however, the concern is to extend to the agents that are known to be trustworthy. Therefore, we elaborate the overall outcome of different agents at the following. The

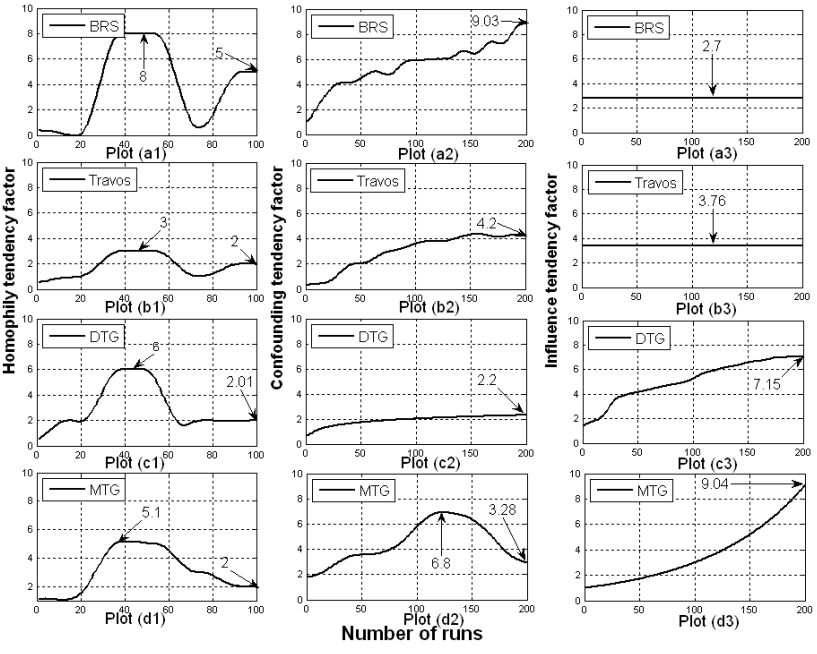

Fig. 3. Overall comparison of the proposed model with BRS and Travos in terms of (a) Homophily; (b) confounding; and (c) influence factors

homophily aspect would be caused by the friendship relation of the agents that have history interaction between them. This is a very general case in the sense that consumer agents over the time would get to know and select the provider agents. If the interacted service is satisfactory for the agent, then the consumer agent may re-select the same provider agent in some future. BRS agents are the ones that mostly rely on the homophily affect in the sense that they keep the history of the interaction in order to re-evaluate the provider agent. The providers that remain trustworthy would be selected over the time. As it is clear from plot $a1$, once the providers change their policies, the selection of them would be affected so fast, as the BRS agents recognize that they should start seeking for the appropriate friends. Travos agents also rely on the previous history and re-select the previously interacted service providers (see plot $b1$). However, over the time the reports regarding to the accuracy of the providers would be divergent, which would lead to refuse the selection. The same reason is the case for *DTG* and *MTG* group (shown in plots $c1$ and $d1$). These agents to some extent rely on the previous history and select the providers. After some certain time, these agents also recognize the inconsistency in the evaluation process of the history interacted providers. Overall, *DTG* and *MTG* agents evaluate the providers in a very accurate manner. The accuracy that Travos, *DTG* and *MTG* agents have cause the decremented manner after some certain time.

Confounding factor reflects the extent to which the provider agents advertise their service to the consumers (could be new or previously serviced ones). This feature also affects BRS group, as they start evaluating the advertising provider, and thus extend their activation area. Plot $a2$ indicates that the BRS group are easy to involve in interaction with the advertising provider agent. Travos agents act in the same way as the provider agents could induce them to take their service. However, Travos agents are considering this case less, because they investigate the previous reports related to the advertising provider and doubt on the inconsistent ones (see plot $b2$). In general, the BRS and Travos agents accept the confounding-related interactions over the time, and thus their graph has an increasing manner. But in DTG and specially MTG, the agents would not accept this service all the time, as over the time, once the network inconsistency level increases, these agents would have confusion in accepting the confounding-related affect caused by unknown service providers (see plots $c2$ and $d2$). MTG agents would accept this option from the providers, but since they are equipped with a maintenance process, they would distribute the performance of the providers to the adjacent agents, which would lead them to get to know the network faster than the other models. This would let the MTG agents to select the best providers, and thus would drop the request from most of the unknown agents while they are already in a good accuracy level.

Influence factor is mostly used by active agents, while they obtain service and tend to distribute the efficiency of the interaction to the adjacent agents. Since BRS agents independently select the providers, the influence is not a factor for these agents (plot $a3$). Treavos agents would act almost independently, however the Travos agents are encouraged by the reports they obtain for the evaluation of a particular provider agent (plot $b3$). DTG group would be encouraged with the same factor as Travos agents. Upon evaluating provides, the DTG agents would consider the reports obtained from adjacent agents and recognize outstanding service provided by the provider that is just served an adjacent agent (see plot $c3$). The influence-related interactions are mostly initiated among MTG group, shown in plot $d3$. This is explained by the fact that the MTG group are equipped with maintenance feature, which enables them to reason about the accuracy and efficiency of the obtained services and propagate the information to the adjacent information.

General Performance. Considering all the involved features, at the end we compare the models in general perspective, starting good provider selection efficiency. In such a biased environment, the number of good providers are comparatively low. Therefore, the agents need to perform an accurate trust assessment to recognize the best providers. As it is clear from the Figures 4, plots $a1$, $b1$, and $c1$, DTG agents function better than other models (Travos and BRS). The reason is that in this model, agents are assessing the credibility of the providers using other agents suggestions depending on their credibility and to what extent they know the provider. Afterwards these agents rate the provider, which would be distributed to other agents upon their request (relatively in plots $a2$, $b2$, and $c2$ the comparison of fickle selection percentage, and in $a2$, $b2$, and $c2$,

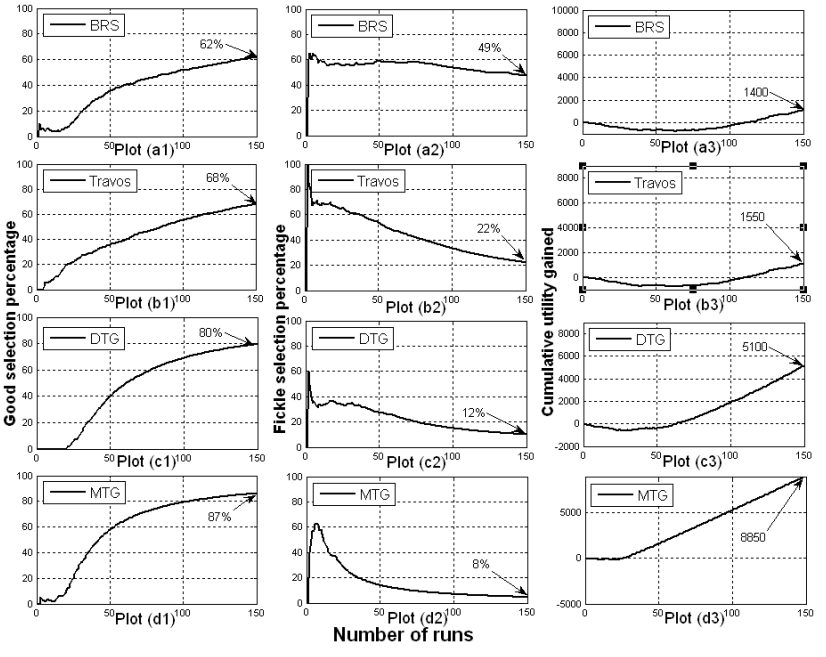

Fig. 4. Overall comparison of the proposed model with BRS and in terms of (a) good selection percentage; (b) fickle selection percentage; and (c) cumulative utility gained

the gained cumulative utility is shown). Not excluding the fact that *DTG* agents are considering partial ratings for consulting agents, we state that they weakly function when the environment contains agents that do not truthfully reveal their believes. *MTG* agents in addition to the direct trust assessment, provide incentives for consulting agents, which encourages them to effectively provide the information aiming to gain more utility. Plot $d1$ shows that *MTG* agents outperform other models in best provider selection. This is expressed by the fact that *MTG* agents recognize the best providers ensuring that the best selected provider would provide the highest utility. Relatively plot $d2$ shows an outperform in fickle selection and consequently higher cumulative utility in plot $d3$.

In BRS model, the trustor agent in the assessment process uses beta distribution method and discards the ratings that deviate the most from the majority of the ratings. Concerning this, BRS is comparatively a static trust method, which causes a low-efficient performance in very dynamic environment. In general, if a BRS agent decides to evaluate an agent that he is not acquainted with, he considers the majority of ratings, which are supposed to be truthfully revealed about the trustee agent. In such a case that the trustee agent has just changed his strategy, the trustor agent would loose in trust assessment and does not verify the accuracy of the gained information. Therefore, as illustrated in figure 4, plots $a1$, the BRS agents would have less percentage of good providers selection, relatively higher percentage of fickle providers selection (plot $a2$), and consequently lower gained cumulative utility (plot $a3$).

Travos [14] trust model is similar to BRS in using beta distribution to estimate the trust based on the previous interactions. Travos model also does not have partial rating. Hence, the trustor agent merges his own experience with suggestions from other agents. However, unlike BRS model, Travos filters the surrounding agents that are fluctuating in their reports about a specific trustee agent. To some extent, this feature would cause a partial suggestion consideration and thus, Travos agents would adapt faster comparing to BRS agents. Rates concerning the good and fickle selection percentage shown in figures 4, plots $b1$ and $b2$ reflect higher efficiency of Travos compared to BRS. However, Travos model considers that agents do not change their behavior towards the elapsing time. These missing assumptions affect the accuracy of trust estimation in a very biased environment (lower gained cumulative utility in plot $b3$).

5 Conclusion

The contribution of this paper is the detailed investigation of a trust-based multi-agent architecture in edge creation and correlation formation in social networks. The analysis of this issue is done by combining both declarative and numerical techniques. The established trust is provided by the proposed framework, that is briefly explained here. The trust assessment procedure is based on integrating suggestion of consulting agents, objectively enhancing the accuracy of agents to make use of the information communicated to them. The surveillance over the surrounding environment makes distributed agents eager to extend their activity area by interacting with high quality agents. In the proposed framework, maintenance process considers the communicated information to judge the accuracy of the consulting agents in the previous trust evaluation process. The ex-interacted analysis allows the agents to propagate the recent and accurate information to their adjacent agents, which is considered as homophily and influence factors in edge creation process.

Our model has the advantage of being computationally efficient as it takes into account the important factors involved in extending the activity zone of agents. Moreover, we have done a detailed empirical analysis over the edge creation and behavior of agents over their age, while they are equipped with different trust mechanism protocols. The proposed mechanism efficiency is compared with other related models to prove the capabilities of the proposed model. Our plan for future work is to advance the assessment model to enhance its efficiency by considering more efficient learning algorithms. In the maintenance process we need to elaborate more on the optimization part, trying to formulate it in the sense to be adaptable to diverse situations. We need to consider more extensions towards having links and correlations between provider agents and thus, we need to deal with the selfish actions that providers may perform under the assumption of having social links with other providers. Game theory and mechanism design are the most promising techniques to be investigated for such an issue. Finally, we plan to maintain more detailed analysis in comparison with other models to capture more results reflecting the proposed model capabilities.

Acknowledgements

This work is supported by Jamal Bentahar's funds from Natural Sciences and Engineering Research Council of Canada (NSERC: 341422-07), Fonds québécois de la recherche sur la nature et les technologies (FQRNT: 2008-NC-119348) and Fonds québécois de la recherche sur la société et la culture (FQRSC: 2007-111881). We would also like to thank the anonymous reviewers for their very interesting comments.

References

1. Anagnostopoulos, A., Kumar, R., Mahdian, M.: Influence and correlation in social networks. In: Proceedings of the 14'th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 7–15 (2008)
2. Backstrom, L., Huttenlocher, D., Kleinberg, J., Lan, X.: Group formation in large scale networks: membership, growth, and evolution. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 44–54 (2006)
3. Bentahar, J., Khosravifar, B., Gomrokchi, M.: Social network-based trust for agent-based services. In: Proceedings of the 23rd IEEE International Conference on Advanced Information Networking and Applications (AINA), Symposium on Web and Mobile Information Services, pp. 298–303 (2009)
4. Bentahar, J., Khosravifar, B.: Using trustworthy and referee agents to secure multi-agent systems. In: Proceedings of the 5th IEEE International Conference on Information Technology: New Generations (ITNG), pp. 477–482 (2008)
5. Buskens, V.: Social Networks and Trust. Kluwer Academic Publishers, Dordrecht (2002)
6. Christakis, N.A., Fowler, J.H.: The spread of obesity in a large social network over 32 Years. The new England Journal of Medicine 357(4), 370–379 (2007)
7. Dong-Huynh, T., Jennings, N.R., Shadbolt, N.R.: Fire: an integrated trust and reputation model for open multi-agent systems. Journal of Autonomous Agents and Multi-Agent Systems 13(2), 119–154 (2006)
8. Dong-Huynh, T., Jennings, N.R., Shadbolt, N.R.: Certified reputation: how an agent can trust a stranger. In: Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), Japan, pp. 1217–1224 (2006)
9. Jesang, A., Ismail, R.: The beta reputation system. In: 15th Bled Electronic Commerce Conference e-Reality: Constructing the e-Economy (June 2002)
10. Leskovec, J., Backstrom, L., Kumar, R., Tomkins, A.: Microscopic evolution of social networks. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 462–470 (2008)
11. Khosravifar, B., Gomrokchi, M., Bentahar, J., Thiran, Ph.: A maintenance-based trust for Open multi-agent systems. In: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 1017–1024 (2009)
12. Sabater, J., Paolucci, M., Conte, R.: Repage: REPutation and ImAGE among limited autonomous partners. Journal of Artificial Societies and Social Simulation 9(2) (2006)

13. Siddiqi, S.M., Boots, B., GGordon, G.J.: A Constraint generation approach to learning stable linear dynamical systems. In: Proccedings of Advances in Neural Information Processing Systems, NIPS (2007)
14. Teacy, W.T., Patel, J., Jennings, N.R., Luck, M.: Travos: trust and reputation in the context of inaccurate information sources. Autonomous Agentsand Multi-Agent Systems 12(2), 183–198 (2006)
15. Xiong, L., Liu, L.: PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities. Journal of IEEE Transactions on Knowledge and Data Engineering 16(7), 843–857 (2004)
16. Wang, Y., Singh, M.P.: Formal trust model for multiagent ststems. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI), pp. 1551–1556 (2007)

Computing Utility from Weighted Description Logic Preference Formulas

Azzurra Ragone[1], Tommaso Di Noia[1], Francesco M. Donini[2], Eugenio Di Sciascio[1], and Michael P. Wellman[3]

[1] SisInfLab, Politecnico di Bari, Bari, Italy
{a.ragone,t.dinoia,disciascio}@poliba.it
[2] Università della Tuscia, Viterbo, Italy
donini@unitus.it
[3] Artificial Intelligence Laboratory–University of Michigan, Ann Arbor, USA
wellman@umich.edu

Abstract. We propose a framework to compute the utility of a proposal w.r.t. a preference set in a negotiation process. In particular, we refer to preferences expressed as weighted formulas in a decidable fragment of First Order Logic (FOL). Although here we tailor our approach for Description Logics endowed with disjunction, all the results keep their validity in any decidable fragment of FOL. DLs offer expressivity advantages over propositional representations, and allow us to relax the often unrealistic assumption of additive independence among attributes. We provide suitable definitions of the problem and present algorithms to compute utility in our setting. We also study complexity issues of our approach and demonstrate its usefulness with a running example in a multiattribute negotiation scenario.

1 Introduction

Effective and expressive specification of preferences is a particularly challenging research problem in knowledge representation. Preference representation is essential, for example, to instruct a software agent to act on behalf of the objectives of a human being. One common approach to preference representation appeals to *multiattribute utility theory* [1], which concerns the construction of *utility functions* mapping vectors of *attributes* to real values. Given that the size of a multiattribute domain is exponential in the number of attributes, applications typically exploit independence relations among the attributes, in the most extreme case to assume that all attributes are additively independent, so that the multiattribute utility function is a weighted sum of single-attribute utility functions.

However, most real-world domains pose significant preferential dependencies, ruled out by the fully additive model. For example, referring to the desktop computer realm, we could not with an additive value function capture the fact that the value of some combination of attributes is not simple the sum of the single attribute values. Indeed, some operating systems perform very poorly without a minimum amount of memory, therefore the value (utility) given to a specific operating system will depend on the amount of memory available.

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 158–173, 2010.

Some recent approaches support relaxation of the fully additive assumption, for example by providing generalized versions [2] or exploiting graphical models of dependence structure [3,4,5], while remaining within the multiattribute framework.

Logical languages likewise provide a means to express interdependencies, but unlike multiattribute formulations they do not necessarily require that we explicitly decompose the domain into an orthogonal set of attributes. Furthermore, logical languages support integration of preference knowledge with domain knowledge modeled through an ontology. Using an ontology, indeed, it is possible to model relations among attributes in the domain (*e.g.*, *a Centrino is an Intel processor with a 32-bit CPU*), as well as the fact that some combination of features may be infeasible (therefore of minimal or undefined preference) due to constraints in the ontology itself (*e.g.*, *a Centrino processor is not compatible with a processor with a 64-bit architecture*).

In decision making problems, preferences are expressed over a set of possible alternatives, in order to rank them. In many cases, such as *e.g.*, bilateral negotiation, auctions, resource allocation, it is important to compute a utility value for, respectively, an agreement, an offer, an allocation w.r.t. the set of preferences expressed by the agent. If preferences are expressed using Propositional Logic, then the utility can be computed considering a particular propositional model (agreement, offer, allocation), taking into account formulas satisfied by that model.

While for Propositional Logic it is possible to refer directly to models (interpretations) in order to compute utility, this computation for First-order Logic (FOL) is less straightforward, as the number of possible models is infinite.

The main contribution of this paper is an approach that, given a set of preferences, represented as weighted DL formulas w.r.t. a shared ontology, computes the utility of a formula (agreement, offer, allocation, etc.) based on its possible models (interpretations). To our knowledge, the only prior method proposed in the literature for this problem is subsumption, which has some limitations, as we show in Section 4.

We point out that even though the results we show in this paper can be easily applied to whatever decidable logic with a model-theoretic semantics, we ground our approach on DLs because of their importance in the development of the Semantic Web.

The remainder of the paper proceeds as follows. First, we introduce Description Logics, then we give a brief overview of the problem of preference representation in the field of logic languages. In Section 4, we first introduce the problem of computing utility of a concept w.r.t. a preference set, showing how, sometime, subsumption leads to counterintuitive results. Then we analyze some complexity issues. In Section 5 we illustrate our framework for the computation of utility for a set of weighted DL formulas with the help of a running example. Finally, we discuss some considerations about the computational properties of the framework. Conclusion closes the paper.

2 Description Logic Basics

Description logics (DLs) are a family of formalisms well-established in the field of knowledge representation. Readers familiar with DLs may safely skim this section, attending mainly to the notation and examples. Those interested in learning more may refer to the *Description Logic Handbook* [6] for a much more comprehensive treatment.

The basic syntax elements of Description Logics are *concept names*, *properties*, and *individuals*. Concept names stand for sets of objects in the domain[1] (`Windows`, `Intel`, `LCDMonitor`), and properties link (sets of) such objects (`hasOS`, `hasCPU`, `hasMonitor`). Individuals correspond to special named elements belonging to concepts (`HP_Pavilion`, `Apple_iMac`). When we do not use proper names, we denote concepts by symbols $A, B, C, D, \ldots, \top, \bot$.

Description logics are usually endowed with a model-theoretic formal semantics. A semantic *interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ represents the *domain* and $\cdot^{\mathcal{I}}$ is the *interpretation function*. This function maps every concept to a subset of $\Delta^{\mathcal{I}}$, and every property to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Then, given a concept name CN and a property name R we have: $CN^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The symbols $\top$ and $\bot$ are used to represent the most generic concept and the most specific concept respectively. Hence their formal semantics correspond to $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ and $\bot^{\mathcal{I}} = \emptyset$.

Properties and concept names can be combined using *existential role quantification*. For example, $\texttt{PC} \sqcap \exists \texttt{netSupport.WiFi}$ describes the set of PCs supporting a wireless connection. Similarly, we can use *universal role quantification*, as in $\texttt{PC} \sqcap \forall \texttt{hasCPU.AMD}$, to describe the set of PCs having only AMD processors on board. The formal semantics of universal and existential quantification is as follows:

$$\exists R.C = \{x \in \Delta^{\mathcal{I}} | \exists y, (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$$
$$\forall R.C = \{x \in \Delta^{\mathcal{I}} | \forall y, (x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$$

Concept expressions can be written using *constructors* to write concept and property *expressions*. Based on the set of allowed constructors we can distinguish different description logics. Essentially every DL allows one to form a *conjunction* of concepts, usually denoted as $\sqcap$; some DLs include also disjunction $\sqcup$ and complement $\neg$ to close concept expressions under boolean operations.

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$$
$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$
$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$$

Constructs involving number restriction enable us to define concepts in terms of the numbers of roles with specified properties. For example, $\texttt{Mac} \sqcap (\geq\ 4\texttt{hasUSBport})$ describes a Macintosh PC with at least four USB ports. Some DLs integrate concrete domains into the language [7]. Formally a concrete domain $\mathcal{D}$ is a pair $\langle \Delta_D, pred(D) \rangle$ where Δ_D is the domain of $\mathcal{D}$ and $pred(D)$ is a set of predicates over Δ_D. An example of concrete domain restrictions appears in the expression $\texttt{PC} \sqcap \exists \texttt{hasRam.}(\geq_2 GB)$, which describes a PC with at least 2 GB of memory. Here the domain of $\mathcal{D}$ is represented by natural numbers while $\geq_2 \in pred(D)$ is a unary predicate for $\mathcal{D}$. Notice that while properties, such as `hasUSBport`, are mapped to a subset of of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, concrete properties, such as GB are mapped to a subset $\Delta^{\mathcal{I}} \times \Delta_D$.

[1] We illustrate the main points here (and throughout the paper) using the domain of desktop computers.

$$(\geq\ n\ R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} |\ |\{b \in \Delta^{\mathcal{I}} | (a,b) \in R^{\mathcal{I}}\}| \geq n\}$$
$$(\leq\ m\ R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} |\ |\{b \in \Delta^{\mathcal{I}} | (a,b) \in R^{\mathcal{I}}\}| \leq m\}$$
$$(\leq_k\ R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | R^{\mathcal{I}}(x) \leq k\}$$
$$(\geq_h\ R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | R^{\mathcal{I}}(x) \geq h\}$$

In general, the expressiveness of a DL depends on the type of constructors allowed.

Given a generic concept C, we use the notation $\mathcal{I} \models C$ to say that $C^{\mathcal{I}} \neq \emptyset$.

In order to formally represent domain knowledge and constraints operating among elements of the domain, we employ a set of background axioms, that is, an *ontology*. Formally, ontology $\mathcal{T}$ (for Terminology) comprises axioms of the form $D \sqsubseteq C$, where D and C are well-formed formulas in the adopted DL, and $R \sqsubseteq S$, where both R and S are properties. $C \equiv D$ is a syntactic sugor for both $C \sqsubseteq D$ and $D \sqsubseteq C$. The formal semantics of such axioms is: $(C \sqsubseteq D)^{\mathcal{I}} = C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, $(R \sqsubseteq S)^{\mathcal{I}} = R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$.

We write $\mathcal{I} \models \mathcal{T}$ to denote that for each axiom $C \sqsubseteq D$ in $\mathcal{T}$ it results $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Similarly $\mathcal{I} \models C \sqsubseteq_{\mathcal{T}} D$, with $C \sqsubseteq D \notin \mathcal{T}$, denotes that both $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models C \sqsubseteq D$.

In the rest of the paper we refer to the Ontology $\mathcal{T}$ depicted in Figure 1.

$\texttt{DesktopComputer} \sqsubseteq \exists \texttt{hasCPU} \sqcap \exists \texttt{hasRam}$

$\texttt{CPUArchitecture} \equiv \texttt{64BitCPU} \sqcup \texttt{32BitCPU}$

$\texttt{32BitCPU} \sqsubseteq \neg \texttt{64BitCPU}$

$\texttt{Intel} \sqcup \texttt{AMD} \sqsubseteq \texttt{CPU}$

$\texttt{Intel} \sqsubseteq \neg \texttt{AMD}$

$\texttt{hasUSBport} \sqsubseteq \texttt{hasPeripheralPort}$

$\texttt{Athlon64} \sqsubseteq \texttt{AMD} \sqcap \exists \texttt{arch} \sqcap \forall \texttt{arch.64BitCPU}$

$\texttt{Centrino} \sqsubseteq \texttt{Intel} \sqcap \exists \texttt{arch} \sqcap \forall \texttt{arch.32BitCPU}$

$\exists \texttt{hasCPU.Centrino} \sqsubseteq \exists \texttt{netSupport.WiFi}$

$\texttt{IntelDuo} \sqsubseteq \texttt{Intel} \sqcap \exists \texttt{arch} \sqcap \forall \texttt{arch.32BitCPU}$

$\texttt{Sempron} \sqsubseteq \texttt{AMD} \sqcap \exists \texttt{arch} \sqcap \forall \texttt{arch.64BitCPU}$

Fig. 1. Reference ontology

Description Logics are considered to be highly expressive representation languages, corresponding to decidable fragments of first-order logic. Reasoning systems based on DLs generally provide at least two basic inference services: *satisfiability* and *subsumption*.

Satisfiability: a concept expression C is satisfiable w.r.t. an ontology $\mathcal{T}$ when $\mathcal{T} \not\models C \sqsubseteq \bot$, or equivalently $C \not\sqsubseteq_{\mathcal{T}} \bot$;

Subsumption: a concept expression C is subsumed by a concept expression D w.r.t. $\mathcal{T}$ when $\mathcal{T} \models C \sqsubseteq D$, or equivalently $C \sqsubseteq_{\mathcal{T}} D$.

In our setting satisfiability is useful, for example, to catch inconsistencies among a buyer's expressed preferences or, analogously, among sellers' offered configurations. On the other hand, subsumption can be employed to verify if a particular seller's offer satisfies one or more buyer's preferences.

3 Preference Representation Using Description Logics

The problem of preference representation deals with the expression and evaluation of preferences over a set of different alternatives (outcomes). This problem can be challenging even for a small set of alternatives, involving a moderate number of features, as the user has to evaluate all possible configurations of feature values in the domain.

In this work, we deal with this problem by a combination of expressive language, to facilitate preference specification, and preference structure exploitation, justified by multiattribute utility theory.

Several approaches to negotiation have exploited logic languages in order to express preferences, most of them using propositional logic [8,9,10], however only few of the approaches proposed in the literature have explored the possibility to use also an ontology to model relations among attributes [11] or the use of more expressive logics as DLs [12,13]. Lukasiewicz and Schellhase [14] propose a framework to model conditional preferences in DLs for matchmaking. In their framework they refer to set of concepts and the formal semantics of implication is defined in terms of set membership. Such a formulation well suits their target matchmaking task. Indeed, they are not interesed in computing a utility value for a concept, e.g. an agreement, but they focus on ranking a set of results w.r.t. a query.

We point out the importance to refer to a background knowledge, *i.e.*, having an ontology $\mathcal{T}$, in order to model not only interdependencies among attributes in preference statements, but also to model inner relations among attributes that cannot be disregarded *e.g.*, is-a, disjoint or equivalence relations. In order to lay out the importance of an ontology and why we cannot abstract from it, we use a simple example involving one perspective buyer and two sellers.

Example 1. Let us suppose the buyer has among her preferences:

$$P = \exists \texttt{hasCPU}.(\texttt{AMD} \sqcap \exists \texttt{arch} \sqcap \forall \texttt{arch}.\texttt{64BitCPU}) \sqcap \exists \texttt{hasRam}.(\geq_2 \ GB)$$

(PC with a 64-bit AMD processor and at least 2 GB of memory)

and there are two sellers, A and B, that can provide the respective configurations:

$$A = \exists \texttt{hasCPU}.\texttt{Athlon64} \sqcap \texttt{hasRam}(=_2 GB)$$
$$B = \exists \texttt{hasCPU}.\texttt{Centrino} \sqcap \texttt{hasRam}(=_1 GB)$$

If we refer to the ontology $\mathcal{T}$ in Figure 1 we can state that seller A can satisfy the preference expressed by the buyer —from $\mathcal{T}$ we know that Athlon64 is a 64-bit AMD processor. Conversely, seller B cannot satisfy buyer's preference, because Centrino is not a 64-bit AMD processor. Δ

We extend the well-known approach of weighted propositional formulas [8,9,10], representing preferences as DL formulas, where at each formula we associate a value v representing the relative importance of that formula.

Definition 1. *Let $\mathcal{T}$ be an ontology in a DL. A **Preference** is a pair $\phi = \langle P, v\rangle$ where P is a concept such that $P \not\sqsubseteq_{\mathcal{T}} \bot$ and v is a real number assigning a worth to P. We call a finite set $\mathcal{P}$ of preferences a **Preference Set** iff for each pair of preferences $\phi' = \langle P', v'\rangle$ and $\phi'' = \langle P'', v''\rangle$ such that $P' \equiv_{\mathcal{T}} P''$ then $v' = v''$ holds.*

From now on, whenever we mention a set of preference we refer to a Preference Set.

Example 2 (Desktop Computer Negotiation). Imagine a negotiation setting where buyer and seller are negotiating on the characteristics of a *desktop computer*. The buyer will have some preferences, while the seller will have some different configurations to offer to the buyer in order to satisfy her preferences. *Let us hence suppose the buyer is looking for a desktop PC endowed with an AMD CPU. Otherwise, if the desktop PC has an Intel CPU, it should only be a Centrino one. The buyer also wants a desktop PC supporting wireless connection.* Following Definition 1 the buyer's Preference set is $\mathcal{P} = \{\langle P_1, v_1\rangle, \langle P_2, v_2\rangle, \langle P_3, v_3\rangle\}$, with:

$P_1 = \forall \texttt{hasCPU.Centrino} \sqcap \exists \texttt{hasCPU}$
$P_2 = \exists \texttt{hasCPU.AMD}$
$P_3 = \exists \texttt{netSupport.WiFi}$

On the other side, the seller could offer *a Desktop computer supporting either a wireless connection or an AMD CPU, specifically a Sempron one, and he does not have a desktop PC endowed with a Centrino CPU*.

$A = \texttt{DesktopComputer} \sqcap \neg\exists \texttt{hasCPU.Centrino} \sqcap (\exists \texttt{netSupport.WiFi} \sqcup \forall \texttt{hasCPU.Sempron})$

Therefore, given a preference set $\mathcal{P}$ and a proposal A, how to evaluate the utility of this proposal w.r.t. buyer's preferences? Intuitively, the utility value should be the sum of the value v_i of the preferences satisfied by the seller's proposal. Next sections will adress this problem, showing a computation method for weighted DL-formulas. Δ

4 Utility

Weighted formulas have been introduced for propositional logic to assign a utility value to a propositional model representing *e.g.*, the final agreement. The computation of the model and its corresponding utility is quite easy since in propositional logic we deal with a finite set of models. Following Chevaleyre et al. [10] utility is computed as:

$$\sum\{v \mid \langle P, v\rangle \in \mathcal{P} \text{ and } m \models P\}$$

Where $\mathcal{P}$ is a propositional Preference Set, m is a propositional interpretation (model) *e.g.*, representing the final agreement. [2]. We call this approach *model-based*. Less straightforward is the case of more expressive logics. Some attempts in this direction have been made by Ragone et al. [12,13] adapting the weighted formulas framework to Description Logics. There, they do not consider models as final agreements but formulas, and the utility value is computed as:

$$\sum\{v \mid \langle P, v\rangle \in \mathcal{P} \text{ and } A \sqsubseteq_{\mathcal{T}} P\}$$

Where $\mathcal{P}$ is a DL preference set, $\mathcal{T}$ is a DL ontology and A is a concept *e.g.*, representing a proposal in a negotiation process. We call this approach *implication-based*. Although very simple to compute and immediate, this basic approach may lead to counter-intuitive examples when dealing with logic languages allowing disjunction in the final agreement.

Example 3. Consider the following preference set $\mathcal{P}$ (here for the sake of clarity we do not consider the ontology $\mathcal{T}$)

$\phi_1 = \langle A_1, v_1\rangle$
$\phi_2 = \langle A_2, v_2\rangle$
$\phi_3 = \langle A_3, v_3\rangle$

and a concept A.

$A = (A_1 \sqcup A_3) \sqcap A_2$

Δ

In Example 3, following an *implication-based* approach the final utility is

$$u^{impl}(A) = v_2$$

Indeed, only $A \sqsubseteq P_2$ holds.

On the other hand, if we use a *model-based* approach we can say that the final utility value is:

$$u^{model}(A) \in \{v_1 + v_2, v_2 + v_3, v_1 + v_2 + v_3\}$$

If we consider interpretarions $\mathcal{I}$ of A we may have that only one of the conditions below holds:

$\mathcal{I} \models A_1 \sqcap \neg A_3 \sqcap A_2$
$\mathcal{I} \models \neg A_1 \sqcap A_3 \sqcap A_2$
$\mathcal{I} \models A_1 \sqcap A_3 \sqcap A_2$

Using a model-based approach, if we want to be as conservative as possible we may consider:

$$u^{model}(A) = min\{v_1 + v_2, v_2 + v_3, v_1 + v_2 + v_3\}$$

Conversely, in the most optimistic case we consider:

$$u^{model}(A) = MAX\{v_1 + v_2, v_2 + v_3, v_1 + v_2 + v_3\}$$

[2] $\sum\{\cdot\}$ indicates the summation over all the elements in the set $\{\cdot\}$.

In the rest of the paper we will refer to the conservative situation but all the results can be easily adapted to the optimistic case. Consequently, we give a definition of Minimal Model and of its corresponding Minimal Utility Value.

Definition 2 (Minimal Models – Minimal Utility Value). *Given an ontology $\mathcal{T}$, a concept A, such that $\mathcal{T} \not\models A \sqsubseteq \bot$, and a Preference Set $\mathcal{P}$, a **Minimal Model** is an interpretation $\mathcal{I}$ such that:*

1. *both $\mathcal{I} \models A$ and $\mathcal{I} \models \mathcal{T}$*
2. *the value $u^c(A) = \sum\{v \mid \langle P, v\rangle \in \mathcal{P}$ and $\mathcal{I} \models P\}$ is minimal*

*We call $u^c(A)$ a **Minimal Utility Value** for A w.r.t. to $\mathcal{P}$.*

4.1 Complexity

In this section, we give some consequences on the complexity of computing the utility of a formula A, when utility is attached to models. In what follows, we abstract from the particular logic language $\mathcal{L}$, which gives our results the maximum attainable generality.

Lemma 1. *Given two concepts $A, P \in \mathcal{L}$, and an ontology $\mathcal{T}$, we have $A \sqsubseteq_{\mathcal{T}} P$ iff there exists a minimal model assigning value v to A when preferences are $\mathcal{P} = \{\langle P, v\rangle\}$.*

Proof. Given a singleton set of preferences $\mathcal{P} = \{\langle P, v\rangle\}$, then $u^c(A)$ can be equal to either 0, or v. Now if there exists a minimal model with value 0, then such a model is a model of $\mathcal{T}$ and A, and it must be not a model of P; hence, when the minimal model has utility 0, $\mathcal{T} \not\models A \sqsubseteq P$. On the other hand, there exists a minimal model assigning utility value v to A, then every model of $\mathcal{T}$ and A is also a model of P. But this is just the condition expressing semantically that $A \sqsubseteq_{\mathcal{T}} P$. □

A consequence of the above lemma is that computing a minimal model is at least as hard as computing subsumption in a DL $\mathcal{L}$; below we can even generalize this result to logical implication.

Theorem 1. *Given a language $\mathcal{L}$ in which deciding logical implication is $\mathcal{C}$-hard, deciding the existence of a minimal model with a given utility value is $\mathcal{C}$-hard, too.*

We observe that the result is the same (by definition) when utilities are directly assigned to *formulas*, as done *e.g.*, by Ragone et al. [12].

We now move to upper bounds on computing utilities over formulas by minimal models. We first assess the upper bound of the decision problem corresponding to computing the utility of a concept.

Theorem 2. *Let $\mathcal{L}$ be a language in which the satisfiability problem belongs to the complexity class $\mathcal{C}$, and such that* NP $\subseteq \mathcal{C}$*; moreover, let v be a positive real number, $\mathcal{P}$ be a set of preferences, $\mathcal{T}$ be a Terminology and A a concept, all expressed in $\mathcal{L}$. Then, deciding whether $u^c(A) < v$ is a problem in $\mathcal{C}$.*

Proof. Let $\mathcal{P} = \{\langle P_1, v_1\rangle, \ldots, \langle P_n, v_n\rangle\}$. Then, for each integer m between 0 and $2^n - 1$, let $(m)_2 = b_1 b_2 ... b_n$ be the binary representation of m, and let D_m be the

concept defined as $A \sqcap B_1 \sqcap \cdots \sqcap B_n$, where for each $i = 1, ..., n$, if $b_i = 0$ then $B_i = \neg P_i$, else $B_i = P_i$. Intuitively, the i-th bit of $(m)_2$ decides whether P_i appears positively or negatively in D_m. Now let $\mathcal{S} = \{m \mid 0 \leq m \leq 2^{n-1}$ and $D_m \not\sqsubseteq_{\mathcal{T}} \bot\}$, *i.e.*, the set of all integers $m \in [0, 2^{n-1}]$ such that D_m be satisfiable in $\mathcal{T}$. Then, the utility of A can be expressed as

$$min \left\{ \sum_{i=1}^{n} b_i * v_i \;\middle|\; m \in \mathcal{S} \text{ and } (m)_2 = b_1 b_2 ... b_n \right\} \tag{1}$$

Intuitively, one searches the minimum of the objective function (1) over a *subset* $\mathcal{S}$ of the hypercube $\{0, 1\}^n$, where the vertices in $\mathcal{S}$ are only the ones which define a satisfiable combination of A and (possibly negated) preferences in $\mathcal{P}$. Clearly, for every satisfiable conjunction at least one model M exists, and when the utility computed by (1) is minimum, M is a minimal model.

Finally, observe that a "right" number m between 0 and 2^{n-1}—*i.e.*, a number individuating a minimal model—can be guessed nondeterministically in polynomial time. Hence, a nondeterministic Turing machine deciding $u^c(A) < v$ guesses a number m, checks the satisfiability of D_m (a problem in $\mathcal{C}$), computes $u = \sum_{i=1}^{n} b_i * v_i$, and halts with "yes" if $u < v$, otherwise "no". Therefore, when $\mathcal{C} =$ NP, the overall decision problem is in NP; when NP $\subset \mathcal{C}$, the satisfiability check in $\mathcal{C}$ dominates the overall complexity. □

For languages such that PSPACE $\subseteq \mathcal{C}$, the above theorem yields also an upper bound on the complexity of *computing* the utility of a concept. For instance, for $\mathcal{L} = \mathcal{ALC}$, and simple Terminologies, satisfiability is a problem PSPACE-complete [15]. Then computing (1) is a problem in PSPACE, too.

For languages such that $\mathcal{C} =$ NP, the above theorem yields an NPO upper bound on the complexity of *computing* the utility of a concept. We discuss this case in more detail, for $\mathcal{L}$ =Propositional Logic (PL). For this case, Theorem 1 implies that the decision problem is NP-hard; yet it does not tell us whether the computation of $u^c(A)$ admits some approximation schema, or not. We can show also NPO-hardness of computing (1), through a (not difficult) reduction from MAX-2-SAT, which is the problem of finding a model maximizing the number of clauses in a CNF of 2-literals (a.k.a. Krom) clauses. For convenience, we use the dual problem of finding a model that minimizes the number of *unsatisfied* conjunctions in a DNF of 2-literals conjunctions $D = D_1 \vee \cdots \vee D_n$. Then such a model minimizes also the utility of any (unused) literal C w.r.t. the set of preferences $\mathcal{P}_D = \{\langle D_1, 1\rangle, \ldots, \langle D_n, 1\rangle\}$. Since computing such a model is NPO-hard, our claim follows.

Observe that the above theorem does not hold for classes below NP, which need separate discussions. The case $\mathcal{C}$=PTIME covers the most simple logics, such as $\mathcal{L}$ =Conjunctions of Literals, or the DL $\mathcal{L} = \mathcal{FL}^-$ [6]. In fact, satisfiability of a conjunction amounts to check the absence of a literal and its negation in the conjunction. Yet, observe that if $\mathcal{P} \supseteq \{\langle A, v_1\rangle, \langle \neg A, v_2\rangle\}$, then for every formula C, $u^c(C) \geq min(v_1, v_2)$. In general, deciding if $u^c(C) \leq k$ is NP-complete, based on a simple reduction from 3-TAUT: given a DNF $D = D_1 \vee \cdots \vee D_n$, where each D_i is a conjunction, D is *not* a tautology iff $u^c(A) \leq k$ (A being any literal) w.r.t. the preferences $\mathcal{P} = \{\langle D_1, 2k\rangle, \ldots, \langle D_n, 2k\rangle\}$.

Less obviously, the same reduction holds for $\mathcal{L} = \mathcal{FL}^-$, using a role R_i for every propositional atom A_i, and letting the encoding γ be: $\gamma(\wedge) = \sqcap$, $\gamma(A_i) = \exists R_i$, and $\gamma(\neg A_i) = \forall R_i.B$ (for some concept name B). Then the previous DNF D is not a tautology iff $u^c(B) \leq k$ w.r.t. $\mathcal{P} = \{\langle\gamma(D_1), 2k\rangle, \ldots, \langle\gamma(D_n), 2k\rangle\}$. The fact that deciding the utility of an $\mathcal{FL}^-$ concept w.r.t. a set of $\mathcal{FL}^-$ preferences is NP-hard is remarkable, since satisfiability in $\mathcal{FL}^-$ is trivial (every $\mathcal{FL}^-$ concept is satisfiable).

5 Computation of Minimal Utility Value

In this section we show how the computation of the *minimal utility value* for a set of preferences $\mathcal{P}$ w.r.t. a concept A can be turned out in solving an optimization problem.

Given the set $\mathcal{P} = \{\phi_1, \phi_2, \phi_3\}$ of preferences and the concept A as in Example 3, we note that:

(a) A is more specific than the simple preference specification A_2;
(b) A is more specific than a disjunction of preference specification (that, in the most general case, may appear even negated).

On the other hand, due to constraints modeled within the ontology we may have some interrelations among elements of $\mathcal{P}$. For instance, it might result that:

(c) two preferences ϕ_1 and ϕ_2 cannot be satisfied at the same time;
(d) the conjunction of the former with the complement of the latter could be unsatisfiable;
(e) the combination of the complement of ϕ_1 and ϕ_2 is more specific than (*i.e.*, it implies) a third preference ϕ_3. In other words, **(c)** no model of A_1 can be also a model of A_2, **(d)** no model of A_1 can be also a model of $\neg A_2$, **(e)** all models of both $\neg A_1$ and A_2 are also models of A_3.

In term of interpretations it results that for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$:

(a) $A^{\mathcal{I}} \subseteq (A_2)^{\mathcal{I}}$
(b) $A^{\mathcal{I}} \subseteq (A_1)^{\mathcal{I}} \cup (A_3)^{\mathcal{I}}$
(c) $(A_1)^{\mathcal{I}} \cap (A_2)^{\mathcal{I}} = \emptyset$
(d) $(A_1)^{\mathcal{I}} \cap (\Delta^{\mathcal{I}} \setminus (A_2)^{\mathcal{I}}) = \emptyset$
(e) $(\Delta^{\mathcal{I}} \setminus (A_1)^{\mathcal{I}}) \cap (A_2)^{\mathcal{I}} \subseteq (A_3)^{\mathcal{I}}$

Actually, if we consider also a concept A it is easy to see that whenever (c), (d) or (e) hold, then also the corresponding relations represented below are true (while the vice versa is false).

(f) $A \sqcap A_1 \sqcap A_2 \sqsubseteq_{\mathcal{T}} \bot \longrightarrow A^{\mathcal{I}} \cap (A_1)^{\mathcal{I}} \cap (A_2)^{\mathcal{I}} = \emptyset$
(g) $A \sqcap A_1 \sqcap \neg A_2 \sqsubseteq_{\mathcal{T}} \bot \longrightarrow \cap A^{\mathcal{I}} \cap (A_1)^{\mathcal{I}} \cap (\Delta^{\mathcal{I}} \setminus (A_2)^{\mathcal{I}}) = \emptyset$
(h) $A \sqcap \neg A_1 \sqcap A_2 \sqsubseteq_{\mathcal{T}} A_3 \longrightarrow A^{\mathcal{I}} \cap (\Delta^{\mathcal{I}} \setminus (A_1)^{\mathcal{I}}) \cap (A_2)^{\mathcal{I}} \subseteq (A_3)^{\mathcal{I}}$

In fact, in order to compute a *Minimal Utility Value*, if A represents *e.g.*, a final agreement, we are more interested in those models satisfying the latter equations rather than the ones satisfying (c), (d) and (e) because they are also models of A (as the *Minimal Model* is). Obviously, (a),(b),(f),(g),(h) can be generalized w.r.t. whatever *Preference Set*.

Noteworthy is that, since we use an ontology $\mathcal{T}$, the above observations apply to preference specifications represented as general concept expressions C and not only as concept names. We illustrate the above ideas with the help of an example.

Example 4 (Desktop Computer Negotiation cont'd). We again refer to the ontology $\mathcal{T}$ depicted in Figure 1.

We recall that the buyer's Preference Set is $\mathcal{P} = \{\langle P_1, v_1\rangle, \langle P_2, v_2\rangle, \langle P_3, v_3\rangle\}$, with:

$P_1 = \forall \texttt{hasCPU.Centrino} \sqcap \exists \texttt{hasCPU}$
$P_2 = \exists \texttt{hasCPU.AMD}$
$P_3 = \exists \texttt{netSupport.WiFi}$

while the seller proposal is:

$A \equiv \texttt{DesktopComputer} \sqcap \neg \exists \texttt{hasCPU.Centrino} \sqcap (\exists \texttt{netSupport.WiFi} \sqcup \forall \texttt{hasCPU.Sempron})$

Notice that, because of the axioms in ontology $\mathcal{T}$:

$\texttt{Intel} \sqsubseteq \neg \texttt{AMD}$
$\texttt{Centrino} \sqsubseteq \texttt{Intel} \sqcap \exists \texttt{arch} \sqcap \forall \texttt{arch.32BitCPU}$

preferences P_1 and P_2 cannot be satisfied at the same time, and, moreover, due to the axiom

$$\exists \texttt{hasCPU.Centrino} \sqsubseteq \exists \texttt{netSupport.WiFi}$$

in the ontology $\mathcal{T}$, preference P_1 is more specific than preference P_3. Δ

Definition 3 (Preference Clause). *Given a set of preferences $\mathcal{P} = \{\langle P_i, v_i\rangle\}, i = 1 \ldots n$, an ontology $\mathcal{T}$ and a concept A such that $A \not\sqsubseteq_{\mathcal{T}} \bot$, we say that $\mathcal{P}$ is constrained if the following condition holds:*

$$A \sqsubseteq_{\mathcal{T}} \hat{P}_1 \sqcup \ldots \sqcup \hat{P}_n \tag{2}$$

*Where $\hat{P}_i \in \{P_i, \neg P_i\}$. We call $\hat{P}_1 \sqcup \ldots \sqcup \hat{P}_n$ a **Preference Clause** if there is no strict subset $\mathcal{Q} \subset \mathcal{P}$ such that $\mathcal{Q}$ is constrained.*

Note that with a *Preference Clause* one can represent not only relations (a) and (b) but also relations (f), (g) and (h) thanks to the well known equivalence:

$$C \sqsubseteq_{\mathcal{T}} D \Longleftrightarrow C \sqcap \neg D \sqsubseteq_{\mathcal{T}} \bot$$

In fact,

(f) $A \sqcap A_1 \sqcap A_2 \sqsubseteq_{\mathcal{T}} \bot \iff A \sqsubseteq_{\mathcal{T}} \neg A_1 \sqcup \neg A_2$
(g) $A \sqcap A_1 \sqcap \neg A_2 \sqsubseteq_{\mathcal{T}} \bot \iff A \sqsubseteq_{\mathcal{T}} \neg A_1 \sqcup A_2$
(h) $A \sqcap \neg A_1 \sqcap A_2 \sqsubseteq_{\mathcal{T}} A_3 \iff A \sqsubseteq_{\mathcal{T}} A_1 \sqcup \neg A_2 \sqcup A_3$

We may say that a *Preference Clause* contains the minimal set of preferences such that Equation (2) holds.

Definition 4 (Preference Closure). *Given a Preference set $\mathcal{P} = \{\phi_i\}, i = 1 \ldots n$, an ontology $\mathcal{T}$ and a concept $A \not\sqsubseteq_{\mathcal{T}} \bot$, we call* ***Preference Closure****, denoted as $\mathcal{CL}$, the set of Preference Clauses built, if any, for each set in $2^{\mathcal{P}}$.*

In other words, a *Preference Closure* represents the set of all possible *Preference Clauses* over $\mathcal{P}$. It represents all possible (minimal) interrelations occurring between A and preference descriptions in $\mathcal{P}$ w.r.t. an ontology $\mathcal{T}$.

Proposition 1. *Given a concept A, a* ***Preference Closure*** *$\mathcal{CL}$ and an ontology $\mathcal{T}$, if $\mathcal{I}^m$ is a Minimal Model of A then*

$$\mathcal{I}^m \models \mathcal{CL} \tag{3}$$

Proof. By Definition 2 since $\mathcal{I}^m$ is a Minimal Model then $\mathcal{I}^m \models \mathcal{T}$ and $\mathcal{I}^m \models A$. As for all models $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{T}$, including $\mathcal{I}^m$, also $\mathcal{I} \models \mathcal{CL}$ is satisfied, then the proposition holds. □

In order to compute *Minimal Utility Value* $u^c(A)$ we reduce to an optimization problem (OP). Usually, in an OP we have a set of constrained numerical variables and a function to be maximized/minimized. In our case we will represent constraints as a set χ of linear inequalities over binary variables, *i.e.*, variables whose value is in $\{0, 1\}$, and the function to be minimized as a weighted combination of such variables. In order to represent χ we need some pre-processing steps.

1. Compute the *Preference Closure* $\mathcal{CL}$ for $\mathcal{P}$;
2. For each *Preference Clause* $A \sqcap \hat{P}_1 \sqcap \ldots \hat{P}_n \sqsubseteq_{\mathcal{T}} \bot \in \mathcal{CL}$, compute a corresponding preference constraint set $\overline{\mathcal{CL}} = \{\neg\hat{P}_1, \ldots, \neg\hat{P}_n\}$. We denote with $\overline{\mathcal{CL}}^c = \{\overline{\mathcal{CL}}\}$ the set of all preference constraint sets.

Observation 1. *The reason why we do not consider $\neg A$ when computing $\overline{\mathcal{CL}}$ is that in order to compute a* Minimal Utility Value *we are looking for Minimal Models,* i.e.*, models such that $A^{\mathcal{I}}$ (and $\mathcal{I} \models \mathcal{T}$) satisfying properties of Definition 2. Each Preference Clause can be rewritten as $\mathcal{T} \models \top \sqsubseteq \neg A \sqcup \neg\hat{P}_1 \sqcup \ldots \neg\hat{P}_n$. If we rewrite the right hand side of the relation in terms of interpretation functions, from the semantics of $\sqcup$ operator, we have*

$$(\neg A)^{\mathcal{I}} \cup (\neg\hat{P}_1)^{\mathcal{I}} \cup \ldots (\neg\hat{P}_n)^{\mathcal{I}} \neq \emptyset \tag{4}$$

Since a Minimal Model is an interpretation such that $A^{\mathcal{I}} \neq \emptyset$, then all the models we are looking for are such that $(\neg A)^{\mathcal{I}} = \emptyset$. As a consequence, for our computation, the term $(\neg A)^{\mathcal{I}}$ in Equation (4) is meaningless. We will clarify further this point in Observation 2 while discussing the OP we build to compute the Minimal Utility Value *in the following.*

Example 5 (Desktop Computer Negotiation cont'd). Consider again the Desktop Computer negotiation of Example 2. Given the set of preference $\mathcal{P} = \{\langle P_1, v_1\rangle, \langle P_2, v_2\rangle, \langle P_3, v_3\rangle\}$ and the proposal A, if we compute the **Preference Closure** $\mathcal{CL}$ we find:

$$\mathcal{CL} = \left\{ \begin{array}{l} A \sqcap P_1 \sqsubseteq_{\mathcal{T}} \bot; \\ A \sqcap \neg P_2 \sqcap \neg P_3 \sqsubseteq_{\mathcal{T}} \bot \end{array} \right\}$$

hence, the two corresponding preference constraint sets in $\overline{\mathcal{CL}}^c$ are:

$$\begin{aligned}\overline{\mathcal{CL}}_1 &= \{\neg P_1\}\\ \overline{\mathcal{CL}}_2 &= \{P_2, P_3\}\end{aligned}$$ Δ

Based on well-known encoding of clauses into linear inequalities (*e.g.*, [16, p.314]) we transform each set $\overline{\mathcal{CL}} \in \overline{\mathcal{CL}}^c$ in a set of linear inequalities χ and then define a function to be minimized in order to solve an OP.

Definition 5 (Minimal Utility Value OP). *Let $\mathcal{P}$ be a set of preferences and $\overline{\mathcal{CL}}^c$ be the set of all preference constraint sets. We define a* Minimal Utility Value OP*, represented as $\langle \chi, u(\mathbf{p})\rangle$, the optimization problem built as follows:*

1. ***numerical variables*** *– for each preference $\langle P_i, v_i\rangle \in \mathcal{P}$, with $i = 1, \ldots, n$ introduce a binary variable p_i and define the corresponding array $\mathbf{p} = (p_1, \ldots, p_n)$ (see Example 6);*
2. ***set χ of linear inequalities*** *– pick up each set $\overline{\mathcal{CL}} \in \overline{\mathcal{CL}}^c$ and build the linear inequalities*

$$\sum\{(1-p) \mid \neg P \in \overline{\mathcal{CL}}\} + \sum\{p \mid P \in \overline{\mathcal{CL}}\} \geq 1$$

3. ***function to be minimized*** *– given the array $\mathbf{p}$ of binary variables*

$$u(\mathbf{p}) = \sum\{v \cdot p \mid p \text{ is the variable mapping } \langle P, v\rangle\}$$

Observation 2. *If we considered also $\neg A$ when computing the sets $\overline{\mathcal{CL}} \in \overline{\mathcal{CL}}^c$ we would have had inequalities in the form:*

$$(1-a) + \sum\{(1-p) \mid \neg P \in \overline{\mathcal{CL}}\} + \sum\{p \mid P \in \overline{\mathcal{CL}}\} \geq 1$$

Since we are interested in models where $A^{\mathcal{I}}$ is interpreted as nonempty, then variable a has to be equal to 1. Hence the first element of the above summation is always equal to 0. In other words, we can omit $\neg A$ when computing a preference constraint set $\overline{\mathcal{CL}}$.

The solution to a *Minimal Utility Value OP* will be an assignment $\mathbf{p}_s$ for $\mathbf{p}$, *i.e.*, an array of $\{0, 1\}$-values, minimizing $u(\mathbf{p})$.

Example 6 (Desktop Computer Negotiation cont'd). Back to the Desktop Computer negotiation of Example 2, after the computation of Preference Closures and set $\overline{\mathcal{CL}}^c$, we build the corresponding optimization problem in order to find the model with the minimal utility value:

$$\begin{aligned}\mathbf{p} &= (p_1, p_2, p_3)\\ \chi &= \begin{cases} 1 - p_1 \geq 1\\ p_2 + p_3 \geq 1\end{cases}\\ u(\mathbf{p}) &= v_1 \cdot p_1 + v_2 \cdot p_2 + v_3 \cdot p_3\end{aligned}$$

Possible solutions are:

$$\mathbf{p}'_s = (0,1,0)\ ,\ u(\mathbf{p}'_s) = v_2$$
$$\mathbf{p}''_s = (0,0,1)\ ,\ u(\mathbf{p}''_s) = v_3$$
$$\mathbf{p}'''_s = (0,1,1)\ ,\ u(\mathbf{p}'''_s) = v_2 + v_3$$

The minimal solution will be either $\mathbf{p}'_s$ or $\mathbf{p}''_s$, depending of the value of v_2 and v_3. $\triangle$

Given a a solution $\mathbf{p}_s$ to a *Minimal Utility Value OP* $\langle \chi, u(\mathbf{p}) \rangle$, we call *Minimal Preference Set* $\overline{\mathcal{P}}^m$ and *Minimal Assignment* A^m, respectively, the set and the formula built as in the following[3]:

$$\overline{\mathcal{P}}^m = \{\langle P_i, v_i \rangle \mid p_i = 1 \textit{ in the solution } \mathbf{p}_s\}$$
$$A^m = \bigsqcap\{P_i \mid p_i = 1 \textit{ in the solution } \mathbf{p}_s\} \sqcap \bigsqcap\{\neg P_i \mid p_i = 0 \textit{ in the solution } \mathbf{p}_s\}$$

Theorem 3. *Given a solution* $\mathbf{p}_s$ *to a* Minimal Utility Value OP $\langle \chi, u(\mathcal{P}) \rangle$ *and a* Minimal Assignment A^m*:*

1. *if* $\mathcal{I}^m$ *is a* Minimal Model *then* $\mathcal{I}^m \models A^m$*;*
2. $u(\mathbf{p}_s)$ *is a* Minimal Utility Value*.*

Proof. First we show that there exists at least one model $\mathcal{I}^m \models \mathcal{T}$ *such that both* $\mathcal{I}^m \models A$ *and* $\mathcal{I}^m \models A^m$*. If* $\mathcal{I}^m$ *did not exist, then* $A^{\mathcal{I}^m} \cap (A^m)^{\mathcal{I}^m} = \emptyset$*. We can easily rewrite the latter relation as* $A \sqcap A^m \sqsubseteq_{\mathcal{T}} \bot$ *which is equivalent to* $A \sqsubseteq_{\mathcal{T}} \neg A^m$*. But this is not possible. Indeed, if* A *and* A^m *were inconsistent with each other w.r.t.* $\mathcal{T}$ *then, by Proposition 1 we should have the corresponding Preference Clause in* $\mathcal{CL}$ *and the related inequality in* χ*:*

$$\sum\{(1-p) \mid P \textit{ appears in } A^m\} + \sum\{p \mid \neg P \textit{ appears in } A^m\} \geq 1$$

In order to be satisfied, the latter inequality must have either (a) at least one variable assigned to 0 *in the first summation or (b) at least one variable assigned to* 1 *in the second one. Case (a) means that the corresponding preference is not satisfied by* A^m *while case (b) means that the corresponding preference is satisfied by* A^m*. Both cases are conflicting with the definition of* A^m*.*

By construction of χ*, we have that if* $\mathcal{I}^m \models A^m$ *then* $\mathcal{I}^m \models A$ *(see Observation 2). Since* A^m *comes from the minimization of* $u(\mathbf{p}_s)$ *then* $\mathcal{I}^m \models A^m$ *represents a model of* A^m *(and then of* A*) such that*

$$\sum\{v \mid \langle P, v \rangle \in \mathcal{P} \textit{ and } \mathcal{I}^m \models P\}$$

is minimal.

It is straightforward to show that $u(\mathbf{p}_s)$ *is a* Minimal Utility Value*.* $\square$

[3] With $\bigsqcap\{\cdot\}$ we denote the conjunction of all the concepts in the set $\{\cdot\}$.

5.1 Computational Properties of the Method

We now relate the computation method proposed in this section with the computational complexity results of the previous section. First of all, we observe that the size of the Preference Closure $|\mathcal{CL}|$ can be—in the worst case—exponential in n, the size of the preference set. Since Linear Integer Programming is an NPO-complete problem[16], overall our problem can be solved nondeterministically in exponential time and space.

However, we observe that $|\mathcal{CL}|$ *does not depend* on the size of the ontology $\mathcal{T}$, which typically is much larger than the size of $\mathcal{P}$. In some sense, $\mathcal{CL}$ *compiles out* all the complexity due to the satisfiability problem in the chosen DL, leaving the OP of the combinatorics related to compatibility of preferences among each other, and with the formula C whose minimal utility value has to be computed. This is perfectly reasonable when Satisfiability in the chosen DL is a PSPACE-complete problem, or harder, since the best known procedures for solving PSPACE-complete problems use exponential time anyway, and the space used is exponential only in the number of preferences, not in the size of $\mathcal{T}$.

For the cases in which the language for preferences has a low-complexity satisfiability problem, say, NP, or PTIME , though, preprocessing into $\mathcal{CL}$ the complete structure of preference compatibilities may be an overshoot. In such cases, it would seem more reasonable to devise specialized procedures that compute on demand the satisfiability of a conjunction of preferences.

An orthogonal analysis can be done on the *scalability* of the method when the utilities of several offers $C_1, \ldots, C_m$ must be compared. Here it seems that one has to solve m separate OPs of size exponential in $|\mathcal{P}|$. While this is the worst case, some optimization based on the logic for offers is possible. In fact, observe that $C_i \sqsubseteq_{\mathcal{T}} C_j$ implies $u^c(C_j) \leq u^c(C_i)$ (a model of C_i is also a model of C_j). Hence, when searching for the offer with the maximum least utility, C_j can be safely disregarded. Intuitively, more specific offers are preferred over more generic ones, with the intuition that a generic offer C_j has a worst-case utility $u^c(C_j)$ which is less than the worst-case utility $u^c(C_i)$ of a more specific offer C_i.

6 Conclusion

Logic languages have been proposed here as a natural and powerful preference representation tool for automated negotiation purposes. We have shown how it is possible to compute a utility value for a concept (agreement, proposal, allocation), when preferences are expressed as weighted DL formulas w.r.t. a shared ontology $\mathcal{T}$. Although we ground our framework in the DLs realm, we point out that the framework itself is completely general and suitable for whatever decidable fragment of FOL. We also reported complexity results and showed the applicability and benefits of our approach with the help of a meaningful example. Currently, we are studying how to combine this approach with graphical models, and in particular GAI (Generalized Additive Independence) [17,3], in order to model multiattribute auctions.

References

1. Keeney, R.L., Raiffa, H.: Decisions with Multiple Objectives: Preferences and Value Tradeoffs. John Wiley & Sons, New York (1976)
2. Gonzales, C., Perny, P.: GAI networks for utility elicitation. In: Ninth Intl. Conf. on Principles of Knowledge Representation and Reasoning, Whistler, BC, Canada, pp. 224–234 (2004)
3. Bacchus, F., Grove, A.: Graphical models for preference and utility. In: Eleventh Conf. on Uncertainty in Artificial Intelligence, Montreal, pp. 3–10 (1995)
4. Boutilier, C., Brafman, R.I., Domshlak, C., Hoos, H.H., Poole, D.: CP-nets: A tool for representing and reasoning about conditional ceteris paribus preference statements. Journal of Artificial Intelligence Research 21, 135–191 (2004)
5. Engel, Y., Wellman, M.P.: CUI networks: A graphical representation for conditional utility independence. Journal of Artificial Intelligence Research 31, 83–112 (2008)
6. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook. Cambridge Univ. Press, Cambridge (2002)
7. Baader, F., Hanschke, P.: A Scheme for Integrating Concrete Domains into Concept Languages. Technical Report Tech. Rep. RR-91-10 (1991)
8. Pinkas, G.: Propositional non-monotonic reasoning and inconsistency in symmetric neural networks. In: Twelfth Intl. Joint Conf. on Artificial Intelligence, pp. 525–531 (1991)
9. Lafage, C., Lang, J.: Logical representation of preferences for group decision making. In: Seventh Intl. Conf. on Principles of Knowledge Representation and Reasoning, pp. 457–468 (2000)
10. Chevaleyre, Y., Endriss, U., Lang, J.: Expressive power of weighted propositional formulas for cardinal preference modelling. In: Tenth International Conference on Principles of Knowledge Representation and Reasoning, pp. 145–152 (2006)
11. Ragone, A., Di Noia, T., Di Sciascio, E., Donini, F.: A logic-based framework to compute Pareto agreements in one-shot bilateral negotiation. In: Seventeenth European Conference on Artificial Intelligence, pp. 230–234 (2006)
12. Ragone, A., Di Noia, T., Di Sciascio, E., Donini, F.M.: Description logics for multi-issue bilateral negotiation with incomplete information. In: Twenty-Second AAAI Conference on Artificial Intelligence, pp. 477–482 (2007)
13. Ragone, A., Di Noia, T., Di Sciascio, E., Donini, F.M.: Alternating-offers protocol for multi-issue bilateral negotiation in semantic-enabled marketplaces. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 395–408. Springer, Heidelberg (2007)
14. Lukasiewicz, T., Schellhase, J.: Variable-strength conditional preferences for matchmaking in description logics. In: KR, pp. 164–174 (2006)
15. Baader, F., Hollunder, B.: $\mathcal{KRIS}$: $\mathcal{K}$nowledge $\mathcal{R}$epresentation and $\mathcal{I}$nference $\mathcal{S}$ystem. SIGART Bulletin 2(3), 8–14 (1991)
16. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Prentice-Hall, Englewood Cliffs (1982)
17. Fishburn, P.C.: Interdependence and additivity in multivariate, unidimensional expected utility theory. International Economic Review 8, 335–342 (1967)

Explaining and Predicting the Behavior of BDI-Based Agents in Role-Playing Games

Michal P. Sindlar, Mehdi M. Dastani, Frank Dignum, and John-Jules Ch. Meyer*

University of Utrecht
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{michal,mehdi,dignum,jj}@cs.uu.nl

Abstract. Virtual characters in games operate in a social context involving other characters and possibly human players as well. If such socially situated virtual characters are to be considered believable to players, they should appear to adjust their behavior based on their (presumed) beliefs about the mental states of other characters. Autonomous BDI-based agents are suitable for modeling characters that base their actions on mental states attributed to other agents. In this paper, it is illustrated how agent-based characters can infer the mental state of other virtual characters by observing others' actions in the context of some scene in a role-playing game. Contextual information can be utilized in explanation and prediction of agents' behavior, and as such can form the basis for developing characters that appear to be socially aware.

1 Introduction

For games and simulations with interactive virtual characters to provide users with a satisfying experience, it is of vital importance that those characters are believable to the user. Appearing to pursue goals and to be responsive to social context are determining factors for believability [1], and interaction with virtual characters is indeed richer and more enjoyable if these anticipate the behavior of other characters [2]. Believable characters that operate in a social context should exhibit social awareness and not only pursue their own interests, but also be able to attribute mental states to other characters and take these into account in their decision-making process. A survey among avid players of role-playing games has shown that these players are generally dissatisfied with the believability of non-player characters in these games, in part because they feel that such characters are not believable as far as their social behavior is concerned [3].

Statistical approaches to game-based plan recognition exist [4] but require large amounts of gameplay data to be processed, which might not always be available. Recent work in the agent programming community has focused on

* This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 174–191, 2010.

recognizing an agent's plan on grounds of its program and observed actions [5], and inferring the mental state of the agent that plausibly explains observed behavior [6]. This offers promising directions for developing virtual characters that are socially believable, as characters which can infer others' mental states have the possibility of incorporating attributed mental states into their own decision-making process. This allows for plausibly misguided behavior if inferred explanations are valid but incorrect, and can thus contribute to believability [7]. However, current approaches ignore the social setting in which agents operate and inferred explanations are independent of social context. If characters in games are designed to behave as autonomous agents, then a game can be regarded as an agent society [8], and the behavior of characters in such a game can be explained and predicted with respect to this context, which can be described or specified by means of organizational models that define roles, norms, etc. [9].

This paper presents a declarative solution to abducing the mental state of virtual characters implemented as BDI-based agents, which takes into account the social context in which these characters operate. It builds on earlier work regarding mental state abduction, which is reviewed in Sect. 2. Sect. 3 introduces the game-related context in terms of scenes and roles, and in Sect. 4 it is shown how this context can be utilized in explaining and predicting agent behavior. Sect. 5 ties things together in an example, and Sect. 6 concludes with a brief discussion and ideas for future research.

2 Mental State Abduction

In this section it is described how the observed behavior of agents can be related to an explanation in terms of a description of their mental state, recapitulating our work in [6]. In Defs. 1-2 the behavior of agents is described, and in the remainder of this section it is shown how behavior can be observed and explained.

Let $\mathcal{L}$ be a propositional domain language with negation and conjunction over propositions. $\mathsf{Lit} \in \mathcal{L}$ is the set of literals in this language, and $\mathcal{L}_\Gamma \subseteq \mathcal{L}$ a simple language allowing only consistent conjunctions of literals in $\mathcal{L}$. Let Act be a set of action names. The behavior of an agent can be described as an expression consisting of actions, which are either atomic observable actions $\alpha \in \mathsf{Act}$, or tests $\phi?$ on propositions $\phi \in \mathcal{L}$. Actions can be composed by means of operators for sequential composition (;) and choice (+).

Definition 1 (behavioral description). *Let $\alpha \in \mathsf{Act}$ be an atomic observable action, and $\phi?$ the test action on proposition $\phi \in \mathcal{L}$. The set of behavioral descriptions $\mathcal{L}_\Pi$ with typical element π is then defined as follows.*

$$\pi ::= \alpha \mid \phi? \mid \pi_1; \pi_2 \mid \pi_1 + \pi_2$$

Note that there is no notion of iteration in $\mathcal{L}_\Pi$. It is assumed, though, that behavior can be iteratively performed if an agent reapplies a behavioral rule of the type defined in Def. 2. Such a rule states that the behavior described by π is appropriate for achieving the goal γ if the condition β is believed to hold, and is assumed to be operationalized by some agent interpreter [10,11].

Definition 2 (behavioral rules). *Behavioral rules specify a relation between behavior* $\pi \in \mathcal{L}_{\Pi}$ *and a mental state consisting of achievement goals* $\gamma \in \mathcal{L}_{\Gamma}$ *and beliefs* $\beta \in \mathcal{L}$*. The set of behavioral rules* $\mathcal{L}_{\mathcal{BR}}$ *has* br *as its typical element.*

$$\mathsf{br} ::= \gamma \leftarrow \beta \uparrow \pi$$

The atomic actions $\alpha \in \mathsf{Act}$ of an agent are taken to be publicly observable, such that a perceived action can be directly related to the performed action. Because behavioral descriptions do not allow for concurrent actions and interpretation of the agent program is assumed to do so neither, the perception of multiple actions performed by a single agent is taken to be sequential. To distinguish sequential perception of actions — which is an incremental process — from actions in sequence as part of a plan, a percept is a list of actions.

Definition 3 (percepts). *Let* $\alpha \in \mathsf{Act}$ *be an observable action and* ϵ *a special empty (null) action, such that* $(\epsilon\delta) \stackrel{def}{=} (\delta) \stackrel{def}{=} (\delta\epsilon)$*. The set of percept expressions* $\mathcal{L}_{\Delta}$*, with typical element* δ*, is then defined as follows.*

$$\delta ::= \alpha \mid \epsilon \mid \delta_1\delta_2$$

In order to explain perceived actions in intentional terms, a relation has to be established between perceived actions, descriptions of behavior, and the behavioral rules that connect a mental state (goals and beliefs) to behavior. This relation should be defeasible, because on grounds of observed actions alone it is usually not possible to analytically infer the mental state that caused the agent to perform those actions. For this reason the behavioral rules of Def. 2 are described as the logical implications defined in Def. 4, stating that a precondition consisting of a goal and belief description implies a certain behavior, which are then interpreted in an abductive way to find explanations for observed behavior.

Definition 4 (rule description). *Let* $(\gamma \leftarrow \beta \uparrow \pi) \in \mathcal{L}_{\mathcal{BR}}$ *be a behavioral rule. The set of rule descriptions* $\mathcal{L}_{\mathcal{RD}}$*, with typical element* rd*, is defined as follows.*

$$\mathsf{rd} ::= \mathsf{goal}(\gamma) \wedge \mathsf{belief}(\beta) \Rightarrow \mathsf{behavior}(\pi)$$

A function $\mathsf{desc} : \mathcal{L}_{\mathcal{BR}} \longrightarrow \mathcal{L}_{\mathcal{RD}}$ *maps rules to their description, such that* $\mathsf{desc}(\mathsf{br}) = (\mathsf{goal}(\gamma) \wedge \mathsf{belief}(\beta) \Rightarrow \mathsf{behavior}(\pi))$ *for any* $\mathsf{br} = (\gamma \leftarrow \beta \uparrow \pi) \in \mathcal{L}_{\mathcal{BR}}$*.*

Because behavioral rules, as defined in Def. 2, are interpreted in an operational context, the implications in the rule descriptions defined in Def. 4 do not hold in a classical logical way with respect to describing agent operation. However, if perceived actions $\delta \in \mathcal{L}_{\Delta}$ can be related to the behavioral descriptions $\pi \in \mathcal{L}_{\Pi}$, then abduction — which states that from an observation ψ and a rule $\phi \Rightarrow \psi$, the defeasible explanation ϕ can be abduced — can be used to infer the preconditions of behavioral rule descriptions. To relate action sequences to descriptions of behavior, a function is defined that maps behavioral descriptions to sets of *observable traces* of behavior by filtering out internal tests, which are taken to be unobservable, and branching observable traces at points where choice

occurs. The operator $\cup$ is standard set union, and $\circ : \wp(\mathcal{L}_\Delta) \times \wp(\mathcal{L}_\Delta) \longrightarrow \wp(\mathcal{L}_\Delta)$ is a non-commutative composition operator defined as $\Delta_1 \circ \Delta_2 = \{\ \delta_1\delta_2 \mid \delta_1 \in \Delta_1 \text{ and } \delta_2 \in \Delta_2\ \}$, where $\Delta_1, \Delta_2 \subseteq \mathcal{L}_\Delta$.

Definition 5 (observable trace function). *Let $\alpha \in \mathsf{Act}$, $\phi \in \mathcal{L}$ and $\pi \in \mathcal{L}_\Pi$. The function $\tau : \mathcal{L}_\Pi \longrightarrow \wp(\mathcal{L}_\Delta)$ is then defined as follows.*

$$\begin{aligned} \tau(\alpha) &= \{\alpha\} & \tau(\pi_1 + \pi_2) &= \tau(\pi_1) \cup \tau(\pi_2) \\ \tau(\phi?) &= \{\epsilon\} & \tau(\pi_1; \pi_2) &= \tau(\pi_1) \circ \tau(\pi_2) \end{aligned}$$

In order to abduce mental state preconditions of the logical rule descriptions in Def. 4 on grounds of observed actions, a relation must be established between an observed action sequence and the traces that represent the observable aspect of the behavior described in the behavioral description part of a rule. If every action of the agent is observed and the rules completely describe an agent's possible behavior, then an observed sequence of actions must be the (non-strict) prefix of an observable trace of behavior described by some $\pi \in \mathcal{L}_\Pi$.

Definition 6 (structural relations). *Let $\preccurlyeq \subseteq \mathcal{L}_\Delta \times \mathcal{L}_\Delta$ be the prefix relation on sequences $\delta, \delta' \in \mathcal{L}_\Delta$ and $\succcurlyeq \subseteq \mathcal{L}_\Delta \times \mathcal{L}_\Delta$ the suffix relation, defined as follows.*

$$\delta \preccurlyeq \delta' \text{ iff } \exists \delta'' \in \mathcal{L}_\Delta :\ [\delta' = \delta\delta''] \qquad \delta \succcurlyeq \delta' \text{ iff } \exists \delta'' \in \mathcal{L}_\Delta :\ [\delta' = \delta''\delta]$$

Note that δ is a non-strict prefix and suffix of itself iff δ'' is the empty action ϵ.

Defining different structural relations, as shown in [6], may allow for relating observed actions to observable traces also in the case that not every action is observed. It was proven that this leads to an increase in abduced explanations, and that the set of explanations inferred on grounds of complete observation is a subset of explanations in case of partial observation. In order to focus on the way contextual information can be used to facilitate the process of mental state abduction, it is assumed in this paper that observation is complete.

An agent's behavior is to be explained in terms of a description of its mental state, ie. the preconditions of rule descriptions defined in Def. 4. In order to refer to these preconditions, let the set $\mathcal{L}_\Omega$ with typical element ω be defined as $\mathcal{L}_\Omega = \{\ \mathsf{goal}(\gamma) \wedge \mathsf{belief}(\beta) \mid \gamma \in \mathcal{L}_\Gamma, \beta \in \mathcal{L}\ \}$. An explicit 'lifting' notation for functions is used, such that for any $f : D \longrightarrow D'$, the *lifted* version of f is ${}^\wp f : \wp(D) \longrightarrow \wp(D')$, such that for $\Phi \subseteq D$, ${}^\wp f(\Phi) = \{\ f(\phi) \mid \phi \in \Phi\ \}$.

An explanatory function is now defined that maps observed action sequences $\delta \in \mathcal{L}_\Delta$ to preconditions of rule descriptions of the type $\mathsf{rd} \in \mathcal{L}_{\mathcal{RD}}$, such that δ is a (partial) trace of the behavior 'implied' by the rule description.

Definition 7 (explanatory function). *The function $\chi : \mathcal{L}_{\mathcal{RD}} \longrightarrow \wp(\mathcal{L}_\Omega \times \mathcal{L}_\Delta)$ maps a rule description to a set of tuples of precondition and trace.*

$$\chi(\omega \Rightarrow \mathsf{behavior}(\pi)) = \{\ (\omega, \delta) \mid \delta \in \tau(\pi)\ \}$$

Let $\delta \in \mathcal{L}_\Delta$ be a percept and $\mathcal{RD} \subseteq \mathcal{L}_{\mathcal{RD}}$ a set of rule descriptions. The explanatory function $\mathsf{explain} : \mathcal{L}_\Delta \times \wp(\mathcal{L}_{\mathcal{RD}}) \longrightarrow \wp(\mathcal{L}_\Omega)$ *is then defined as follows.*

$$\mathsf{explain}(\delta, \mathcal{RD}) = \{\ \omega \mid \exists(\omega, \delta') \in {}^{\wp}\chi(\mathcal{RD}) : [\delta \preccurlyeq \delta']\ \}$$

Somewhat less formally, the explanatory function defined in Def. 7 states that the precondition of a rule description is in the set of explanations for a certain observed sequence of actions, if the observed sequence is a (non-strict) prefix of any trace of the behavioral description which is described in the postcondition of the rule description. This function definition is not meant to be computationally efficient, but it can be proven that $\mathsf{explain}(\delta\delta', \mathcal{RD}) \subseteq \mathsf{explain}(\delta, \mathcal{RD})$ for any $\delta, \delta' \in \mathcal{L}_\Delta$, allowing for an efficient implementation as the set of explanations and corresponding rules does not increase with cumulating coherent percepts.[1]

3 Agents Playing Games

Autonomous agents in a multi-agent system have their mental state and (inter)act in pursuit of their private goals, taking into account their beliefs, goals, capabilities and the means provided by their environment(s) [10]. An *agent-based game*, arguably should be more than a 'regular' multi-agent system, as the latter lacks particular qualities that a game might be required to have. When implementing game characters as autonomous agents, a designer gives away part of the behavioral control that a more traditional approach to character design provides [12]. In return, the daring move of the designer is rewarded with a plot that takes unexpected turns because of decisions made by the autonomous characters. However, there are certain aspects of the game's 'flow of events' that the game designer wants to ensure, without having to rely on providence or agents' inclination to partake in an interesting story.

In this section declarative game-related concepts are defined, inspired by organizational principles, that are used to illustrate our view on how an agent-based game can be designed that respects the storyline marked out by some designer. Moreover, the same concepts can be used as a guideline with respect to agents' expected behavior, as will be shown in Sect. 4.

3.1 A Declarative Game Specification

The general concept 'game' is hard to define, so that in the present approach a particular kind of game is considered, namely the *agent-based role-playing game*. Such a game is considered to be populated by virtual characters implemented as autonomous BDI-based agents, which play roles similar to the way actors do in a movie. Autonomous agents, however, may be allowed more freedom in the way they enact their role than movie actors are. Such role-enacting agents can be allowed to have private goals that conflict with or supercede those specified by their role, which will show in the behavior they exhibit [13,8].

[1] The authors thank Henry Prakken for this stronger version of their proofs in [6].

Most definitions of the concept 'role' recognize that a role comes with obligations, permissions, authority, or rights, which can pertain to actions that agents have at their disposition or states of the system. As such, roles describe behavior which can be expected of the role-enacting agent [14], and in recognition of this fact roles are defined in this paper to provide an agent with goals to achieve, information that is made available to the role-enacting agent, and a set of behavioral rules. These concepts correspond to roughly the Beliefs, Desires, and Intentions of the BDI-paradigm, and as such can form the basis for instantiation of the role-enacting agent. Note that the relation of role-derived goals and goal-directed rules is taken to be not necessarily one-to-one; multiple rules for a single goal can exist, or a conjunctive goal may be provided by the role where only rules for literal goals exist. To be able to refer to roles and other entities uniquely, a set of constants ID is introduced.

Definition 8 (role). *Let $\Gamma_{\leq} = (\Gamma, \leq_\Gamma)$ be an ordered set of goals $\Gamma \subseteq \mathcal{L}_\Gamma$, with $\leq_\Gamma \subseteq \mathcal{L}_\Gamma \times \mathcal{L}_\Gamma$ a partial order on Γ. Let $\mathcal{I} \subseteq \mathcal{L}$ be role-accessible information, and $\mathcal{BR} \subseteq \mathcal{L}_{\mathcal{BR}}$ a set of behavioral rules. A role R, identified by a unique identifier $r \in \mathsf{ID}$, is then defined as $\mathsf{R} = \langle r, \Gamma_{\leq}, \mathcal{I}, \mathcal{BR} \rangle$.*

It is left in the middle in this paper how role-enacting agents exactly play their role, leaving open the possibility that they (by design or by deliberation) ignore the goals their role prescribes. For technical reasons it is assumed, though, that the behavior of any agent is based on some rule provided by it's role. A typical role, featured in many games of the role-playing game (RPG) genre, is that of the *thief*. Unsurprisingly, the thief-role may provide the goal to take possession of a particular item by stealing it. Moreover, a thief could have the goal to steal the item whilst double-checking that nobody is near. If the thief assesses a particular situation to be risky, the goal to steal the item but also ensure that nobody is around might supercede the goal to just steal the item.

Roles do not per definition remain unchanged throughout a game. The context in which a role is enacted influences the way it should be enacted, and this context may change as things happen in the game. Autonomous agents can in principle be assumed to enact their role as they see fit, resulting in different types of behavior given the same role specification. Nevertheless, agents may be restricted in their actions by the norms of the agent society in which they operate. To formalize the norms that regulate behavior, a language of normative expressions $\mathcal{L}_\mathcal{N}$ is defined which captures *prima facie norms*, with typical element N, such that $\mathsf{N} ::= F(\alpha) \mid O(\alpha)$. The expression $F(\alpha)$ states that the action $\alpha \in \mathsf{Act}$ is forbidden, $O(\alpha)$ states that this action is obligatory.

In [15], prima facie norms are defined to be *norms [which] usually do not arise from actions, but arise in certain situations [and remain] valid as long as the situation in which they arise stays valid.* Scenes are taken to constitute the norm-governed context in which roles remain unchanged. The scene definition includes the roles figuring in the scene, a set of norms pertaining to the scene, and a set of literals denoting the initial state of the scene environment.

Definition 9 (scene). *Let $\mathcal{R}$ be a set of roles as defined in Def. 8, $\mathcal{N} \subseteq \mathcal{L}_{\mathcal{N}}$ a set of norms, and $\mathsf{E} \subseteq \mathsf{Lit}$ the initial environment state. Scene S, with unique identifier s, is then defined as $\mathsf{S} = \langle s, \mathcal{R}, \mathcal{N}, \mathsf{E} \rangle$.*

Take a scene in an RPG that features the thief and a store owner in some store where goods can be purchased. In this scene it is most likely forbidden to take items without paying for them, or to damage the merchandise.[2] Now take a scene in which the thief and the store owner are joined in the store by a city guard. The same norms may apply as in the previous scene, but the thief now gives priority to ensuring nobody is around before stealing anything because of the presence of the guard, whereas the store owner might be more at ease in knowing that the eyes of the law are keeping watch over her belongings.

It should be noted that norms can also be considered to be role-dependent, such that a particular set of norms applies only to some role (the permission to ignore a red traffic light, for example). This is not currently taken into account, but in the future could be incorporated to enrich the framework.

3.2 The Multi-Agent Game

A game is taken to be a composition of scenes. The way scenes are composed (the 'storyboard' of the game) is defined in a game specification, which identifies the scenes occurring in a game, and specifies when a specific scene makes a transition to another scene. Such transitions might depend on conditions being fulfilled with respect to the environment of the scene, on specific actions being (jointly) executed by agents, or even some condition becoming true with respect to agents' mental states. To have a system of agents obey this specification, scene transition has to be operationalized in the semantics of the agent system. Because a detailed presentation of how the scene transition is realized does not contribute to the scope of the present approach, this is left unspecified and the game is taken to simply be a set of scenes, of the type defined in Def. 9.

Agents in 2APL [10] are defined by a configuration, which specifies their mental state in terms of goals, belief, plans, and rules. In this paper, we do not commit ourselves to restrictions regarding the language in which the agents are implemented, but do require that the behavior of agents is somewhat in accordance with the declarative specification of the role they enact, which contains elements that can be directly related to elements of agents' mental states, such as goals, information (beliefs) and behavioral rules. Specifically, the following assumptions and restrictions are enforced.

– *Any agent in the multi-agent game behaves in accordance with some role, such that the behavior of agents in a scene is completely described by the behavioral descriptions which are part of the rules of the roles in that scene.*

[2] Note that the prima facie norms of $\mathcal{L}_{\mathcal{N}}$ do not allow for conditional statements, and it is therefore not possible to express statements such as the fact that it is obligatory to pay for an item after taking it.

- *Roles prescribe specific partially ordered goals, and the rules accompanying a role are taken to enable achievement of all these goals. However, it is not necessarily the case that every goal for which there exists some rule is part of the goals that the role prescribes.*
- *It is assumed that agents do not interleave plans, even if they have multiple goals. If an agent has adopted multiple goals and has selected a plan based on the application of a behavioral rule for one of its goals, it will not apply a new rule until its selected plan is completed.*

The first scene of the multi-agent game is determined by the game's initial state, and consecutive scenes are determined as the game evolves; ie. as agents act in pursuit of their goals and 'things happen' in the game. Because it is not in the interest of the topic at hand, which is explanation and prediction of agents' behavior in the context of some scene in a multi-agent role-playing game, the operational transition of configurations of the multi-agent game will not be presented formally. It is simply assumed that the game takes place in some (known) scene, which provides a guideline with respect to behavior that can be expected of the agents populating the scene, given that the behavior of each agent is based on some (known) role.

4 Explaining and Predicting Agent Behavior

Mental state abduction can be used to abduce the mental state of BDI-based agents whose actions can be observed. If these actions are performed in the context of a multi-agent game, then information about the scene of the game and the role which agents enact can improve the abduction process. If an agent's role is known, the set of rules the agent is taken to have at its disposition is reduced to the set of rules provided by the role.

4.1 Explaining Agent Behavior

In the approach to mental state abduction as described in Sect. 2 (and in [6] in more detail), the behavior of an agent is explained on grounds of all rules this agent can be assumed to have if context is not considered. In the present setting, only the rules which are ascribed to the agent on account of its role in a particular scene are considered in the explanatory process. This ensures that the explanations provided for its behavior are *contextually grounded*, and that the set of rules which need to be considered is restricted in size.

The role of the agent contains behavioral rules and a partially ordered set of goals. There might exist agents which dutifully pursue the goals their role prescribes, and others which don't care about their role in the least. To capture these aspects of *role conformance*, two refined versions of the explanatory function are defined. Because the role-prescribed goals do not necessarily have a one-to-one correspondence with the goals that form the head of behavioral rules, a relation between the two has to be established. The functions g and b

are defined which 'extract', respectively, the goals and beliefs from mentalistic explanations for behavior, such that for $\omega = (\mathsf{goal}(\gamma) \wedge \mathsf{belief}(\beta))$, it is the case that $g(\omega) = \{\gamma\}$ and $b(\omega) = \{\beta\}$. $\mathsf{Cn}(\Phi)$ denotes the closure of the set Φ under the consequence operator Cn, such that $\mathsf{Cn}(\Phi) = \{\phi \mid \Phi \models \phi\}$.

Definition 10 (loosely role-conformant explanation). *Let $\delta \in \mathcal{L}_\Delta$ be a percept and $\langle r, (\Gamma, \leq_\Gamma), \mathcal{I}, \mathcal{BR}\rangle$ a role, as defined in Def. 8. The function $\mathsf{explain}_{lrc}$ for loosely role-conformant explanation is then defined as follows.*

$$\begin{aligned}
&\mathsf{explain}_{lrc}(\delta, \langle r, (\Gamma, \leq_\Gamma), \mathcal{I}, \mathcal{BR}\rangle) = \ (\Omega, \leq_\Omega), \ \textit{where}\\
&\quad \Omega = \mathsf{explain}(\delta, {}^{\wp}\mathsf{desc}(\mathcal{BR})), \ \textit{and for any } \omega, \omega' \in \Omega\\
&\quad \leq_\Omega = \ \{(\omega, \omega') \mid [\mathsf{Cn}(g(\omega)) \not\subseteq \mathsf{Cn}(\Gamma)] \wedge [\mathsf{Cn}(g(\omega')) \subseteq \mathsf{Cn}(\Gamma)]\}\\
&\qquad \cup \ \{(\omega, \omega') \mid [\mathsf{Cn}(g(\omega)) \cap \mathsf{Cn}(\Gamma) = \emptyset] \wedge [\mathsf{Cn}(g(\omega')) \not\subseteq \mathsf{Cn}(\Gamma)]\} \cup \{(\omega, \omega)\}
\end{aligned}$$

A rule can be said to be relevant to a role, if the goal for which this rule applies is in the closure of the role-derived goals. Thus, the rules for goals ϕ and ψ are both relevant to a role that prescribes the goal $\phi \wedge \psi$, just as the rule for $\phi \wedge \psi$ is relevant to a role that prescribes ϕ and ψ independently. The function $\mathsf{explain}_{lrc}$ maps to a poset of explanations, where the explanations are ordered on grounds of an ordering that ranks explanations containing role-derived goals over those with goals that derive from behavioral rules only. Explanations which contain exclusively role-derived goals rank over those with some role-derived goals, which in turn rank over explanations without role-derived goals.

Definition 11 (strictly role-conformant explanation). *The definition of the function $\mathsf{explain}_{src}$ is based on $\mathsf{explain}_{lrc}$, but takes into account the order on Γ.*

$$\begin{aligned}
&\mathsf{explain}_{src}(\delta, \langle r, (\Gamma, \leq_\Gamma), \mathcal{I}, \mathcal{BR}\rangle) = \ (\Omega, \leq_\Omega)\\
&\quad \textit{where} \quad \mathsf{explain}_{lrc}(\delta, \langle r, (\Gamma, \leq_\Gamma), \mathcal{I}, \mathcal{BR}\rangle) = (\Omega, \leq'_\Omega), \ \textit{and for any } \omega, \omega' \in \Omega\\
&\qquad \leq_\Omega = \ \{\ (\omega, \omega') \mid \exists \gamma \in \mathsf{Cn}(g(\omega)), \exists \gamma' \in \mathsf{Cn}(g(\omega')) : [\gamma <_\Gamma \gamma'] \wedge\\
&\qquad\qquad \neg\exists \gamma \in \mathsf{Cn}(g(\omega)), \exists \gamma' \in \mathsf{Cn}(g(\omega')) : [\gamma' <_\Gamma \gamma] \ \} \cup \ \leq'_\Omega
\end{aligned}$$

Strictly role-conformant agents are taken to also obey the priority ordering on goals specified by their role, and the function for strictly role-conformant explanation $\mathsf{explain}_{src}$ accordingly takes this ordering into account as well. Because not all goals need to be explicitly ordered, it is defined that some explanation ω is preferred to ω' on grounds of $\mathsf{explain}_{src}$ if and only if some goal γ, derived from ω, has explicit priority over some goal γ', derived from ω', and no goal derived from ω' has explicit priority over any goal derived from ω.

Instead of conforming to their role, agents might *rebel* against their role. Also, as explained in [13], agents which are allowed to have private objectives along with role-derived objectives can enact their roles in a selfish or social manner. This could imply an ordering which is the reverse of that seen in loose role conformance, or even of strict role conformance. Although it is not further dealt with, the fact that our approach allows for modeling explicit rebellion and different types of role enactment deserves pointing out.

4.2 Predicting Agent Behavior

An observed and explained sequence of actions can be regarded as the performed part of some plan trace. Given that the goal for which the plan was selected is still 'active', the agent can be expected to perform the remaining actions, which are the suffix of the trace of which the observed actions are the prefix. In explaining an agent's behavior, it was defined that a description of the agent's mental state can be regarded as an explanation. When predicting the behavior of the agent with respect to actions it has been observed to perform, multiple (distinct) action sequences may be predicted based on different assumed mental states. A predictive function is defined, taking these aspects into account.

Definition 12 (predictive function). *Let $\delta, \delta' \in \mathcal{L}_\Delta$ be percepts, $\omega \in \mathcal{L}_\Omega$ a mental state description and $\mathcal{RD} \in \mathcal{L}_{\mathcal{RD}}$ a set of rule descriptions. The function* $\mathsf{predict} : \mathcal{L}_\Delta \times \mathcal{L}_{\mathcal{RD}} \longrightarrow \wp(\mathcal{L}_\Omega \times \mathcal{L}_\Delta)$ *is then defined as follows.*

$$\mathsf{predict}(\delta, \mathcal{RD}) = \{\ (\omega, \delta') \mid (\omega, \delta\delta') \in {}^{\wp}\chi(\mathcal{RD})\ \}$$

Agents in a norm-governed society can be assumed to take norms into account in choosing their actions, either by design or by deliberation [16]. Similar to the explanatory functions taking into account role conformance of the agent (Defs. 10 & 11), one can consider *norm obedience* when predicting agent behavior. The norms of $\mathcal{L}_\mathcal{N}$ were defined to possibly state about actions whether these are either forbidden (F) or obligatory (O). Informally, $F(\alpha)$ is taken to mean that the action $\alpha \in \mathsf{Act}$ is forbidden and that agents may be punished if they perform the action, whereas $O(\alpha)$ states that agents are obliged to perform action α and they may be punished if they do not perform it. Note that it is not defined what it means that the agent "*may* be punished", but the explanation that the behavior of the agent is somehow monitored (possibly by law-enforcing agents in the game), and that this monitoring is not infallible, should suffice.

Thus, it may occur that the agent performs a forbidden action, but gets away with it. The predicates forb and obl are defined on $\delta \in \mathcal{L}_\Delta$, such that

$$\mathcal{N} \models \mathsf{forb}(\delta) \ \textit{iff}\ \ \exists \alpha, \delta', \delta'' \in \mathcal{L}_\Delta : [(\delta = \delta'\alpha\delta'') \wedge (F(\alpha) \in \mathcal{N})]$$
$$\mathcal{N} \models \mathsf{obl}(\delta) \ \ \textit{iff}\ \ \exists \alpha, \delta', \delta'' \in \mathcal{L}_\Delta : [(\delta = \delta'\alpha\delta'') \wedge (O(\alpha) \in \mathcal{N})]$$

Based on the above, a predictive function is defined which takes norm obedience into account. This function predicts a sequence of actions on grounds of an observed sequence of actions and behavioral rules, and relates it to the presumed mental state which would account for observed behavior if it were the agent's actual mental state. Moreover, this predictive function takes into account that norms may exist which forbid or oblige the agent to perform specific actions, as expressed in the ordering that ranks pairs with an action sequence containing some obliged but no forbidden actions above all others, and pairs with sequences that contain some forbidden but no obliged actions below all others.[3]

[3] Note that a sequence with forbidden as well as obligatory actions is treated no differently than one that has only 'neutral' actions.

Definition 13 (norm-obedient prediction). *Let $\delta \in \mathcal{L}_\Delta$ be a percept, the tuple $\langle r, \Gamma_\leq, \mathcal{I}, \mathcal{BR}\rangle$ a role as defined in Def. 8 and $\langle s, \mathcal{R}, \mathcal{N}, \mathsf{E}\rangle$ a scene as defined in Def. 9. The predictive function $\mathsf{predict}_{no}$ is then defined as follows.*

$$\mathsf{predict}_{no}(\delta, \langle r, \Gamma_\leq, \mathcal{I}, \mathcal{BR}\rangle, \langle s, \mathcal{R}, \mathcal{N}, \mathsf{E}\rangle) = \ (\Theta, \leq_\Theta), \text{ where}$$
$$\Theta = \mathsf{predict}(\delta, {}^{\wp}\mathsf{desc}(\mathcal{BR})), \text{ and for any } (\omega, \delta), (\omega', \delta') \in \Theta$$
$$\leq_\Theta = \{\ ((\omega, \delta), (\omega', \delta'))\ |\ \mathcal{N} \models [\mathsf{obl}(\delta') \wedge \neg \mathsf{forb}(\delta')]\ \}$$
$$\cup \{\ ((\omega, \delta), (\omega', \delta'))\ |\ \mathcal{N} \models [\mathsf{forb}(\delta) \wedge \neg \mathsf{obl}(\delta)]\ \} \cup \{\ ((\omega, \delta), (\omega, \delta))\ \}$$

Considering role conformance and norm obedience is not restricted to explanation and prediction, respectively. Behavior which has been observed can give a guideline with respect to norm obedience, such that an agent which has been observed to consistently obey the norms in the past can be expected to do so in the future. Similarly, an agent which is assumed to conform to its role and is attributed some mental state on grounds of this assumption can be expected to set forth on its course of action and finish the trace that is in the explanation/-trace pair given by the predictive function. Prediction of ϵ — possibly indicating goal achievement — is outside of the scope of this approach, but very well worth further investigation. In Sect. 4.1 the remark was made that agents can explicitly rebel against their role. Similarly, agents might rebel against 'society', such that traces with forbidden actions are considered to be preferred by the agent.

4.3 The Observer

To explain and predict behavior, an abstract external Observer is proposed (in line with our approach in [6]) which perceives the atomic observable actions performed by agents, attempting to explain those actions in context of the game and making predictions about actions it expects agents to perform next. The Observer maintains a model of each of the agents it observes, which contains the role the Observer attributes to the agent and a sequence of actions the agent has been observed to perform, along with explanations and predictions based on observed behavior in context of the attributed role.

Definition 14 (agent model). *Let $r \in \mathsf{ID}$ name a role of the type in Def. 8, $\delta \in \mathcal{L}_\Delta$ a list of actions, $\Omega \subseteq \mathcal{L}_\Omega$ and $\Theta \subseteq \mathcal{L}_\Omega \times \mathcal{L}_\Delta$ a set of explanations and predictions. An agent model, with identifier $i \in \mathsf{ID}$, is then $\mathsf{A} = \langle i, r, \delta, \Omega, \Theta\rangle$.*

The Observer is assumed to have perfect observation of the environment and the actions agents perform. In many games, the roles of characters are evident from their external characteristics. The role might be indicated by the color of a suit, or simply by a label hovering over the character. In the following, it is assumed that agent i's role r can be deduced from the state of the environment, such that $\mathsf{E} \models \mathsf{enacts}(i, r)$. If scene transitions are taken to depend only on changes in the environment, the Observer always knows the scene in which the game takes place if it is made aware of the initial scene when the game starts, and is correct about the roles it attributes to agents.

Relaxing these assumptions — either by introducing more uncertainty on part of the Observer by design or because the game does not allow for perfect observation of the environment, scene transitions, or agents' roles — leads to interesting scenarios. Instead of just performing mental state abduction, the Observer is forced to perform role abduction and/or scene abduction as well. If observation of the environment or agents' actions is imperfect as well, yet more defeasibility is introduced. Given that our goal is to allow for designing agent-based game characters which have uncertainty about other characters' mental states, it is not in our interest to introduce any more uncertainty than necessary. Partial observation was discussed in [6], but here perfect observation is assumed.

Definition 15 (Observer). *Let $\mathcal{G}$ be a set of scenes as defined in Def. 9, and $s \in \mathsf{ID}$ a scene identifier such that $\langle s, \ldots \rangle \in \mathcal{G}$. The set of literals $\mathsf{E} \subseteq \mathsf{Lit}$ is the environment state and for every (perceived) agent i, A_i is an agent model. The Observer is then defined as $\langle \mathcal{G}, s, \mathsf{E}, \{\mathsf{A}_i, \ldots, \mathsf{A}_j\} \rangle$.*

The Observer as defined in this approach is an abstract entity, which serves to illustrate the explanatory and predictive process ultimately to be used by agents that observe other agents' behavior in some environment. For this reason the details of how the Observer configuration evolves with successive action observations and scene transitions are left to the imagination of the reader, and instead the focus is on the procedures defined in Sect. 4.1 and 4.2. Given a single sequence of observed actions for some agent i, the Observer can explain as well as predict this sequence of actions. Prop. 1 shows that each explanation — in terms of an agent's mental state — is accompanied by a matching prediction.

Proposition 1 (explanation matches prediction). *Let $\langle i, r, \delta, \Omega, \Theta \rangle$ be an agent model, and $\langle r, \Gamma_{\leq}, \mathcal{I}, \mathcal{BR} \rangle$ the role of agent i. Given $^{\wp}\mathsf{desc}(\mathcal{BR}) = \mathcal{RD}$, $\mathsf{explain}(\delta, \mathcal{RD}) = \Omega$, and $\mathsf{predict}(\delta, \mathcal{RD}) = \Theta$, it holds that every explanation has a corresponding prediction, such that $\forall \omega \in \Omega : [\exists \theta \in \Theta, \delta' \in \mathcal{L}_\Delta : [\theta = (\omega, \delta')]]$.*

Proof. Def. 7 and Def. 12 show that explain *and* predict *are both based on $^{\wp}\chi$. In case of $\mathsf{explain}(\delta, \mathcal{RD}) = \Omega$, some $\omega \in \Omega$ iff $\exists (\omega, \delta'') \in {}^{\wp}\chi(\mathcal{RD}) : [\delta \preccurlyeq \delta'']$. For $\mathsf{predict}(\delta, \mathcal{RD}) = \Theta$, some $(\omega, \delta') \in \Theta$ iff $(\omega, \delta\delta') \in {}^{\wp}\chi(\mathcal{RD})$. It follows from Def. 6 that if $\delta \preccurlyeq \delta''$, then $\delta'' = \delta\delta'$ for some δ' (possibly $\delta' = \epsilon$), and therefore $(\omega, \delta') \in \Theta$. By definition of $\forall$, the proposition holds for $\Omega = \emptyset$.* □

Corollary 1. *Evidently, every prediction is matched by an explanation.* □

Note that Prop. 1 and Coroll. 1 extend to the functions $\mathsf{explain}_{lrc}$, $\mathsf{explain}_{src}$ and $\mathsf{predict}_{no}$, as these are directly based on explain and predict. This is a very welcome fact, because it ensures that for every explanation a corresponding prediction can be made, and vice versa, also in the case of the context-dependent explanatory and predictive functions. Based on role-conformant explanation some explanation may come out as 'top-ranked'. This can be considered the best explanation for the agent's behavior, and corresponding predicted behavior be regarded as the best prediction.

It can occur that traces have overlapping segments. In such a case, the possibility exists that the Observer is able to explain a sequence of actions, but at a certain point observes an action that can neither be considered coherent with the currently presumed trace, nor as being the start of a new trace. This situation is visualized in Fig. 1, where δ' is the suffix of some trace $\delta\delta'$, as well as the prefix of another trace $\delta'\delta''$. Let α be the first action of δ'', and $\delta''' = \delta\delta'\alpha$. If the Observer explains $\delta\delta'$ as a coherent whole, then after perceiving α, it may be that $\mathsf{explain}(\delta''', \mathcal{RD}) = \emptyset$ because δ''' is not the prefix of any trace. It may, however, also be that α itself is not the prefix of any trace, such that $\mathsf{explain}(\alpha, \mathcal{RD}) = \emptyset$.

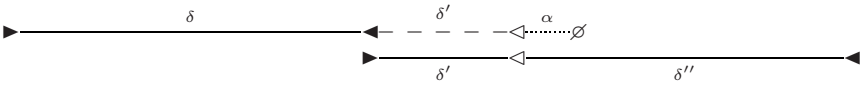

Fig. 1. Two traces, $\delta\delta'$ and $\delta'\delta''$, with an overlapping part δ'. The start of actual traces is denoted by ► and the end by ◄, explainable (segments of) potential traces end with ◁, and ⌀ denotes a non-matching segment (ie. failure of explanation).

If the situation sketched in the previous paragraph occurs and explanation fails, then the Observer can backtrack along δ', starting at the end, until it finds a suffix $\delta'''' \succcurlyeq \delta'$ that can be explained in coherence with the last observed action α, such that $\mathsf{explain}(\delta''''\alpha, \mathcal{RD}) \neq \emptyset$. Given the assumption that agents are assumed to be able to completely execute their plans, the maximum overlap of any two traces of an agent's plans can be computed and used to give a measure of the maximum amount of backtracking the Observer has to perform. Let $\mathsf{len} : \mathcal{L}_\Delta \longrightarrow \mathbb{N}$ be a function that maps a percept to its length, such that $\mathsf{len}(\alpha_1, \ldots, \alpha_n) = n$, and let binds be predicate denoting that some sequence δ 'binds' the sequences δ' and δ'' together with overlapping action sequences, defined as

$$\mathsf{binds}(\delta, \delta', \delta'') \qquad \textit{iff} \qquad [(\delta \succcurlyeq \delta' \wedge \delta \preccurlyeq \delta'') \vee (\delta \succcurlyeq \delta'' \wedge \delta \preccurlyeq \delta')]$$

Given the definitions of len and binds, let $\mathsf{overlap} : \mathcal{L}_\Delta \times \mathcal{L}_\Delta \longrightarrow \mathbb{N}$ be a function that computes the overlap between two (distinct) action sequences, defined as

$$\mathsf{overlap}(\delta', \delta'') = \begin{cases} \mathsf{len}(\delta) \textit{ if } \exists \delta \in \mathcal{L}_\Delta : [\mathsf{binds}(\delta, \delta', \delta'') \wedge (\delta' \neq \delta'')], \textit{ and} \\ \qquad\qquad \neg \exists \delta''' \in \mathcal{L}_\Delta : [\mathsf{binds}(\delta''', \delta', \delta'') \wedge (\mathsf{len}(\delta''') > \mathsf{len}(\delta))] \\ 0 \quad \textit{otherwise} \end{cases}$$

Proposition 2 (backtrack with maximum trace overlap). *If agents always finish their plans, given agent model* $\langle i, r, \delta, \Omega, \Theta \rangle$ *and role* $\langle r, \Gamma_{\leq}, \mathcal{I}, \mathcal{BR} \rangle$, *where* $^{\wp}\mathsf{desc}(\mathcal{BR}) = \mathcal{RD}$ *and* $\delta = \delta'\alpha_1 \cdots \alpha_n$, *such that* $\mathsf{explain}(\delta, \mathcal{RD}) = \emptyset$, $\mathsf{explain}(\delta'\alpha_1 \cdots \alpha_{n-1}, \mathcal{RD}) \neq \emptyset$, *and* δ' *is the complete trace of some executed plan, there exists a non-empty suffix* $\delta'' \succcurlyeq \delta$ *such that* $\mathsf{explain}(\delta'', \mathcal{RD}) \neq \emptyset$ *and* $\mathsf{len}(\delta'')$ *is smaller than or equal to one, plus the maximum overlap between any two traces of any plan which is part of the rules in* $\mathcal{BR}$.

Proof. *Let* $\Delta = \bigcup\{\ \tau(\pi) \mid (\gamma \leftarrow \beta \uparrow \pi) \in \mathcal{BR}\ \}$ *be the set of all (finite) observable traces of all plans part of the rules prescribed by the role. Because* δ' *is a complete plan trace,* $\mathsf{explain}(\delta', \mathcal{RD}) \neq \emptyset$ *and* $\delta' \in \Delta$*. Let* $\delta'' \succcurlyeq \delta$ *such that* δ'' *is the prefix of a trace of the agent's latest plan and it holds that* $\delta'' \succcurlyeq \alpha_1 \cdots \alpha_n$*. Then either* $\exists \delta''' \in \mathcal{L}_\Delta : [(\delta''' \preccurlyeq \alpha_1 \cdots \alpha_n) \wedge (\delta'\delta''' \in \Delta)]$ *such that the actual 'old' trace* δ' *is also the prefix of a misleading 'false' trace* $\delta'\delta'''$*, or not. If not, then* $\delta = \delta'\alpha$ *such that* $\alpha \succcurlyeq \delta$*,* $\mathsf{explain}(\alpha) \neq \emptyset$*, and* $\mathsf{len}(\alpha) = 1$*.*

If $\exists \delta''' \in \mathcal{L}_\Delta : [(\delta''' \preccurlyeq \alpha_1 \cdots \alpha_n) \wedge (\delta'\delta''' \in \Delta)]$*, then* δ''' *is the suffix of a 'false' trace* $\delta'\delta'''$ *and the strict prefix of* $\alpha_1 \cdots \alpha_n$*, such that* $\mathsf{len}(\delta''') < n$*. The 'new' trace, of which* δ'' *is the prefix, is started somewhere after the plan of which* δ' *is a complete trace has finished, such that* $\delta'' \succcurlyeq \delta'\alpha_1 \cdots \alpha_n$*. If the 'new' trace of which* δ'' *is the prefix is not started directly after* δ'*, because inbetween a complete trace of yet another plan was executed which together with* δ' *could be matched to a misleading trace, then* δ'' *is a strict suffix of* $\alpha_1 \cdots \alpha_n$*, such that* $\mathsf{len}(\delta'') > n$*. If* δ'' *is started directly after* δ'*, then the sequence* $\delta'\delta'''$ *and the sequence* $\delta'' = \alpha_1 \cdots \alpha_n$ *have an overlap of* $n-1$*, which is the overlap of* $\delta'\delta'''$ *and the trace of which* δ'' *is the prefix, such that* $\mathsf{len}(\delta'') = n$*.*

Let x *be the maximum overlap of any two traces* $\delta_1, \delta_2 \in \Delta$*, such that* $(\mathsf{overlap}(\delta_1, \delta_2) = x) \wedge (\neg\exists \delta_3, \delta_4 \in \Delta : [(\mathsf{overlap}(\delta_3, \delta_4) = y) \wedge (y > x)])$*. Given that the trace* $\delta'\delta'''$ *and the 'new' trace of which* δ'' *is the prefix are both in* Δ *and have an overlap of* $n-1$*, it holds that* $0 \leq n-1 \leq x$*.* □

Prop. 2 can be guaranteed if the Observer does not 'forget' any percepts and agents complete their plans. Especially the latter condition is unmaintainable in certain environments. If agents can drop their plans, 'freak' scenarios can arise with respect to explanation/prediction of behavior. Take, for example, the case where $\alpha_1 \cdots \alpha_n$ is a plan trace, but the individual actions $\alpha_1, \ldots, \alpha_n$ are also the initial actions of individual plans. If an agent selects those plans in order, executing only the first action and then dropping the plan, the resulting sequence is indistinguishable from the trace.[4] In this case Prop. 2 does not hold, because it relies on the assumption that plans are always completed. It is still possible in this case to give some measure of backtracking, though, because if traces are finite and actions perfectly observable then Coroll. 2 still applies.

Corollary 2. *If agents can drop their plans before they are completed, then the Observer backtracks at most up to the length of the longest existing plan trace to find an explanation if* $\mathsf{explain}(\delta, \mathcal{RD}) = \emptyset$ *occurs.*

Proof. *Let* $\alpha_1 \cdots \alpha_n \in \Delta$ *be the longest trace of any plan part of the rules prescribed by some agent's role. As* $\mathsf{explain}(\delta, \mathcal{RD}) = \emptyset$*, it must be that* $\exists \delta' \in \mathcal{L}_\Delta : [\delta' \succcurlyeq \delta]$ *and* δ' *is the prefix of the agent's current plan. In worst case,* $\delta = \delta''\alpha_1 \cdots \alpha_n$ *for some* δ''*, such that* $\mathsf{explain}(\delta''\alpha_1 \cdots \alpha_{n-1}) \neq \emptyset$*. After backtracking* $\mathsf{len}(\alpha_1 \cdots \alpha_n) = n$ *actions, Observer finds* $\mathsf{explain}(\alpha_1 \cdots \alpha_n) \neq \emptyset$*.* □

[4] One might ask whether explaining and predicting behavior has any benefit at all if such situations abound in some scenario, but that is not the point now.

5 Example

To illustrate the present approach, an example inspired by the popular role-playing game Oblivion [17] is introduced. For reasons of presentation and space conservation some shorthand notation will be used. Lowercase predicate arguments represent ground atoms, and uppercase arguments represent variables. Our propositional language of course does not allow for variables, and therefore these are to be interpreted as a finite number of ground expressions, as should be clear from the context in which the notation is used. Spatial environments require moving around, and therefore $\mathsf{goto}(Loc)$ is defined, where the variable Loc stands for any valid location in the environment. The construct $(\phi?;\pi)+(\neg\phi?;\pi')$ is to be read as `if` ϕ `then` π `else` π'.

$$\mathsf{goto}(Loc) \equiv (\neg\mathsf{nearby}(Loc)?;\mathsf{walk\_towards}(Loc)) + (\mathsf{nearby}(Loc)?)$$

It is assumed here that, for some specific target location loc, $\mathsf{walk\_towards}(loc)$ is a single external action — therefore also observable as such — which brings the character nearby to loc.

The scene S takes place in a store and features 'thief' and 'store owner' roles. The norm in this scene forbids stealing any item, as expressed in shorthand notation using the capitalized $Item$, such that $\mathsf{S} = \langle s, \{\mathsf{R}_t, \mathsf{R}_{so}\}, \{F(\mathsf{steal}(Item))\}, \mathsf{E}\rangle$. The 'thief' role prescribes the goal to have a particular item of interest, such that $(\gamma = \mathsf{have}(item))$, and provides rules to achieve this goal. Also, rules for exploring the store (by making sure that all cabinets have been inspected) are provided.

$$\begin{aligned}
\mathsf{R}_t &= \langle thief, (\{\mathsf{have}(item)\}, \{(\gamma,\gamma)\}), \mathcal{I}, \{\mathsf{br}_{1_t}, \mathsf{br}_{2_t}, \mathsf{br}_{3_t}, \mathsf{br}_{4_t}\}\rangle \\
\mathsf{br}_{1_t} &= \mathsf{have}(item) \leftarrow \mathsf{distracted}(owner) \wedge \mathsf{in}(Cabinet, item) \uparrow \\
&\qquad \mathsf{goto}(Cabinet); \mathsf{open}(Cabinet); \mathsf{steal}(item); \mathsf{close}(Cabinet) \\
\mathsf{br}_{2_t} &= \mathsf{have}(item) \leftarrow \neg\mathsf{distracted}(owner) \uparrow \mathsf{goto}(owner); \mathsf{distract}(owner) \\
\mathsf{br}_{3_t} &= \mathsf{ensured}(safety) \leftarrow \neg\mathsf{nearby}(Person) \uparrow \mathsf{double_check_if_nearby}(Person) \\
\mathsf{br}_{4_t} &= \mathsf{explored}(store) \leftarrow \neg\mathsf{inspected}(cabinet_1) \uparrow \\
&\qquad \mathsf{goto}(cabinet_1); \mathsf{inspect}(cabinet_1) \\
&\qquad\qquad \vdots \\
\mathsf{br}_{m_t} &= \mathsf{explored}(store) \leftarrow \neg\mathsf{inspected}(cabinet_n) \uparrow \\
&\qquad \mathsf{goto}(cabinet_n); \mathsf{inspect}(cabinet_n)
\end{aligned}$$

The 'store owner' role R_{so} is left unspecified, except that it is stated she wants to protect her merchandise. In this paper the use of procedural rules has not been discussed, but for the sake of the example we (informally) allow them here, if only used for goal generation in response to events. If a customer breaks an object in the store, the perception of this fact prompts the store owner to adopt the goal to demand money from the culprit. This high-level approach focusing on mentalistic concepts already remedies shortcomings in scripted character behavior in a natural way. In a similar scene in Oblivion it is possible, for example,

to behave asocially in the store without repercussions, because the store owner is only scripted to react to theft, and not to other norm violations. We believe such defective behavior to be typical of games with characters that are scripted with a focus on reactive behavior, foregoing the underlying mental states, as in dynamic social settings the complexity of such scripts grows out of hand fast.

The scene S transitions to some new scene S' upon entry of a city guard, as mentioned in Sect. 3.1, such that $\mathsf{S}' = \langle s', \{\mathsf{R}'_t, \mathsf{R}'_{so}, \mathsf{R}_{cg}\}, \{F(\mathsf{steal}(\mathit{Item}))\}, \mathsf{E}'\rangle$. The 'city guard' role R_{cg} is left unspecified, but it should suffice to say that the guard may merely have come into the store to buy some item or chat with the store owner. If the guard becomes aware that someone is breaking the law (possibly he observes the thief stealing the item), he may come into action and arrest the perpetrator. In the new scene the thief can be expected to show more cautious behavior because of the presence of the guard. This change in the thief's behavior is illustrated by a change in the thief's role specification, such that $\mathsf{R}'_t = \langle \mathit{thief}', (\{\gamma, \gamma'\}, \{(\gamma, \gamma), (\gamma', \gamma'), (\gamma, \gamma')\}), \mathcal{I}, \{\mathsf{br}_{1_t}, \mathsf{br}_{2_t}, \mathsf{br}_{3_t}, \mathsf{br}_{4_t}\}\rangle$. In this slightly changed role specification, it still is the case that $\gamma = \mathsf{have}(\mathit{item})$, but there is another role-prescribed goal $\gamma' = \mathsf{have}(\mathit{item}) \wedge \mathsf{ensured}(\mathit{safety})$ for which it is the case that $\gamma' >_\Gamma \gamma$. The thief's more cautious enactment of his changed role could show through in his behavior, and use of role-based explanation (or prediction) on this same grounds would reflect such as well.

Various possibilities exist for elaboration, but here we take the liberty to skim over subtleties. More interesting is it to see how the Observer comes into play. Let $\mathcal{G}$ be the scenes of the game, and $\mathsf{A}_{gent} = \langle \mathit{gent}, \mathit{thief}, \mathsf{walk\_towards}(\mathit{cabinet}_1)\rangle$ the model of some agent called *gent*, such that the Observer has observed *gent*, in its role of *thief*, to perform the action of walking towards a certain cabinet. Let the Observer configuration for the first scene be $\langle \mathcal{G}, s, \{\mathsf{A}_{gent}\}\rangle$. Given the rules $\mathcal{BR}_t$ for the *thief* role, which the Observer deduces to be enacted by *gent* from the initial state of the environment of s and has knowledge of as part of the roles for scene s in $\mathcal{G}$, $\mathsf{explain}(\mathsf{walk\_towards}(\mathit{cabinet}_1), {}^{\wp}\mathsf{desc}(\mathcal{BR}_t))$ maps to a set of explanations $\Omega = \{\omega_1, \omega_2\}$, such that $g(\omega_1) = \{\mathsf{goal}(\mathsf{have}(\mathit{item}))\}$ and $g(\omega_2) = \{\mathsf{goal}(\mathsf{explored}(\mathit{store}))\}$. Based on role-conformant explanation, either loose or strict, $\omega_1 > \omega_2$ because having the item is a role-derived goal.

Prop. 1 states that every explanation is matched by a prediction. For the explanation ω_1, the tuple $(\omega_1, [\mathsf{open}(\mathit{Cabinet}), \mathsf{steal}(\mathit{item}), \mathsf{close}(\mathit{Cabinet})])$ (with the percept in Prolog-style list notation for readability) is in the set of predictions. Given the small scenario and limited set of rules, this is the only possible prediction for the goal of having the item, given the previously observed action and the assumption plans are completed. Should we allow for dropping of plans and belief revision, it may happen that the thief who *does* have the goal to have the item in his possession, after walking towards the cabinet the agent comes to believe the owner is not distracted, such that he does not open the cabinet but instead walks up to the owner and attempts to distract him first.

The thief can be expected not to be norm-obedient, such that the prediction that he will open the cabinet and steal the item comes out as a good one, which is plausible in the given context. Assuming the thief actually *is* norm-obedient

(which would be plausible in the scene S' where the guard is also present) gives a different picture. In that case another explanation for walking towards the cabinet might be considered best if the corresponding predicted action sequence doesn't contain any forbidden actions. In our example only the goal to explore the store qualifies, but in a more extensive case this could include the goal to choose and purchase some item located in the cabinet.

The example in this section is inspired by an actual commercial role-playing game. It serves mainly to illustrate some of the focal points of the approach presented in this paper, and is necessarily limited in its scope and detail. Nevertheless, we hope it to be sufficiently developed to convince the reader of the fact that the high-level concepts of organizational modeling and agent programming apply transparently to the rich domain of role-playing games. Moreover, the use of high-level social/intentional concepts has the additional benefit that these concepts can ideally be used for modeling, programming, and inter-character explanation and prediction of behavior.

6 Conclusion and Future Work

In this paper mental state abduction in the context of an agent-based game was described. A declarative game specification based on organizational principles such as roles, norms, and scenes, was introduced, and it was mentioned how it can be employed to have a system of autonomous agents behave in accordance with an intended storyline. An abstract Observer was said to observe the behavior of agents and provide explanations that take into account role-conformant behavior, making the abduction process more efficient because it is based on a subset of rules, and ensuring that explanations are relevant to context. The Observer can also predict agents' future actions based on previously observed behavior, taking norm-obedience into account if the situation warrants this assumption. Role-conformant explanation and norm-obedient prediction have been shown to be complementary.

Future research should focus on explicitly taking models of the environment and agents' presumed mental states into account in the abductive process. Depending on the nature of the environment, it could be possible for an observer, be it an abstract entity or situated agent, to actively check whether specific (predicted) actions are possible, or whether an agent has achieved its goal or can be inferred to have some particular belief. Investigating the use of even richer social context does also seem a viable direction. Such scenarios could be where norms hold for scenes but also for roles (and possibly conflict), violation (obedience) of norms is quantitatively sanctioned (rewarded) leading to a more fine-grained measure of prediction, and where uncertainty regarding role enactment or other components is introduced. Finally, the path of formally investigating the multi-agent game as an operational system seems promising. The idea of a game of only software agents without human involvement might seem a tad far-fetched, but its usefulness becomes apparent if such a game is regarded as being played by agents (without direct human involvement) as a socially believable backdrop

to the human player's actions in the game. Should agent-based virtual characters be required to attribute mental states to human players on grounds of their observed actions, a desideratum if they are to interact autonomously with said players, then some other (qualitative or quantitative) model of behavior would have to be employed in abducing their mental states, as players cannot be expected to follow fixed action traces.

References

1. Loyall, A.B.: Believable Agents. PhD thesis, Carnegie Mellon University (1997)
2. Laird, J.E.: It knows what you're going to do: Adding anticipation to a Quakebot. In: AGENTS (2001)
3. Afonso, N., Prada, R.: Agents that relate: Improving the social believability of non-player characters in role-playing games. In: Stevens, S.M., Saldamarco, S.J. (eds.) ICEC 2008. LNCS, vol. 5309, pp. 34–45. Springer, Heidelberg (2008)
4. Albrecht, D.W., Zukerman, I., Nicholson, A.E.: Bayesian models for keyhole plan recognition in an adventure game. User Modeling & User-Adapted Interaction 8(1-2), 5–47 (1998)
5. Goultiaeva, A., Lespérance, Y.: Incremental plan recognition in an agent programming framework. In: Proceedings of PAIR (2007)
6. Sindlar, M.P., Dastani, M.M., Dignum, F., Meyer, J.-J.Ch.: Mental state abduction of BDI-based agents. In: Baldoni, M., Son, T.C., van Riemsdijk, M.B., Winikoff, M. (eds.) DALT 2008. LNCS (LNAI), vol. 5397, pp. 110–125. Springer, Heidelberg (2009)
7. Scott, B.: Architecting a Game AI. In: AI Game Programming Wisdom, pp. 285–289. Charles River Media, Hingham (2002)
8. Dignum, V.: A Model for Organizational Interaction. PhD thesis, SIKS Dissertation Series (2004)
9. Coutinho, L.R., Sichman, J.S., Boissier, O.: Modeling organization in MAS. In: SEAS (2005)
10. Dastani, M.: 2APL: A practical agent programming language. Autonomous Agents and Multi-Agent Systems 16, 214–248 (2008)
11. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley-Interscience, Hoboken (2007)
12. Millington, I.: Artificial Intelligence for Games. Morgan Kaufmann, San Francisco (2006)
13. Dastani, M., Dignum, V., Dignum, F.: Role-assignment in open agent societies. In: Proceedings of AAMAS (2003)
14. Dastani, M.M., van Riemsdijk, M.B., Hulstijn, J., Dignum, F.P.M., Meyer, J.-J.Ch.: Enacting and deacting roles in agent programming. In: Odell, J.J., Giorgini, P., Müller, J.P. (eds.) AOSE 2004. LNCS, vol. 3382, pp. 189–204. Springer, Heidelberg (2005)
15. Dignum, F.: Autonomous agents with norms. Artificial Intelligence and Law 7(1), 69–79 (1999)
16. Castelfranchi, C., Dignum, F., Jonker, C.M., Treur, J.: Deliberative normative agents: Principles and architecture. In: Jennings, N.R. (ed.) ATAL 1999. LNCS, vol. 1757, pp. 364–378. Springer, Heidelberg (2000)
17. Bethesda Game Studios: The Elder Scrolls IV: Oblivion (2006)

Correctness Properties for Multiagent Systems

Munindar P. Singh[1] and Amit K. Chopra[2]

[1] North Carolina State University, Raleigh, USA
singh@ncsu.edu
[2] Università degli Studi di Trento, Trento, Italy
akchopra.mail@gmail.com

Abstract. What distinguishes multiagent systems from other software systems is their emphasis on the interactions among autonomous, heterogeneous agents. This paper motivates and characterizes correctness properties for multiagent systems. These properties are centered on commitments, and capture correctness at a high level. In contrast to existing approaches, commitments underlie key correctness primitives understood in terms of meaning; for example, commitment *alignment* maps to *interoperability*; commitment *discharge* maps to *compliance*. This paper gives illustrative examples and characterizations of these and other properties. The properties cover the specification of the principal artifacts—protocols, roles, and agents—of an interaction-based approach to designing multiagent systems, and thus provide the formal underpinnings of the approach.

1 Introduction

Interaction is the key distinguishing feature of multiagent systems. We investigate the science of interaction as it underlies the engineering of multiagent systems whose constituent agents are *heterogeneous* (independently designed) and *autonomous* (independently motivated). In such systems, the internal workings of the agents take backstage to the interactions among them.

We begin from a simple yet profound question: *How may we treat interactions as first-class citizens in modeling and analyzing multiagent systems?* The usual objectives of engineering—modularly specifying, developing, composing, verifying, and validating parts—apply for interactions just as for traditional software approaches. However, existing solutions, which are designed for components such as objects, do not readily lift to interactions: an interaction somehow must simultaneously accommodate more than one perspective. Thus, importantly, the novelty of the interactive setting yields fresh and crucial technical challenges, which offer a great opportunity for multiagent systems research.

Of the many applications of multiagent systems, those in *cross-organizational* business processes provide the happy mix of practical value, theoretical subtlety, and opportunity (in the form of interest in industry) that our research community needs to sustain this research effort. Cross-organizational processes fundamentally differ from conventional software in that they are naturally modeled via interactions among *heterogeneous* and *autonomous* agents [1]. The interactions of interest are of an arms-length nature, and thus naturally understood as *communications*. In our study, we assume the existence

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 192–207, 2010.

of suitable approaches for the transmittal of information and therefore concentrate on communication understood at the level of meaning.

To engineer a multiagent system based on interactive principles presupposes a notion of the correctness of interactions among agents—in particular, here, of communications. Given such a notion, we ask if an agent is *compliant* with its expected behavior. Further, we can ask if the given agents are *interoperable* meaning that they are able to work together as expected. We can ask the above questions from the perspective of the system as a whole or of any of the participants. To formalize interoperability and compliance in interactive terms requires that we develop a theory of types using which we might modularize communications into *protocols*. We might then create repositories of protocols; determine if one protocol refines another or aggregates two or more protocols; modularly validate the protocols; modularly verify agents with respect to each relevant protocol; and so on. Notice that interfaces in object-oriented computing correspond to protocols and support constructs such as refinement and aggregation as well as the usual forms of type inference.

1.1 Approach

The meaning of an interaction lies at the crux of the question of its correctness. When we think at levels above the transmission of information, the meaning of communication is grounded in the relationships among the parties involved. Communication then is based on conventions by which such relationships are created, progressed (or otherwise altered), and ended. We concentrate on the contractual relationships expressed through the notion of commitments. A *commitment* involves a debtor, a creditor, an antecedent, and a consequent; it is represented as $\mathsf{C}(debtor, creditor, antecedent, consequent)$. Roughly, the debtor stakes a claim or makes a promise to the creditor about the specified consequent provided that the antecedent holds. Commitments naturally express the *what* of business relationships, and minimally constrain the *how*. For example, a commitment to pay for goods received may be discharged by paying directly, or delegated to someone who would discharge or delegate it, and so on (for any finite sequence of delegations).

In our approach, a *protocol* specifies business interactions primarily by stating how messages affect the participants' commitments. For example, returning purchased goods unopened may release the buyer from a commitment to pay. Thus many possible enactments may result from the same protocol. This is how commitments yield both rigor and flexibility. Because of its naturalness, the commitment-based approach has attracted the attention of finance and health care industry groups [2].

Protocols are interfaces: they constrain how agents interact, not how they are implemented. Protocols are *doubly* modular: in terms both of functionality and autonomy. For example, for functionality, an ORDER protocol between a customer and a merchant would specify only interactions dealing with order placement, leaving other functionalities to separate protocols, e.g., one for INVENTORY FULFILLMENT. Our approach enables composing protocols to yield more complex protocols, of enhanced functionality. Further, for autonomy, ORDER would specify the interactions, leaving to each the autonomous decision making of whether and how to interact, which could depend on

its goals [3]. We define a *process* as the aggregation of the behaviors of the parties involved, including both their interactions and their local reasoning.

To model a process, we identify the protocols using which the different participants interact [1]. For example, a merchant and a customer may interact with each other using a NEGOTIATION protocol; the merchant, customer, and payment agency may interact via an ESCROW protocol; and, the merchant, customer, and shipper may interact through some specialized LOGISTICS protocol. When each participant acts according to its local reasoning but respecting the stated protocols, they jointly enact a multiparty business process. The contractually significant parts of the process would have been encoded in the commitments specified in the protocols; the other parts may feature only in the local policies of the participants and need not be visible externally. An agent's policies could be geared to optimize its outcomes. For example, policies would help decide what item to order, what price to quote, and so on.

The above approach obviates defining a monolithic global flow that specifies the actions of each party. Each protocol could be refined to capture additional requirements, e.g., adding receipts or guarantees to SHIPPING or PAYMENT to produce new refined protocols. Protocols can involve more than two parties; in typical usage, one partner would play multiple roles in multiple protocols [4]. For example, a purchase process may be defined as a composition of ORDER, SHIPPING, and PAYMENT protocols where the buyer in ORDER is the receiver in SHIPPING and the payer in PAYMENT.

The potential benefits of our protocol-based approach over traditional approaches include the following. One, for process *design*, protocols are naturally reusable whereas complete processes are not. More importantly, protocols lend themselves to modeling abstractions such as refinement and aggregation. Two, for process *implementation*, implementations of agents playing multiple roles can be more readily assembled from specifications of the roles. Three, for process *enactment*, flexible protocols enable each agent to exercise discretion via its policies or preferences even as it follows a protocol. For example, a merchant may accept only cash for discounted goods and a customer may prefer to pay for goods early or late depending upon private considerations such as of fiscal year accounting. This flexibility also enables us to capture and handle business exceptions and opportunities in a natural manner at the level of protocols. Four, for process *monitoring*, protocols provide a clean basis for determining that the interacting agents are complying with their roles in the given protocols.

1.2 Contributions

We motivate and characterize the key properties that would enable engineering multiagent systems with a special emphasis on applications such as cross-organizational processes. Compared to traditional formal approaches, the emphases on communications and commitments give us a novel start. By assigning meaning to communications in terms of commitments, we accomplish the following. One, we reconstruct the correctness of behaviors by characterizing *compliance* as the eventual discharge of commitments. Two, we characterize the *interoperability* of agents as the alignment of their commitments, meaning that a creditor's expectations about a commitment are met by the debtor. Three, we expand the treatment of design artifacts such as protocols by

viewing them as communication types and showing how to refine and aggregate them. Using the above, we characterize the *conformance* of an agent with a role in a protocol. Further, we characterize important properties of a protocol such as its *transparency* in terms of the ability of the parties involved to verify each other's compliance. By contrast, traditional approaches (formal or otherwise) are largely confined to details such as message ordering and occurrence, and thus miss the forest for the trees.

Importantly, unlike most other multiagent systems work, our approach is undergirded by key ideas of distributed computing, especially dealing with the fact that key information is not immediately shared by all parties (even if they wish to share it). In fact, this is why protocols are important beyond plain commitments. This paper characterizes the above concepts under realistic assumptions, including multiparty settings with asynchronous communication (which aren't accommodated even in fairly recent interoperability research, e.g., [5,6,7]). Hence, this paper reflects crucial basic research not being addressed elsewhere. Its relevance to declarative agent languages and techniques arises from the fact that declarative representations for interaction are central to engineering robust, flexible multiagent systems, and this paper introduces and illustrates correctness criteria based on such declarative representations.

We do not introduce a formal framework in which to characterize the properties; nonetheless, we discuss the properties with rigor appropriate to illuminate their essential nature. This is consistent with our aim of motivating the properties and pointing out the challenges in their verification.

The rest of this paper is organized as follows. Section 2 introduces commitments and protocols in greater detail. Section 3 characterizes the correctness properties for interactions. Section 4 describes our contributions in relation to the most relevant literature. Section 5 lays out an ambitious agenda for multiagent systems research.

2 Background on Protocols and Commitments

In classical software engineering methodologies, information modeling involves the application of key abstractions such as classification, aggregation, and association among components. It would be valuable to develop similar abstractions for interactions. Notice that traditional flow-based process models don't readily support such abstractions. One, existing work imposes severely limiting assumptions to support such abstractions—refinement is specified for Petri nets restricted to one input and one output place [8], which are not as expressive as general Petri nets needed to express real processes. Two, absent a business-level semantics, the models are rigid and any deviation would be potentially erroneous, thus making it difficult to refine or generalize processes.

By contrast, protocols focus on interactions, not on implementations. Our commitment-based semantics of protocols enables us to determine if a protocol refines another protocol, and how protocols may be aggregated into other protocols. Further, we specify a protocol primarily in terms of the vocabulary for communication that it defines and only secondarily in terms of (generally, ad hoc) constraints on the ordering and occurrence of messages. By basing correctness on the discharge of commitments, we enable agents to behave flexibly. For example, a merchant may ship before receiving payment if it wishes; a customer may pay directly or via a third party; and so on. On

occasion, an application may impose an otherwise ad hoc constraint. For example, in a (sit-down) restaurant, the protocol is to pay after food has been received and consumed; in a drive-through, payment precedes delivery. Such constraints often are merely guidelines for the participants and have no bearing on correctness unless they are enshrined in commitments. For example, a restaurant patron may pay early; a drive-through clerk may hand over the food before taking payment from the customer.

Flexible enactment and modeling in terms of refinement and aggregation are possible only because our semantics establishes the correctness criteria by which legitimate enactments, refinements, and aggregations can be identified [4]. Commitments express how contractual relationships form and progress during the agents' interactions. The commitment-based semantics is readily grounded via operational or messaging-level constraints [9].

Commitments. Contracts are key to flexible interoperation. Hohfeld [10] clarified a legal notion of contracts. Commitments cover the relevant aspects of Hohfeld's notions [11], and thus naturally represent the contractual relationships of interest.

Two main forms of commitments arise [12]: *practical* commitments are about bringing about a future condition (i.e., oriented toward tasks), whereas *dialectical* commitments [13] are about staking a claim (as in argumentation) about the past, present, or future (i.e., oriented toward assertions). The distinction between them is significant even when directed to the future. For example, I might commit dialectically that the postman will ring twice, without committing practically to ensure that the postman rings twice. This paper deals with practical commitments. For example, the customer's agreement to pay the price for the book after it is delivered is a practical commitment that the customer (as debtor) has towards the bookstore (as creditor) to ensure the price is paid.

Using commitments enables us to model interactions *computation independently* (using this term as in Model-Driven Architecture (MDA) [14]). On the one hand, commitments describe the evolving state of the ongoing business interaction and how it evolves due to the participants' communications. On the other hand, commitments help express the expectations that participants have of one another: this is the fundamental purpose of a protocol. Jointly, these enable us to readily detect and accommodate business exceptions and opportunities. Consequently, commitments lend coherence to interactions [15].

Commitments can be manipulated through a small set of operations, including create, discharge, cancel, release, assign, and delegate [11], which we lack the space to discuss here. With additional assumptions, commitments can be enforced—by penalizing agents who do not comply with their commitments.

Protocols and commitments. An advantage of incorporating commitments in our models is that they directly represent contractual relationships, are flexible, and lend coherence to the interactions of the participants in a process. The formalization of the specialization and generalization hierarchy of protocols is made the more interesting and useful because of the presence of commitments and roles in our model. Instead of considering uninterpreted runs (of actions and states), we consider how the commitments of the various roles evolve over different runs. The use of commitments enables more sophisticated reasoning about meaning than in traditional approaches. In

Table 1. A purchase protocol (customer is c and merchant is m)

$Offer(m, c, payment, book)$ means $Create(m, c, payment, book)$
$Accept(c, m, payment, book)$ means $Create(c, m, book, payment)$
$Reject(c, m, payment, book)$ means $Release(m, c, payment, book)$
$Deliver(m, c, book)$ means $Inform(m, c, book)$
$Pay(c, m, payment)$ means $Inform(c, m, payment)$

particular, it enables us to characterize the similarity of states and protocol refinement in potentially subtle ways. An example is when a participant from its local perspective considers two states as interchangeable simply because it features as the creditor and debtor in the same commitments regardless of the other parties. For instance, in some settings, Alice may care only of her total accounts receivable, and not care if it is Bob or Charlie who is committed to paying her the money. In other words, instead of merely considering raw computations, it makes sense to "normalize" them in terms of commitments so as to make more precise judgments about how protocols relate to one another.

Table 1 shows the messages in a purchase protocol and their meanings. Offer from m to c creates $\mathsf{C}(m, c, payment, book)$; Accept by c creates the countercommitment $\mathsf{C}(c, m, book, payment)$; c's Reject releases m from his commitment. Deliver means that m is informing c that the book has been delivered; essentially, it causes the proposition $book$ to hold. Pay means that c is informing m that the payment has been made; essentially, it causes the proposition $payment$ to hold. The meanings of the messages are crucial, because they help characterize the protocol declaratively. The meanings are systematically formalized in a declarative action language. Our language and technique are introduced in [16, 17, 18].

Figure 1 shows some possible enactments of the purchase protocol between a customer Alice and a merchant EBook concerning the book BNW (for *Brave New World*) and a payment of \$12. In the figure, c_A is $\mathsf{C}(Alice, EBook, BNW, \$12)$; c_B is $\mathsf{C}(EBook, Alice, \$12, BNW)$; c_{UA} and c_{UB} are the unconditional commitments $\mathsf{C}(Alice, EBook, \top, \$12)$ and $\mathsf{C}(EBook, Alice, \top, BNW)$, respectively.

Traditional approaches force a tradeoff: checking compliance is simple with rigid automaton-based representations and difficult with flexible unconstrained reasoning agents. Commitments help us find the happy middle: protocols maximize flexibility

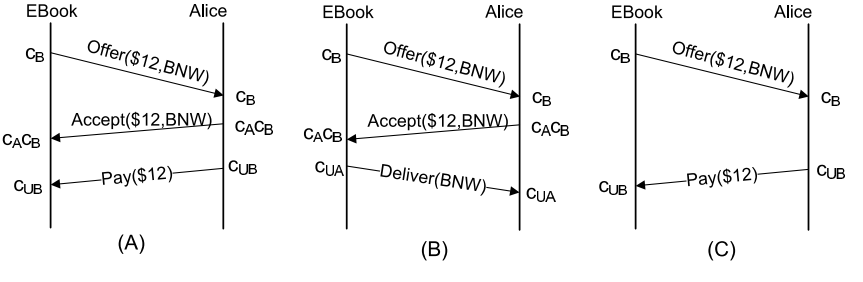

Fig. 1. Three *possible* enactments of protocol in Table 1

by constraining the participants' interactions at the business level, yet provide a crisp notion of compliance: a party complies if its commitments are discharged, no matter if delegated or otherwise manipulated.

Protocols and computations. In essence, each protocol allows a set of computations or runs, each run being an alternative that requires a specific sequence of actions upon the participants. Two basic intuitions about protocol refinement are that (1) a more general protocol includes additional runs (more ways to satisfy) beyond those in a less general protocol; and (2) a more general protocol includes shorter runs (fewer steps to satisfy) than a less general protocol.

Our commitment-based semantics yields a rigorous basis for protocol *refinement* and *aggregation* [19]. In principle, these properties enable reusing protocols from a repository. For example, PAYMENT BY CHECK refines PAYMENT. Further, ORDER, PAYMENT, and SHIPPING can be combined into a new protocol for PURCHASE. This composed protocol would capture the reusable interactions and service agreements that underlie a business process. For example, PURCHASE would specify how orders may be placed, payments made, and shipping arranged. When protocols are composed, so are the roles; e.g., the payer in PAYMENT may be composed with the receiver in SHIPPING. Multiple copies of the same protocol may be composed: in an ARBITRAGE protocol, the arbitrageur role would compose the seller role in one copy of PAYMENT with the buyer role in the second copy.

As in other formal semantics, the runs are merely abstract entities used to establish logical properties. We would never explicitly enumerate the potentially infinite number of possible runs, but we can use the abstract definition to show important algebraic relationships. Mallya & Singh [19] show important progress, but their approach is far from complete. Specifically, it deals with sets of runs, but does not apply directly on protocol specifications as one would find in a repository.

3 Correctness Properties

We begin by motivating some key definitions. Notice that although the above discussion uses protocols as design artifacts, compliance and interoperability apply without regard to any protocol. Although our main definitions and methods are oriented toward commitments, they are undergirded by considerations of distributed computing, especially of asynchrony in messaging.

3.1 Interoperability

The *interoperability* of a set of roles or agents, regardless of protocol, means that they jointly meet the expectations they place on each other. Some aspects of interoperability depend on meanings; others on the messaging system that underlies communications.

We assume that messaging is asynchronous, reliable, and pairwise (for each sender and receiver) order-preserving: this matches what emerging middleware standards [20] offer. Thus in two-party cases, each party would eventually learn of the relevant moves and expectations of the other: the only kind of pathology possible is that the parties

may view some pairs of messages in opposite orders. In multiparty cases, the messaging conditions can become more subtle: e.g., a party would lack direct information about messages exchanged among other parties. Mostly, this is a good thing because the parties can proceed with minimal mutual dependencies. However, when such information materially affects a desired property, we would need to change either the requirements (so information about remote events becomes irrelevant) or the specification (so that the necessary information flows to the appropriate parties).

Interoperation classically is treated as a conjunction of liveness and safety. To these we add alignment.

Liveness means that progress will take place: desirable states will be visited infinitely often. Liveness can fail if a receiver blocks (awaiting a message that is never sent). For example, let Buyer-A demand delivery before payment and Seller-A demand payment before delivery. Now, Buyer-A and Seller-A would deadlock, each awaiting the other's message.

Safety means that the system doesn't enter an undesirable state: agents must be ready to receive the messages being sent to them. Safety is best understood in a multiparty setting. If a buyer expects to receive a confirmation before a shipment but receives them in the opposite order, its resultant state is not defined. We should ensure the messages occur in only those orders that the buyer accepts.

We apply causality [21] to model the above concepts. The sending of a message is causally prior to its receipt; for any two locally ordered events (sends or receives), the first is (potentially) causally prior to the second: "potential" because from external observations we cannot infer if the two events are truly related. We can infer true causality from the agents' specifications, in settings where the specifications are available. We can characterize liveness and safety in terms of the compatibility among causal orders involving receives and sends. We conjecture that the above will yield superior solutions to those in the recent distributed computing literature, e.g., [5,6,7]. The literature considers two-party cases or violates substitutability: that substituting an agent with a conforming agent must preserve interoperability.

Alignment is interoperability with respect to expectations at the level of meaning: do the participants agree about the states of their commitments to each other? A set of agents or roles is *aligned* provided throughout any enactment, whenever one concludes it is the creditor of a commitment, the corresponding debtor x concludes that x is the debtor of the commitment [22]. In other words, the debtor recognizes a commitment that the creditor expects of it. How commitments are created, discharged, and manipulated depends on the messages sent and received.

From the point of view of interoperability, interesting agent specifications are of two kinds: constitutive and regulative [22]. An agent's constitutive specification deals only with the meaning of messages. In other words, it specifies what messages *count as* for the agent. An agent's regulative specification, in contrast, describes agent behavior; i.e., it describes the conditions under which the agent sends and receives particular messages. Regulative specifications are thus closer to implementations.

Agents could be misaligned if, in their constitutive specifications, messages are interpreted differently. For example, if the buyer and seller interpret the Offer message as different commitments, they would be misaligned [22] even though they satisfy safety.

Judging the constitutive alignment of a set of agents by statically analyzing their specifications is nontrivial because message meanings may be conditional, and thus potentially affected by how other messages change the relevant conditions. For example, if one message constitutes an authorization and the meaning of a second message relies upon that authorization, the commitments resulting from the second message would depend upon whether the first message precedes it.

Agents could also become misaligned due to asynchrony: the debtor's and the creditor's conclusions about a commitment may conflict because they see different messages occurrences or orders. Delegations and assignments of commitments inherently involve three parties, and are thus even more challenging.

A specification may fail safety or liveness without failing alignment. We saw above that Buyer-A and Seller-A fail liveness. However, they may never disagree about their commitments and hence would satisfy alignment.

3.2 Conformance and Operability

Conformance and operability apply to each interoperability property: liveness, safety, and alignment. A role *conforms* to, i.e., is a subtype of, another role provided the first role meets all expectations placed on the second and holds no expectations of others beyond what the second does. Similarly, an agent conforms to, i.e., instantiates, a role. Conformance is important because it helps us build a library of roles without which engineering would lapse into one-off solutions. To handle conformance properly would require considering the semantics of protocols not in terms of simple runs, but in terms of the choices they afford each role. Echoing the intuition of alternating refinement [23], expectations placed on a role correspond to "external" choices; expectations held by a role correspond to "internal" choices.

A protocol is *operable*, i.e., potentially enactable, if the roles it specifies are interoperable. A protocol may fail to be operable when it requires a role to act based on events that the role cannot observe. Operability is an important quality criterion for protocols: ideally, the protocols in a library should be operable, so developers may implement selected roles conformantly, and be assured of interoperation.

Let protocol FLEXIBLE PURCHASE allow a payment to occur before or after the delivery of goods. It is easy to see that Buyer-A and Seller-A (introduced above), respectively, conform to the customer and merchant roles in FLEXIBLE PURCHASE. Recall, however, that Buyer-A and Seller-A together fail liveness. Hence FLEXIBLE PURCHASE is not operable for liveness. Conversely, let PREPAID PURCHASE require payment to occur before delivery. Then, any pair of conforming customer and merchant would be live and safe. Hence, PREPAID PURCHASE is operable. Buyer-A is nonconformant with the customer role, whereas Seller-A is conformant with the merchant role of PREPAID PURCHASE. Seller-A and Buyer-A failing liveness doesn't mean PREPAID PURCHASE is inoperable: it is Buyer-A that is messed up.

3.3 Compliance and Transparency

Compliance means that each agent performs as expected by others, by discharging its commitments. We can prove compliance only when we know each agent's specification

and relevant assumptions about the environment hold. That is, compliance can be verified for specific runs but not proved in general for open systems [24]. Notice that alignment and compliance are independent of each other: e.g., an interoperable buyer may be committed to pay, but may refuse to do so. An agent may *verify* a debtor's compliance based on its observations in a specific enactment. Assuming that the discharge of a commitment is observable (e.g., occurs via a message), verifying compliance is simple in two-party cases. If a debtor complies, the creditor would eventually know. If a debtor does not comply, then the creditor would eventually know—provided the commitment includes a deadline. In multiparty cases, a creditor may lack some important observations, and hence special techniques would be required to verify alignment.

A protocol is *transparent* if each role in it can verify the compliance of its debtors. However, not all protocols enable each role to verify compliance at runtime: a protocol may be such that "news" relevant to a commitment might not be propagated to the creditor. Transparency is an important quality criterion for protocols: it ensures that participants can verify if others are not complying.

3.4 Refinement and Compatibility

The *refinement* of a protocol by another protocol means that the second protocol generates only computations that are allowed by the first. Modeling via commitments enables us to finesse the intuitions about protocol refinement. For example, a simple PAYMENT protocol might require that the payer transfer funds to the payee. A particular refinement of this might be PAYMENT WITH A CHECK. To pay with a check, the payer would send a check to the payee, who would deposit the check at his bank, which would present it to the payer's bank, which would send the funds to the payee's bank, which would make those funds available to the payee. Thus PAYMENT BY CHECK is a specialization of PAYMENT, but it involves additional roles and steps, and skips some of the steps of PAYMENT, e.g., direct transfer. With a commitment-based definition, we can formally establish that PAYMENT BY CHECK refines PAYMENT—something that would not be possible with traditional approaches because of the above differences between the two protocols. The key intuition is that the commitments at critical states line up correctly. This is a significant departure from traditional notions of refinement which, because they lack commitments, insist upon the computations to match in their detailed steps.

Notice that an agent designed to play a role in a refined protocol may not comply with any role in the original protocol. This is because the agent may not interpret messages in a way compatible with the original protocol. For example, in PAYMENT BY CHECK, a merchant may interpret a check as being adequate as a receipt (once it is cleared and returned to the customer by the customer's bank), but the customer may not interpret it like that and may continue to expect a separate receipt as in PAYMENT. Further, the agent may fail to interoperate with roles defined in the original protocol. This is because it may send messages that are not defined in the original protocol. In general we would not be able to substitute a role from a refined protocol for a role in the original protocol. The foregoing is motivation for the property of *compatibility*, which determines if roles in one protocol conform to roles in another protocol.

Table 2 summarizes the above properties. With the exception of compliance, these properties can be verified by a static analysis of the appropriate specifications.

Table 2. The properties summarized

Property	Of What?
Refinement, compatibility, operability, transparency	Protocols
Interoperability (safety, liveness, or alignment)	Agents and roles
Conformance	Roles
Compliance	Agents

4 Discussion: Relevant Literature

Our main contribution in this paper is in characterizing the key correctness properties that would support an interaction-oriented approach to building software systems, particularly cross-organizational business processes. In particular, the correctness properties reflect high-level requirements of such systems.

Interestingly, Parnas [25] proposed early in the study of software architectures that connectors be treated not as control or data flow constructs but as *assumptions* made by each component about the others. Arguably, much of the subsequent work on software architecture regressed from Parnas' insight: it has primarily considered connectors at the level of flow, e.g., dealing exclusively with message order and occurrence [26]. In formulating the assumptions at a high level, we see a great opportunity for multiagent systems research to address some of the long-standing challenges in software.

Conventional formal methods. Current modeling formalisms, such as finite state machines and Petri Nets, originated in distributed computing and apply at lower levels of abstraction than needed for flexible business interactions [27,8]. When applied to business protocols, these formalisms result in specifications that are over-constrained to the level of specific sequences of actions. Recent approaches have sought to express scheduling requirements declaratively, via temporal logic [28,29,30]. Although they are more flexible and modular than operational representations, these approaches do not express business semantics.

FIPA, the Foundation for Intelligent and Physical Agents (now part of IEEE) recognized the importance of reusable interaction protocols in the late 1990s [31]. Odell *et al.* [32] give one of the earliest uses of UML for protocols. They show how various UML diagrams can be applied for modeling agent interactions. This work shows about how far you can go in a conventional software framework, and has inspired our work. The present paper is about fundamental enhancements to conventional models to capture protocols and their commitment-based semantics.

Leading approaches model conversations via finite-state machines and establish properties such as how roles may realize a protocol or a protocol subsumes another [33,34]. Dastani *et al.* [35] show how to model a rich family of coordination connectors for multiagent systems. Honda *et al.* [36] develop a type theory that would support multiparty sessions: in essence this would help robustly generate roles. These works formalize protocols as data and control flow abstractions. They do not consider the meaning of messages and thus lack the business-level semantics that distinguishes our

work. However, their treatment of messages and computations at a low level is useful, and complementary to our work.

Whereas deontic logic only deals with what is obligatory or permissible and thus disregards an agent's obligations *to* another agent, commitments are directed and context sensitive. Commitments include support for a variety of operations [11,37]. Foster *et al.* [38] seek to capture the semantics of process interactions via the notion of obligation policies. Obligations are rather weak in their formulation, however. Specifically, obligations are not reified, and cannot be manipulated to capture flexible interactions among independent parties. Lomuscio *et al.* [39] formalize correctness properties in a temporal logic and show how to verify them. They consider obligations but do not consider commitments as here. Lomuscio *et al.* also concentrate on only one correctness property, which is somewhat like compliance.

Business processes. The MIT Process Handbook (MITPH) [40] is of great relevance intellectually. MITPH includes an extensive classification and systematic organization of business processes based on two dimensions of process hierarchies, one that composes the *uses* of a process out of its constituent *parts*, and another that subclasses *generalizations* of a process into *specializations*. Our work can provide the rigorous underpinnings for work such as the MITPH. Grosof and Poon [41] develop a system to represent and execute business rules from MITPH. Wyner and Lee [42] study specialization for data flow diagrams. Their approach can form the basis of the processes identified in MITPH. These concepts turn out to be complex and not readily applied to entire business processes. Further, since Wyner and Lee do not capture the content through a high-level representation such as commitments, the results are not intuitive.

Our approach agrees with the newer declarative forms of artifacts-based process modeling [43] in terms of deemphasizing low-level operational details in favor of business semantics. However, these approaches do not have a central organizing principle on par with commitments, and thus do not offer a generic and flexible basis for determining the properties we introduced above.

Agent communications. Fornara and Colombetti [44] describe how commitments relate to FIPA messages, demonstrating this with an example. Rovatsos [45] proposes a commitment-based semantics for communications under synchronous messaging. His approach violates autonomy by legislating agent behaviors from within the language specification: this level of prescription is ill-suited to most multiagent applications.

Yolum and Singh [46] [47] offer one of the first accounts of the use of commitments in modeling protocols to improve flexibility for participating agents, which was enhanced by Winikoff *et al.* [48]. Johnson *et al.* [49] develop a scheme for identifying when two commitment-based protocols are equivalent. Their scheme, however, is simplistic, classifying protocols based solely on their syntactic structure. Our work provides stronger results from an application point of view and relates better to Web services.

Commitments have found application in formalizing argumentation, e.g., [50,51]. Usually, though, this work makes simplifying assumptions such as (1) maintaining a unique commitment store; (2) informally specifying the meanings of communicative acts as effects on the store; (3) assuming synchronous two-party communications.

Agent-oriented software engineering (AOSE). A number of useful software methodologies for building multiagent systems for IT applications have emerged that incorporate rich metamodels and describe how to build a series of software artifacts [52,53,3]. Garcia-Ojeda *et al.* [54] synthesize existing metamodels into a comprehensive metamodel of organizations geared toward process modeling. We recently developed Amoeba, a protocol-based methodology compatible with the ideas of this paper [1].

The above methodologies address the challenges of autonomy and heterogeneity by giving prominence to communication. Such works are clearly valuable and worthwhile. However, current approaches do not consider the full subtleties both of meaning and of distribution. By contrast, this paper addresses the foundations for business interactions understood in terms of commitments. The proposed definitions will offer a foundations for building a new family of tools that, in principle, could be used within any of the above methodologies, because they all support aspects of interaction and of agents playing roles in interactions.

5 Conclusions and Directions

This paper presents a key step in our program of research to develop underpinnings of multiagent systems—and indeed, of all software—on interactive grounds with an emphasis on declarative formulations. The main point to take away is the richness of the correctness properties. These properties echo well-known conventional properties but their characterization in a declarative, interactive setting adds a lot of subtlety that traditional approaches cannot express. The foregoing leads to two broad questions.

- *Theory*. What are practical decision algorithms for these properties? How can we specify agents who may play specified roles (while applying their local policies)? How can we determine that agents (supposedly) enacting a protocol are complying with the protocol? What are practical algorithms for judging the varieties of interoperability, conformance, operability, compliance, and transparency?
- *Suitability and applicability*. Does representing meaning via commitments provide a sufficiently natural basis for business interoperation? How readily can meaning be associated with tools to engineer and use protocols? Can we specify commitments sufficiently precisely in real-life business settings? How can we use the above properties and algorithms to enable protocol design and agent implementation?

The above questions constitute a substantial research agenda. Addressing this agenda presupposes an adequate formalization of commitments. Recent work on the formal semantics of commitments [12] and commitment operations [9] are steps in that direction.

References

1. Desai, N., Chopra, A.K., Singh, M.P.: Amoeba: A methodology for modeling and evolution of cross-organizational business processes. ACM Transactions on Software Engineering and Methodology (TOSEM) 19(2) (to appear, April 2010)

2. Desai, N., Chopra, A.K., Arrott, M., Specht, B., Singh, M.P.: Engineering foreign exchange processes via commitment protocols. In: Proceedings of the 4th IEEE International Conference on Services Computing (SCC), pp. 514–521. IEEE Computer Society Press, Los Alamitos (2007)
3. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Journal of Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (2004)
4. Desai, N., Mallya, A.U., Chopra, A.K., Singh, M.P.: Interaction protocols as design abstractions for business processes. IEEE Transactions on Software Engineering 31(12), 1015–1027 (2005)
5. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: A priori conformance verification for guaranteeing interoperability in open environments. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 339–351. Springer, Heidelberg (2006)
6. Fournet, C., Hoare, C.A.R., Rajamani, S.K., Rehof, J.: Stuck-free conformance. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 242–254. Springer, Heidelberg (2004)
7. Giordano, L., Martelli, A.: Verifying agent conformance with protocols specified in a temporal action logic. In: Basili, R., Pazienza, M.T. (eds.) AI*IA 2007. LNCS (LNAI), vol. 4733, pp. 145–156. Springer, Heidelberg (2007)
8. van der Aalst, W., van Hee, K.: Workflow Management Models, Methods, and Systems. MIT Press, Cambridge (2002)
9. Chopra, A.K., Singh, M.P.: Multiagent commitment alignment. In: Proceedings of the 8th International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), Columbia, SC, IFAAMAS, May 2009, pp. 937–944 (2009)
10. Hohfeld, W.N.: Fundamental Legal Conceptions as Applied in Judicial Reasoning and other Legal Essays. Yale University Press, New Haven (1919); A 1919 printing of articles from 1913
11. Singh, M.P.: An ontology for commitments in multiagent systems: Toward a unification of normative concepts. Artificial Intelligence and Law 7, 97–113 (1999)
12. Singh, M.P.: Semantical considerations on dialectical and practical commitments. In: Proceedings of the 23rd Conference on Artificial Intelligence (AAAI), July 2008, pp. 176–181. AAAI Press, Menlo Park (2008)
13. Walton, D.N., Krabbe, E.C.W.: Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning. State University of New York Press, Albany (1995)
14. OMG: The Object Management Group's Model Driven Architecture, MDA (2006), http://www.omg.org/mda/
15. Jain, A.K., Aparicio IV, M., Singh, M.P.: Agents for process coherence in virtual enterprises. Communications of the ACM 42(3), 62–69 (1999)
16. Chopra, A.K., Singh, M.P.: Contextualizing commitment protocols. In: Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1345–1352 (2006)
17. Desai, N., Singh, M.P.: A modular action description language for protocol composition. In: Proceedings of the 22nd Conference on Artificial Intelligence (AAAI), July 2007, pp. 962–967. AAAI Press, Menlo Park (2007)
18. Desai, N., Singh, M.P.: On the enactability of business protocols. In: Proceedings of the 23rd Conference on Artificial Intelligence (AAAI), July 2008, pp. 1126–1131. AAAI Press, Menlo Park (2008)
19. Mallya, A.U., Singh, M.P.: An algebra for commitment protocols. Journal of Autonomous Agents and Multi-Agent Systems 14(2), 143–163 (2007)
20. AMQP: Advanced message queuing protocol (2007), http://www.amqp.org
21. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21(7), 558–565 (1978)

22. Chopra, A.K., Singh, M.P.: Constitutive interoperability. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), May 2008, pp. 797–804 (2008)
23. Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.Y.: Alternating refinement relations. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 163–178. Springer, Heidelberg (1998)
24. Venkatraman, M., Singh, M.P.: Verifying compliance with commitment protocols: Enabling open Web-based multiagent systems. Autonomous Agents and Multi-Agent Systems 2(3), 217–236 (1999)
25. Parnas, D.L.: Information distribution aspects of design methodology. In: Proceedings of the International Federation for Information Processing Congress, vol. TA-3, pp. 26–30. North Holland, Amsterdam (1971)
26. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Signature Series. Addison-Wesley, Boston (2004)
27. Harel, D., Gery, E.: Executable object modeling with statecharts. IEEE Computer 30(7), 31–42 (1997)
28. Singh, M.P.: Distributed enactment of multiagent workflows: Temporal logic for service composition. In: Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), July 2003, pp. 907–914. ACM Press, New York (2003)
29. Pesic, M., Schonenberg, M.H., Sidorova, N., van der Aalst, W.M.P.: Constraint-based workflow models: Change made easy. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 77–94. Springer, Heidelberg (2007)
30. Wu, Q., Pu, C., Sahai, A., Barga, R.S.: Categorization and optimization of synchronization dependencies in business processes. In: Proceedings of the 23nd International Conference on Data Engineering (ICDE), pp. 306–315. IEEE, Los Alamitos (2007)
31. FIPA: FIPA interaction protocol specifications, FIPA: The Foundation for Intelligent Physical Agents (2003), http://www.fipa.org/repository/ips.html
32. Odell, J., Parunak, H.V.D., Bauer, B.: Representing agent interaction protocols in UML. In: Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering, AOSE (2001)
33. Benatallah, B., Casati, F., Toumani, F.: Representing, analysing and managing web service protocols. Data & Knowledge Engineering 58(3), 327–357 (2006)
34. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation specification: A new approach to design and analysis of e-service composition. In: Proceedings of the Twelfth International World Wide Web Conference (WWW), pp. 403–410 (2003)
35. Dastani, M., Arbab, F., de Boer, F.S.: Coordination and composition in multi-agent systems. In: Proceedings of the 4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 439–446. ACM, New York (2005)
36. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL). ACM, New York (2008)
37. Singh, M.P., Chopra, A.K., Desai, N.: Commitment-based SOA. IEEE Computer 42 (accepted, 2009), http://www.csc.ncsu.edu/faculty/mpsingh/papers/
38. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Model-based analysis of obligations in web service choreography. In: Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW), pp. 149–156 (2006)

39. Lomuscio, A., Qu, H., Solanki, M.: Towards verifying compliance in agent-based web service compositions. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), Columbia, SC, International Foundation for Autonomous Agents and MultiAgent Systems, pp. 265–272 (2008)
40. Malone, T.W., Crowston, K., Herman, G.A. (eds.): Organizing Business Knowledge: The MIT Process Handbook. MIT Press, Cambridge (2003)
41. Grosof, B.N., Poon, T.C.: SweetDeal: Representing agent contracts with exceptions using XML rules, ontologies, and process descriptions. In: Proceedings of the 12th International Conference on the World Wide Web, pp. 340–349 (2003)
42. Wyner, G.M., Lee, J.: Defining specialization for process models. In: [40], pp. 131–174. MIT Press, Cambridge (2003)
43. Hull, R.: Artifact-centric business process models: Brief survey of research results and challenge. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)
44. Fornara, N., Colombetti, M.: Defining interaction protocols using a commitment-based agent communication language. In: Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), July 2003, pp. 520–527. ACM Press, New York (2003)
45. Rovatsos, M.: Dynamic semantics for agent communication languages. In: Proceedings of the 6th international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), pp. 1–8 (2007)
46. Yolum, P., Singh, M.P.: Commitment machines. In: Meyer, J.-J.C., Tambe, M. (eds.) ATAL 2001. LNCS (LNAI), vol. 2333, pp. 235–247. Springer, Heidelberg (2002)
47. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: Applying event calculus planning using commitments. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), July 2002, pp. 527–534. ACM Press, New York (2002)
48. Winikoff, M., Liu, W., Harland, J.: Enhancing commitment machines. In: Leite, J., Omicini, A., Torroni, P., Yolum, p. (eds.) DALT 2004. LNCS (LNAI), vol. 3476, pp. 198–220. Springer, Heidelberg (2005)
49. Johnson, M.W., McBurney, P., Parsons, S.: When are two protocols the same? In: Huget, M.-P. (ed.) Communication in Multiagent Systems. LNCS (LNAI), vol. 2650, pp. 253–268. Springer, Heidelberg (2003)
50. Amgoud, L., Maudet, N., Parsons, S.: An argumentation-based semantics for agent communication languages. In: Proceedings of the 15th European Conference on Artificial Intelligence (ECAI), pp. 38–42. IOS Press, Amsterdam (2002)
51. Norman, T.J., Carbogim, D.V., Krabbe, E.C.W., Walton, D.N.: Argument and multi-agent systems. In: Reed, C., Norman, T.J. (eds.) Argumentation Machines. Kluwer, Dordrecht (2004)
52. Bergenti, F., Gleizes, M.P., Zambonelli, F. (eds.): Methodologies and Software Engineering for Agent Systems. Kluwer, Boston (2004)
53. Henderson-Sellers, B., Giorgini, P. (eds.): Agent-Oriented Methodologies. Idea Group, Hershey (2005)
54. Garcia-Ojeda, J.C., DeLoach, S.A., Robby, O.W.H., Valenzuela, J.: O-MaSE: A customizable approach to developing multiagent processes. In: Luck, M., Padgham, L. (eds.) Agent-Oriented Software Engineering VIII. LNCS, vol. 4951, pp. 1–15. Springer, Heidelberg (2008)

Reasoning and Planning with Cooperative Actions for Multiagents Using Answer Set Programming

Tran Cao Son[1] and Chiaki Sakama[2]

[1] Dept. of Computer Science, New Mexico State University, Las Cruces, NM 88003, USA
tson@cs.nmsu.edu
[2] Computer and Communication Sciences, Wakayama University, Wakayama 640-8510, Japan
sakama@sys.wakayama-u.ac.jp

Abstract. In this paper, we investigate the multiagent planning problem in the presence of cooperative actions and agents, which have their own goals and are willing to cooperate. To this end, we extend the action language $\mathcal{A}$ in [12] to represent and reason about plans with cooperative actions of an individual agent operating in a multiagent environment. We then use the proposed language to formalize the multiagent planning problem and the notion of a joint plan for multiagents in this setting. We discuss a method for computing joint plans using answer set programming and provide arguments for the soundness and completeness of the implementation.

1 Introduction

Cooperative actions are actions of an agent which can be executed only if the agent is operating in a multiagent environment. They can be actions for soliciting something from other agents or actions for setting up some conditions for other agents. They differ from individual actions in that they might affect other agents. Cooperative actions are important not only in situations where multiple agents have to work together to accomplish a common goal but also in situations where each agent has its own goal. This can be seen in the following story, a modified version of the story in [21]:

Example 1. Three new students A, B, and C are moving in a shared apartment and planning to decorate their rooms. Each would like to hang one of their objects on the wall, e.g., A would like to hang a mirror, B a diploma, and C a painting. A and B know how to use either a nail or a screw to complete their job but C knows to use the screw only. A has neither a nail or a screw. B has both. C has only a nail. To use a nail, one will need a hammer. Among three, only B has a hammer.

Do the students have a joint-plan that allows each of them to achieve his/her goal?

Intuitively, we can see that only B can accomplish her job independent of A and C. The three can achieve their goals if B uses the hammer and the nail to hang her diploma then gives A the hammer and C the screw, respectively. C, on the other hand, gives A the nail and uses the screw to hang her painting. A uses the nail (from C) and the hammer (from B) to hang her mirror. Of course, to avoid unpleasant moments, A should ask for the nail (from C) and the hammer (from B) and C should ask for the screw (from B).

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 208–227, 2010.

However, it is easy to see that if either B or C does not want to give out anything, then only B can achieve her goal. Furthermore, if B decides to use the screw instead of using the nail in hanging her diploma, then C has no way of achieving her goal. ◇

In the above example, the action of giving a nail, a hammer, or a screw between the students can be considered as cooperative actions. The action of requesting something from others can also be considered as cooperative actions. It is obvious that without some cooperative actions, not all students can achieve their own goals. Even with the cooperative actions at their disposal, the students might still need to coordinate in creating their corresponding plans.

In Example 1, agents (the students) maintain their own local worlds and their actions do generally not affect others' worlds. It should be emphasized that the fact that agents have their own world representation does not exclude the situations in which the worlds of different agents overlap and the execution of one agent's individual actions might affect others as well or the execution of their joint-action.

Example 2. Let us consider A and B who are in one room and studying at their tables. Each of them sits next to a switch which can control the lamp in the room. Flipping either switch will change the status of the light.

Assume that A and B maintain their world representation separately. (They might use the same theory for this purpose but we will not impose this.) Obviously, if A flips the switch next to her, the world in which B is in will also change.

Similarly, if A and B lift a table and place it at different location, their joint-action change the world of both as well. ◇

In this paper, we will consider multiagent planning problems in which each agent maintains its own representation about the world and its capabilities, which includes individual actions and cooperative actions; and has its own goal. We are mainly interested in the process of creating a joint plan prior to its execution. We will begin by extending the language $\mathcal{A}$ in [12] to allow cooperative actions for a single agent. The semantics of the new language is defined by a transition function which maps pairs of actions and states to states. We then define the multiagent planning problems and the notion of a joint plan for multiagents in presence of cooperative actions. Finally, we discuss a method for computing joint plans using answer set programming [18, 19].

2 An Action Language with Cooperative Actions

In this section, we present a language for representing and reasoning about plans for an agent in the multiagent environment with cooperative actions. To this end, we extend the language $\mathcal{A}$ in [12] to allow cooperative actions[1]. In this paper, we consider cooperative actions as actions that an agent would not have if she were in a single agent environment. Specifically, we consider two types of cooperative actions, one that requests the establishment of a condition in an agent's world and another establishes some conditions in the world of another agent. We will assume an arbitrary but fixed set of agent identifiers $\mathcal{AG}$. A planning problem of an agent in $\mathcal{AG}$ is defined over a set

[1] The choice of $\mathcal{A}$ will be discussed in Section 5.

of fluents (or state variables) F, a set of *individual actions* A, and a set of *cooperative actions* C. We will assume that A always contains a special action wait which does not have any effect on the agent's world[2]. Furthermore, we will require that actions in C do not appear in A. This highlights the fact that the cooperative actions are presented due to the presence of other agents.

A *fluent literal* (or *literal*) is either a fluent or its negation. Fluent formulas are propositional formulas constructed from literals and propositional connectives.

2.1 Specifying Individual Actions

A *domain specification* DI over F and A describes the individual actions of an agent and consists of laws of the following form:

$$a \textbf{ causes } l \textbf{ if } \phi \tag{1}$$

$$a \textbf{ executable } \phi \tag{2}$$

where a is an individual action (in A), l is a fluent literal and ϕ is a set of fluent literals.

A law of the form (1), called a *dynamic law*, states that if a is executed when ϕ is true then l becomes true. (2) is an *executability condition* and says that a can be executed only if ϕ is true. The semantics of a domain specification is defined by the notion of *state* and by a *transition function* Φ, that specifies the result of the execution of an action a in a state s.

A set of literals S satisfies a literal l (l holds/is true in S), denoted by $S \models l$, if $l \in S$. For a set of literals ϕ, $S \models \phi$ if $S \models l$ for every $l \in \phi$. A *state* s is a set of fluent literals that is *consistent* and *complete*, i.e., for every $f \in \mathtt{F}$, either $f \in s$ or $\neg f \in s$ but $\{f, \neg f\} \not\subseteq s$. In the following, $\overline{l}$ denotes the negation of l, i.e., if $l = f$ and $f \in \mathtt{F}$, then $\overline{l} = \neg f$; if $l = \neg f$ for some $f \in \mathtt{F}$, then $\overline{l} = f$. For a set of literals S, $\overline{S} = \{\overline{l} \mid l \in S\}$.

An action a is *executable* in a state s if there exists an executability condition (a **executable** ϕ) in DI such that $s \models \phi$.

Let $e_a(s) = \{l \mid \exists (a \textbf{ causes } l \textbf{ if } \phi) \in DI.[s \models \phi]\}$. The result of the execution of a in s is defined by

- $\Phi(a, s) = \mathit{fails}$ if a is not executable in s; and
- $\Phi(a, s) = (s \setminus \overline{e_a(s)}) \cup e_a(s)$ if a is executable in s.

A domain specification DI is *consistent* if $\Phi(a, s) \neq \mathit{fails}$ holds for every pair of action a and state s such that a is executable in s.

Φ is extended to reason about effect of a sequence of actions as follows.

Definition 1 (Transition function). *Let DI be a domain specification, s be a state, and $\alpha = [a_1; \ldots; a_n]$ be a sequence of actions.*

- $\hat{\Phi}(\alpha, s) = s$ *if* $n = 0$;
- $\hat{\Phi}(\alpha, s) = \Phi(a_n, \hat{\Phi}([a_1; \ldots; a_{n-1}], s))$, *otherwise*

where $\Phi(a, \mathit{fails}) = \mathit{fails}$.

[2] We envision a multiagent environment where agents may have to wait for other agents to finish some actions before they can go on with their course of actions.

An agent can use the transition function to reason about effects of its actions and to planning. An action sequence α is a *plan* achieving a set of literals O from a state I iff O is true in $\hat{\Phi}(\alpha, I)$.

Example 3. The domain specification DI_A for A in Example 1 is defined over $\mathtt{F}_A = \{h\_nail, h\_screw, mirror\_on, h\_ham\}$ and $\mathtt{A}_A = \{hw\_nail, hw\_screw\}$ with the set of laws[3]:

hw_nail	**causes**	$mirror\_on$	hw_screw	**causes**	$mirror\_on$
hw_nail	**causes**	$\neg h\_nail$	hw_screw	**causes**	$\neg h\_screw$
hw_nail	**executable**	h_nail, h_ham	hw_screw	**executable**	h_screw

In all of the above, the prefix "hw" stands for "hang with" and "h" stans for "has." □

2.2 Specifying Cooperative Actions

The specification of the set of cooperative actions of an agent, denoted by DC, is defined over $\mathtt{C}$ and $\mathtt{F}$ and consists of laws of the following form:

$$r \textbf{ requests } \gamma \textbf{ from } \mathcal{A}_\mathbf{i} \textbf{ may\_cause } \phi \textbf{ if } \psi \text{ and} \tag{3}$$

$$p \textbf{ provides } \gamma \textbf{ for } \mathcal{A}_\mathbf{i} \textbf{ causes } \phi \textbf{ if } \psi \tag{4}$$

r and p are action names in $\mathtt{C}$, γ, ϕ, and ψ are sets of literals and $\gamma \subseteq \phi$, and $\mathcal{A}_\mathbf{i}$ is a set of agent identifiers in $\mathcal{AG}$. r is called a *request* for γ and p an *offer* for γ. Since these actions are intended to address other agents, we require that the identifier of the agent having r and/or p does not belong to $\mathcal{A}_\mathbf{i}$. Furthermore, for a request-action, we require that $\bar{\phi} \cap \psi \neq \emptyset$ which indicates that an agent will only request for something that he/she does not have.

Intuitively, (3) represents a set of requests that can be made by the agent; if the agent makes the request for γ (which is the action r) directed to an agent in $\mathcal{A}_\mathbf{i}$ then ϕ might become true. The condition $\gamma \subseteq \phi$ guarantees that requested literals (γ) are true if the request is satisfied (ϕ). Furthermore, the action can only be executed if ψ is true. For this reason, we call $r(\gamma, i), i \in \mathcal{A}_\mathbf{i}$, an instance of a request (3). Similarly, (4) represents the set of offers $p(\gamma, i)$, $i \in \mathcal{A}_\mathbf{i}$. This offer addresses a request made to the agent by establishing γ (for the requestor). This action is similar to the individual actions in $\mathtt{A}$ of an agent. The main difference is that they also change the worlds of other agents. It can only be executed if ψ is true and its effects is ϕ.

For simplicity of the presentation, we will assume that each action in $\mathtt{C}$ occurs in at most one law of the form (3) or (4). We use *cooperative action* to refer to either a request- or an offer-action. When $\mathcal{A}_\mathbf{i}$ is the set of all other agents, we often omit the part '**from** $\mathcal{A}_\mathbf{i}$' from (3) and '**for** $\mathcal{A}_\mathbf{i}$' from (4).

Example 4. In Example 1, it is reasonable for A to request and/or offer other agents on the literal h_nail. An action for requesting for (offering of) h_nail for A can be specified by

$give\_me\_nail$ **requests** h_nail **from** $\{B, C\}$ **may_cause** h_nail **if** $\neg h\_nail$
get_this_nail **provides** h_nail **for** $\{B, C\}$ **causes** $\neg h\_nail$ **if** h_nail

[3] To simplify the representation, we often write $l_1, \ldots, l_n$ instead of $\{l_1, \ldots, l_n\}$ in describing the domain.

where $give\_me\_nail$ is a request-action and get_this_nail is an offer-action. If the agent A wants to ask for help, then her set of cooperative actions needs to include the action $give\_me\_nail$. On the other hand, if she wants to help others, then it should include the action get_this_nail. $\diamond$

Definition 2 (Planning problem with cooperative actions). *A* planning problem with cooperative actions[4] *$\mathcal{P}$ is a tuple $\langle DI, I, O, DC\rangle$ where DI is a domain specification, I is a state representing the initial state, O is a set of literals representing the goal, and DC is a set of laws of the form (3) and (4).*

Given a planning problem $\mathcal{P} = \langle DI, I, O, DC\rangle$, we need to specify what is a "plan" achieving O in the presence of the cooperative actions. Intuitively, we could consider these actions as the actions of the agent and use the notion of a plan mentioned in the previous subsection. This is, however, not enough since an agent, when executes a request, might or might not receive an offer satisfying his/her request. For example, a request for a nail from A to C might not result in A having the nail because C has already given the nail to B.

We will therefore extend the transition function Φ of the domain specification DI to consider cooperative actions. We will use Φ_D to denote the transition function of $DI \cup DC$. By assuming that cooperative actions are different from the individual actions (i.e., $\mathtt{A} \cap \mathtt{C} = \emptyset$), it suffices to specify what is the result of the execution of a request/offer-action in a given state.

For simplicity of the presentation, we assume that each individual agent executes only one action at a time. The method presents in this paper can be easily extended to the case where individual agents can execute parallel actions.

Let s be a state. We say that an instance $r(\gamma, i)$ of a request-action specified by the law

$$r \textbf{ requests } \gamma \textbf{ from } \mathcal{A}_\mathrm{i} \textbf{ may\_cause } \phi \textbf{ if } \psi$$

in DC is *executable* in s if ψ is true in s. Executing the action $r(\gamma, i)$ in s does not guarantee that the agent will obtain ϕ in the resulting state. This is because the agent, whom the request was made to, might not have the capability to establish ϕ for the requestor. We say that the execution of $r(\gamma, i)$ in s might or might not succeed. As such, the result of executing $r(\gamma, i)$ in s is either s, representing the case when the request is not satisfied (by the agent whom the request was made to); or $(s \setminus \overline{\phi}) \cup \phi$, representing the case when the request is satisfied.

Remark 1. Observe that under the assumption that an agent will execute a request-action only when it is necessary (i.e., $\bar{\phi} \cap \psi \neq \emptyset$), we have that $s \neq (s \setminus \overline{\phi}) \cup \phi$ for every instance $r(\gamma, i)$. This allows us to recognize when a request is satisfied.

An instance $p(\gamma, i)$ of an offer-action specified by the law

$$p \textbf{ provides } \gamma \textbf{ for } \mathcal{A}_\mathrm{i} \textbf{ causes } \phi \textbf{ if } \psi$$

[4] For simplicity of presentation, we will use planning problem instead of planning problem with cooperative actions whenever no confusion is possible.

in DC is *executable* in s if ψ is true in s. The state resulting from executing $p(\gamma, i)$ in s is given by $(s \setminus \overline{\phi}) \cup \phi$.

Definition 3 (Transition function). *The transition function Φ_D over $DI \cup DC$, a mapping from pairs of actions and states to sets of states, is defined as follows. Let s be a state.*

- *For $a \in \mathtt{A}$, $\Phi_D(a, s) = \{\Phi(a, s)\}$ if $\Phi(a, s) \neq$ fails; otherwise, $\Phi_D(a, s) = \emptyset$.*
- *For an instance of an offer-action $p(\gamma, i)$, $\Phi_D(p(\gamma, i), s) = \{(s \setminus \overline{\phi}) \cup \phi\}$ if p is executable in s; otherwise, $\Phi_D(p, s)=\emptyset$.*
- *For an instance of a request-action $r(\gamma, i)$, $\Phi_D(r(\gamma, i), s) = \{s, (s \setminus \overline{\phi}) \cup \phi\}$ if $r(\gamma, i)$ is executable in s; otherwise, $\Phi_D(r(\gamma, i), s) = \emptyset$.*

Remark 2. The definition of Φ_D assumes that each cooperative action occurs in only one law of the form (3) or (4). The definition can be extended to remove this restriction by (*i*) defining a set $ec_{r(\gamma,i)}(s)$ (resp. $ec_{p(\gamma,i)}(s)$), similar to the definition of the set of effects of an action $e_a(s)$ and (*ii*) changing the definition accordingly.

The transition function is extended to reason about plans as follows.

Definition 4 (Plan with cooperative actions). *Let $\mathcal{P}$ be a planning problem $\langle DI, I, O, DC\rangle$. We define*

- *A sequence $s_0, a_0, s_1, \ldots, a_{n-1}, s_n$, where s_i's are states and a_j's are actions, is a* trajectory *if $s_{i+1} \in \Phi_D(a_i, s_i)$ for $0 \leq i < n$.*
- *A trajectory $s_0, a_0, s_1, \ldots, a_{n-1}, s_n$ is a* possible plan *achieving O (or a solution of $\mathcal{P}$) if $s_0 = I$ and $s_n \models O$.*
- *An occurrence of a request $r(\gamma, i) = a_j$ in a trajectory $s_0, a_0, s_1, \ldots, a_{n-1}, s_n$ is* satisfied *if $s_{j+1} \neq s_j$; otherwise, the request is said to be unsatisfied.*

Notice that the third item in the above definition is sensible due to Remark 1. A trajectory satisfying the goal O of the planning problem is a solution of $\mathcal{P}$ if all satisfied requests assumed in the trajectory indeed materialized, i.e., for each satisfied $r(\gamma, i)$ in the trajectory, the agent i executes the action $p(\gamma, j)$ (j is the identifier of the agent issuing the request). The topic of coordination between agents will be discussed in the next section.

Example 5. Let $\mathcal{P}_A = \langle DI_A, I_A, O_A, DC_A\rangle$ be the planning problem for A with DI_A (Example 3), $I_A = \{\neg h\_nail, \neg h\_screw, \neg h\_ham, \neg mirror\_on\}$ and $O_A = \{mirror\_on\}$, and DC_A is the set of actions $give\_me\_nail$ and get_this_nail whose specifications are given (Example 4) and the two actions

$$give\_me\_ham \textbf{ requests } h\_ham \textbf{ from } \{B, C\} \textbf{ may\_cause } h\_ham \textbf{ if } \neg h\_ham,$$
$$get\_this\_ham \textbf{ provides } h\_ham \textbf{ for } \{B, C\} \textbf{ causes } \neg h\_ham \textbf{ if } h\_ham.$$

We can easily check the following:

- for $n \leq 2$, the problem has no possible plan.
- for $n = 3$, $\mathcal{P}_A$ has a possible plan which is the following trajectory: $s_0^A, give\_me\_nail(h\_nail, C), s_1^A, give\_me\_ham(h\_ham, B), s_2^A, hw\_nail, s_3^A$

where

$s_0^A = \{\neg h\_nail, \neg h\_ham, \neg h\_screw, \neg mirror\_on\}$,
$s_1^A = \{h\_nail, \neg h\_ham, \neg h\_screw, \neg mirror\_on\}$,
$s_2^A = \{h\_nail, h\_ham, \neg h\_screw, \neg mirror\_on\}$,
$s_3^A = \{\neg h\_nail, h\_ham, \neg h\_screw, mirror\_on\}$. ◇

3 Planning for Multiagents

In a multiagent environment, each agent needs to know her capabilities. She also needs to know from whom she can ask for some favors or to whom she could offer helps. Furthermore, it is also common that groups of agents need to know about their joint capabilities. It is also possible that agents might talk the same language. This can be summarized as follows.

- Each agent has its own planning problem, which is described in the previous section.
- The agent might or might not share the same world representation. By default, the world representation of the agent is local. For example, the three agents in Example 1 can use the same set of fluents and actions; and A has $\neg h\_nail$ in her initial state whereas B has h_nail in hers, yet this is not a contradictory statement about the world since the fluents are local. On the other hand, the two agents in Example 2 share certain features (e.g. the light) and therefore the fluents encoding these features should have the same value in their representations.
- An agent might request another agent to establish certain conditions in her own world. For example, A might request B to establish h_nail to be true for her.
- An agent might execute some actions that change the local world of another agent. For example, B can give A the nail, thus establishing h_nail in the world of A.
- There might be actions that a set of agents should not execute in parallel. For example, two cars– one goes north-south and another east-west– cannot cross an intersection at the same time.
- There might be actions that a set of agents need to execute in parallel. For example, the action of lifting a table by two agents need to be done in parallel.

It turns out that the language developed in the previous section can be extended to represent and reason about plans/actions of agents in a multiagent environment. With the help of the notion of a planning problem with cooperative actions, a multiagent planning problem can be defined as follows.

Definition 5 (Multiagent planning problem). *A* multiagent planning problem $\mathcal{M}$ *is a tuple* $\langle \mathcal{AG}, \{\mathcal{P}_i\}_{i \in \mathcal{AG}}, \mathcal{F}, \mathcal{IC}, \mathcal{C} \rangle$ *where*

- $\mathcal{AG}$ *is a set of agents,*
- $\mathcal{P}_i$ *is a planning problem with cooperative actions for each agent* $i \in \mathcal{AG}$*,*
- $\mathcal{F}$ *is the set of tuples of the form* (i, j, f_i, f_j) *where* $i, j \in \mathcal{AG}$ *and* $f_i \in \mathtt{F}_i$ *and* $f_j \in \mathtt{F}_j$*, and*
- $\mathcal{IC}$ *and* $\mathcal{C}$ *are sets of sets of agent action pairs of the form* (i, a_i) *where* i *is an agent and* a_i *is an action in* $\mathtt{A}_i$*.*

Intuitively, each tuple (i, j, f_i, f_j) indicates that f_i and f_j represent the same state variable in the worlds of two agents i and j and can be changed by either i or j. This mean that they should have the same value in every state of i and j. A set of agent-action pairs $\{(i_1, a_{i_1}), \ldots, (i_k, a_{i_k})\} \in \mathcal{IC}$ indicates that the agents $i_1, \ldots, i_k$ cannot execute the actions $a_{i_1}, \ldots, a_{i_k}$ at the same time. On the other hand, a set of agent-action pairs $\{(i_1, a_{i_1}), \ldots, (i_k, a_{i_k})\} \in \mathcal{C}$ indicates that the agents $i_1, \ldots, i_k$ must execute the actions $a_{i_1}, \ldots, a_{i_k}$ concurrently for their effects to be materialized. The sets $\mathcal{F}$, $\mathcal{IC}$, and $\mathcal{C}$ are called constraints of $\mathcal{M}$.

Example 6. The planning problem in Example 1 can be represented by $\mathcal{M}_1 = \langle\{A, B, C\}, \{\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C\}, \emptyset, \emptyset, \emptyset\rangle$ where

- A, B, and C are the students from Example 1;
- $\mathcal{P}_A$ is defined as in Example 5;
- $\mathcal{P}_B = \langle DI_B, I_B, O_B, DC_B\rangle$ where DI_B is defined over $\mathtt{F}_B = \{h\_nail, h\_screw, diploma\_on, h\_ham\}$ and $\mathtt{A}_B = \{hw\_nail, hw\_screw\}$ with the set of laws:
  hw_nail **causes** $diploma\_on$ $\quad$ hw_nail **causes** $\neg h\_nail$
  hw_nail **executable** h_ham, h_nail $\quad$ hw_screw **causes** $diploma\_on$
  hw_screw **causes** $\neg h\_screw$ $\quad$ hw_screw **executable** h_screw
  $I_B = \{h\_nail, h\_screw, h\_ham, \neg diploma\_on\}$ and $O_B = \{diploma\_on\}$, and DC_B contains cooperative actions similar to that in DC_A and DC_C (below).
- $\mathcal{P}_C = \langle DI_C, I_C, O_C, DC_C\rangle$ where DI_C is defined over

$$\mathtt{F}_C = \{h\_nail, h\_screw, painting\_on\}$$

$$\mathtt{A}_C = \{hw\_screw\}$$

  with the set of laws: hw_screw **causes** $painting\_on$
  hw_screw **causes** $\neg h\_screw$ $\quad$ hw_screw **executable** h_screw
  $I_C = \{h\_nail, \neg h\_screw, \neg painting\_on\}$, $O_C = \{painting\_on\}$, and DC_C contains the following laws:

  $give\_me\_screw$ **requests** h_screw **from** $\{A, B\}$ **may_cause** h_screw **if** $\neg h\_screw$
  get_this_screw **provides** h_screw **for** $\{A, B\}$ **causes** $\neg h\_screw$ **if** h_screw □

On the other hand, the problem in Exp. 2 can be represented by $\mathcal{M}_2$ as follows.

Example 7. $\mathcal{M}_2 = \langle\{A, B\}, \{\mathcal{P}_A, \mathcal{P}_B\}, \mathcal{F}, \mathcal{IC}, \emptyset\rangle$ where $\mathcal{IC} = \{\{(A, flip_1), (B, flip_2)\}\}$, $\mathcal{F} = \{(A, B, light\_on, light\_on)\}$, and $\mathcal{P}_A$ and $\mathcal{P}_B$ are the planning problems of A and B, respectively, where

- $\mathcal{P}_A = \langle DI_A, I_A, O_A, \emptyset\rangle$ where DI_A is defined over $\mathtt{F}_A = \{light\_on\}$ and $\mathtt{A}_A = \{flip_1\}$ with the set of laws:

  $flip_1$ **causes** $light\_on$ **if** $\neg light\_on$ $\quad$ $flip_1$ **causes** $\neg light\_on$ **if** $light\_on$

  Finally, $I_A = \{\neg light\_on\}$ and $O_A = \{light\_on\}$.

- $\mathcal{P}_B = \langle DI_B, I_B, O_B, \emptyset \rangle$ where DI_B is defined over $\mathtt{F}_B = \{light\_on\}$ and $\mathtt{A}_B = \{flip_2\}$ with the set of laws:

  $flip_2$ **causes** $light\_on$ **if** $\neg light\_on$ $\qquad$ $flip_2$ **causes** $\neg light\_on$ **if** $light\_on$

  Finally, $I_B = \{\neg light\_on\}$ and $O_B = \{light\_on\}$. $\diamond$

We now define the notion of a solution for a planning problem.

Definition 6 (Joint plan for multiagents). *Let $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i \in \mathcal{AG}}, \mathcal{F}, \mathcal{IC}, \mathcal{C} \rangle$ be a multiagent planning problem. For each $i \in \mathcal{AG}$, let $S_i = [s_0^i a_0^i, \ldots, a_{n-1}^i s_n^i]$ be a possible plan of $\mathcal{P}_i$. We say that $\{S_i\}_{i \in \mathcal{AG}}$ is a* joint plan *(or* solution*) of length n for $\mathcal{M}$ if for every $0 \leq k \leq n$:*

- *for each instance of a request $a_k^i = r(\gamma, j)$ that is satisfied in S_i, we have that $a_k^j = p(\gamma, i)$;*
- *for each $(i, j, f_i, f_j) \in \mathcal{F}$, $f_i \in s_k^i$ iff $f_j \in s_k^j$;*
- *for each $S \in \mathcal{IC}$, there exists some $(i, a) \in S$ such that $a_k^i \neq a$; and*
- *for each $S \in \mathcal{C}$, either $\{a \mid (i, a) \in S \text{ and } a = a_k^i\} = \{a \mid (i, a) \in S\}$ or $\{a \mid (i, a) \in S \text{ and } a = a_k^i\} = \emptyset$.*

Intuitively, a joint plan is composed of individual plans which allow the agents to achieve their own goals and satisfy the various constraints of the problem. In the process, agents can help each other in establishing certain conditions. However, if a request of an agent is assumed (by the requestor) to be satisfied within a joint plan then the joint plan must also contain an agent who actually executes an offer action satisfying the request (first item). The second item states that the individual plans must agree with each other on their effects of shared fluents, i.e., it enforces the constraints in $\mathcal{F}$. The third and fourth items make sure that non-parallel and parallel constraints in $\mathcal{IC}$ and $\mathcal{C}$ are maintained by the joint plan.

Example 8. For the multiagent planning problem $\mathcal{M}_1$ from Example 6, We can easily check the following:

- for $n \leq 2$, $\mathcal{M}_1$ has no solution.
- for $n = 3$, it has a solution consisting of the following plans
  - $S_A = [s_0^A,\ give\_me\_nail(h\_nail, C),\ s_1^A,\ give\_me\_ham(h\_ham, B),$
    $s_2^A, hw\_nail,\ s_3^A,\ \mathtt{wait},\ s_4^A]$
  - $S_B = [s_0^B,\ hw\_nail, s_1^B,\ get\_this\_ham(h\_ham, A),$
    $s_2^B,\ get\_this\_screw(h\_screw, C),\ s_3^B,\ \mathtt{wait},\ s_4^B,]$
  - $S_C = [s_0^C,\ get\_this\_nail(h\_nail, A),\ s_1^C,\ \mathtt{wait}, s_2^C,\ give\_me\_screw(h\_screw, B),$
    $s_3^C,\ hw\_screw,\ s_4^C]$

  where all requests are satisfied and the states are uniquely determined by the initial states and the executed actions. $\diamond$

The joint plan for the agents in Example 8 requires that each agent executes some cooperative actions. It is easy to see that any joint plan for the two agents in the problem $\mathcal{M}_2$ requires that only one agent to flip the switch next to her and other agent to wait.

4 Computing Joint Plans

In this section, we will present different approaches to computing joint plans. Our approaches utilize answer set programming [18,19], a declarative programming paradigm that has recently emerged from the study of logic programming under the answer set semantics [11].

4.1 Answer Set Semantics of Logic Programs

A logic program Π is a set of rules of the form

$$a_0 \leftarrow a_1, \ldots, a_m, not\ \ a_{m+1}, \ldots, not\ \ a_n \tag{5}$$

where $0 \leq m \leq n$, each a_i is an atom of a propositional language[5] and not represents *negation-as-failure*. A negation as failure literal (or naf-literal) is of the form $not\ \ a$ where a is an atom. For a rule of the form (5), the left (right) hand sides of the rule are called the *head* (*body*), respectively. The head and the body can be empty (but not at the same time). A rule is a *constraint* if its head is empty; it is a *fact* if its body is empty.

Consider a set of ground atoms X. The body of a rule of the form (5) is *satisfied* by X if $\{a_{m+1}, \ldots, a_n\} \cap X = \emptyset$ and $\{a_1, \ldots, a_m\} \subseteq X$. A rule of the form (5) with nonempty head is satisfied by X if either its body is not satisfied by X or $a_0 \in X$. In other words, X satisfies a rule of the form (5) if its head belongs to X whenever X satisfies its body. A constraint is *satisfied* by X if its body is not satisfied by X.

For a set of ground atoms S and a program Π, the *reduct* of Π w.r.t. S, denoted by Π^S, is the program obtained from the set of all ground instances of Π by deleting

1. each rule that has a naf-literal $not\ a$ in its body with $a \in S$, and
2. all naf-literals in the bodies of the remaining rules.

S is an *answer set* of Π if it satisfies the following conditions.

1. If Π does not contain any naf-literal (i.e. $m = n$ in every rule of Π) then S is the smallest set of atoms that satisfies all the rules in Π.
2. If the program Π does contain some naf-literal ($m < n$ in some rule of Π), then S is an answer set of Π if S is the answer set of Π^S. (Note that Π^S does not contain naf-literals, its answer set is defined in the first item.)

A program Π is said to be *consistent* if it has an answer set. Otherwise, it is inconsistent. To make answer set style programming easier, Niemelä et al. [20] introduce a new type of rules, called *cardinality constraint rule* (a special form of the *weight constraint rule*) of the following form:

$$A_0 \leftarrow A_1, \ldots, A_m, not\ \ A_{m+1}, \ldots, not\ \ A_n$$

where each A_i is a *choice atom* of the form $l\{b_1, \ldots, b_k\}u$ with b_j are atoms and l and u are two integers, $l \leq u$; and A_0 can be empty. An atom $l\{b_1, \ldots, b_k\}u$ is said to be true wrt. a set of literals S iff $l \leq |S \cap \{b_1, \ldots, b_k\}| \leq u$. The satisfaction of a rule wrt.

[5] Rules with variables are viewed as a shorthand for the set of its ground instances.

a set of atoms is extended in the usual way. Using rules of this type, one can greatly reduce the number of rules of programs in answer set programming. The semantics of logic programs with such rules is given in [20].

4.2 Finding a Possible Plan for One Agent

We will represent each individual problem of each agent $\mathcal{P}_i$ by a logic program. The program will consist of rules describing the effects of actions, the initial knowledge of the agent, and the goal of the agent. Answer set planning [16] refers to the use of answer set programming in planning. This method has been applied to a variety of problems [10,26]. Let $\mathcal{P} = \langle DI, I, O, DC \rangle$ be a planning problem. We will now describe the program $\Pi(\mathcal{P})$ that encodes $\mathcal{P}$. We adapt the conventional style in logic programming: terms starting with lower-case letter are constant and others are variables. It also has a parameter denoting the maximal length of the plan that the agent considers permissible. The key predicates of $\Pi(\mathcal{P})$ are:

- $h(l,t)$ – fluent literal l holds at the time step t; and
- $o(a,t)$ – action a is executed (by the agent) at the time step t;
- $poss(a,t)$ – action a can be executed at the time step t.

$h(l,t)$ can be extended to define $h(\phi,t)$ for an arbitrary fluent formula ϕ, which states that ϕ holds at the time moment t. In writing the program, we use $h(\{l_1,\ldots,l_k\},T)$ as a shorthand for $h(l_1,T),\ldots,h(l_k,T)$. In addition, we write $ok(r(\gamma,i),t)$ to denote that the request-action $r(\gamma,i)$ is satisfied at the time step t. The rules of the program is divided into groups:

- *Group 1*: The program contains the following facts:

  $\{fluent(f) \mid f \in \mathtt{F}\} \cup \{action(a) \mid a \in \mathtt{A}\} \cup$
  $\{action(r(\gamma),i) \mid r$ occurring in a law of form (3), $i \in \mathcal{A}_\mathtt{i}\} \cup$
  $\{action(p(\gamma),i) \mid p$ occurring in a law of form (4), $i \in \mathcal{A}_\mathtt{i}\}$

  These facts declare the fluents and the actions of the problem.
- *Group 2*: rules for reasoning about effects of actions. For each action $a \in \mathtt{A}$,
  - if DI contains the law $(a$ **executable** $\phi)$ then $\Pi(\mathcal{P})$ contains the rules

  $$poss(a,T) \leftarrow h(\phi,T) \tag{6}$$
  $$\leftarrow o(a,T), not\ \ poss(a,T) \tag{7}$$

  - if DI contains the law $(a$ **causes** l **if** $\phi)$ then $\Pi(\mathcal{P})$ contains the rule

  $$h(l,T+1) \leftarrow o(a,T), h(\phi,T) \tag{8}$$

- *Group 3*: rules for reasoning about request-actions. For each statement of the form

  $$r \textbf{ requests } \gamma \textbf{ from } \mathcal{A}_\mathtt{i} \textbf{ may\_cause } \phi \textbf{ if } \psi$$

and each $i \in \mathcal{A}_\mathbf{i}$, $\Pi(\mathcal{P})$ contains the rules

$$poss(r(\gamma, i), T) \leftarrow h(\psi, T) \tag{9}$$
$$\leftarrow o(r(\gamma, i), T), not\ \ poss(r(\gamma, i), T) \tag{10}$$
$$0\ \{ok(r(\gamma, i), T + 1)\}\ 1 \leftarrow o(r(\gamma, i), T). \tag{11}$$
$$h(\phi, T) \leftarrow ok(r(\gamma, i), T) \tag{12}$$

where (12) is a shorthand for the collection of rules $\{h(l, T) \leftarrow ok(r(\gamma, i), T) \mid l \in \phi\}$. Observe that atoms of the form $ok(r(\gamma, i), T)$ are used to record the satisfaction of the request $r(\gamma, i)$ and there might be different ways for a condition γ to be satisfied. Hence, (11) and (12) need to be separated even though it looks like they could have been merged into one.

- *Group 4*: rules for reasoning about offer-actions. For each statement of the form

$$p\ \textbf{provides}\ \gamma\ \textbf{for}\ \mathcal{A}_\mathbf{i}\ \textbf{causes}\ \phi\ \textbf{if}\ \psi$$

and $i \in \mathcal{A}_\mathbf{i}$, $\Pi(\mathcal{P})$ contains the rules

$$poss(p(\gamma, i), T) \leftarrow h(\psi, T) \tag{13}$$
$$\leftarrow o(p(\gamma, i), T), not\ \ poss(p(\gamma, i), T) \tag{14}$$
$$h(\phi, T + 1) \leftarrow o(p(\gamma, i), T). \tag{15}$$

These rules are similar to the rules encoding the effect of individual actions of the agent. The difference between the encoding of a request-action and the encoding of an offer-action lies in that we do not need to introduce an atom of the form $ok(p(\gamma, i), T)$ to record the execution of $p(\gamma, i)$, i.e., effects of offer-actions are deterministic.

- *Group 5*: rules describing the initial state. For each literal $l \in I$, $\Pi(\mathcal{P})$ contains the fact $h(l, 0)$.
- *Group 6*: rules encoding the goal state. For each literal $l \in O$, $\Pi(\mathcal{P})$ contains

$$\leftarrow not\ \ h(l, n). \tag{16}$$

where n is the desired length of the plan.

- *Group 7*: rules for reasoning by inertial. For each fluent $F \in \mathtt{F}$, $\Pi(\mathcal{P})$ contains

$$h(F, T + 1) \leftarrow h(F, T), not\ \ h(\neg F, T + 1). \tag{17}$$
$$h(\neg F, T + 1) \leftarrow h(\neg F, T), not\ \ h(F, T + 1). \tag{18}$$
$$\leftarrow h(F, T), h(\neg F, T). \tag{19}$$

- *Group 8*: rules for generating action occurrences. $\Pi(\mathcal{P})$ contains the rule

$$1\ \{o(A, T) : action(A)\}\ 1 \leftarrow T < n. \tag{20}$$

which states that at any time step, the action must execute one of its actions[6].

[6] Since we assume that `wait` always belongs to the set of actions of an agent, this is not a strict requirement as it might sound.

Example 9. As an example, some of the rules encoding the problem $\mathcal{P}_A$ in Example 3 is given next

$$\begin{aligned} fluent(h\_nail) &\leftarrow \\ h(mirror\_on, T+1) &\leftarrow o(hw\_nail, T). \\ h(\neg h\_nail, T+1) &\leftarrow o(hw\_nail, T). \\ poss(hw\_nail, T) &\leftarrow h(h\_nail, T), h(h\_ham, T). \\ &\leftarrow o(hw\_nail, T), not\ \ poss(hw\_nail, T). \\ h(\neg h\_nail, 0) &\leftarrow \\ h(\neg h\_screw, 0) &\leftarrow \\ &\leftarrow not\ \ h(mirror\_on, n). \end{aligned}$$

The first rule defines the fluent h_nail. The next four rules encode the executability condition of the action hw_nail and its effects. The next two rules specify a part of the initial state and the last rule encodes the goal.

It is instructive to discuss the encoding of the two actions $give\_me\_nail$ and get_this_nail. For the action $give\_me\_nail$ and b (representing the agent B), we have the following rules:

$$\begin{aligned} &poss(give\_me\_nail(h\_nail, b), T) \leftarrow h(\neg h\_nail, T). \\ &\leftarrow o(give\_me\_nail(h\_nail, b), T), not\ \ poss(give\_me\_nail(h\_nail, b), T). \\ &0\ \{ok(give\_me\_nail(h\_nail, b), T+1)\}\ 1 \leftarrow o(give\_me\_nail(h\_nail, b), T). \\ &h(h\_nail, T) \leftarrow ok(give\_me\_nail(h\_nail, b), T). \end{aligned}$$

and for the action get_this_nail, we have the rules:

$$\begin{aligned} &poss(get\_this\_nail(h\_nail, b), T) \leftarrow h(h\_nail, T). \\ &\leftarrow o(get\_this\_nail(h\_nail, b), T), poss(get\_this\_nail(h\_nail, b), T). \\ &h(\neg h\_nail, T+1) \leftarrow o(get\_this\_nail(h\_nail, b), T). \end{aligned}$$

Let $\mathcal{P} = \langle DI, I, O, DC \rangle$ be a planning problem and $\Pi(\mathcal{P}, n)$ denote the set of ground rules of $\Pi(\mathcal{P})$ in which the variable T is instantiated with integers between 0 to n. Let M be an answer set of $\Pi(\mathcal{P}, n)$. Let $s_t[M] = \{l \mid l \text{ is a fluent literal and } h(l, t) \in M\}$. By $\alpha[M]$ we denote the sequence $s_0[M], a_0, s_1[M], \ldots, a_{n-1}, s_n[M]$ where $o(a_i, i) \in M$. We can show the following:

Theorem 1. *Let $\mathcal{P} = \langle DI, I, O, DC \rangle$ be a planning problem. Then,*

- *for each possible plan α of $\mathcal{P}$ there exists an n and an answer set M of $\Pi(\mathcal{P}, n)$ such that $\alpha = \alpha[M]$;*
- *for each n, if $\Pi(\mathcal{P}, n)$ has an answer set M then $\alpha[M]$ is a solution of $\mathcal{P}$; and*
- *for each n, if $\Pi(\mathcal{P}, n)$ is inconsistent then $\mathcal{P}$ does not have a solution of length less than or equal to n.*

Proof. (Sketch) The proof of the first two items is similar to the proof of Theorem 3.2 in [22] and relies on the following properties of an answer set M of $\Pi(\mathcal{P}, n)$:

– if $o(a,i) \in M$ then a is executable in $s_i[M]$ and $s_{i+1} \in \Phi_D(a, s_i[M])$; and
– O is satisfied by $s_n[M]$.

The last item is obvious given the first two items. □

4.3 Compatible Answer Sets and Joint Plan

Individual possible plans can be computed using the program $\Pi(\mathcal{P}_i)$. We will now discuss an approach for combining them to create a plan for all the agents. Intuitively, we need to make sure that if a request is assumed to be satisfied by an agent then there exists an instance of an offer-action matching this request. This can be easily characterized by the notion of a compatible answer sets.

Definition 7 (Compatible answer sets). *Let $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i\in\mathcal{AG}}, \mathcal{F}, \mathcal{IC}, \mathcal{C}\rangle$ be a multiagent planning problem and $M = \langle M_i\rangle_{i\in\mathcal{AG}}$ be a sequence of answer sets of $\langle \Pi(\mathcal{P}_i, n)\rangle_{i\in\mathcal{AG}}$ where the constant n is fixed. M is a set of* compatible *answer sets if for each $k \leq n$,*

- *for each $i \in \mathcal{AG}$, if $ok(r(\gamma, j), k+1) \in M_i$ then $o(p(\gamma, i), k) \in M_j$;*
- *for each $i \in \mathcal{AG}$, if $o(p(\gamma, j), k) \in M_i$ then $ok(r(\gamma, i), k+1) \in M_j$;*
- *for each (i, j, f_i, f_j) in $\mathcal{F}$, $h(f_i, k) \in M_i$ iff $h(f_j, k) \in M_j$;*
- *for each $S \in \mathcal{IC}$ there exists some $(i, a_i) \in S$ such that $o(a_i, k) \notin M_i$; and*
- *for each $S \in \mathcal{C}$, either $\{a_i | (i, a_i) \in S$ and $o(a_i, k) \in M_i\} = \{a | (i, a) \in S\}$ or $\{a_i | (i, a_i) \in S$ and $o(a_i, k) \in M_i\} = \emptyset$.*

Intuitively, a set of compatible answer sets corresponds to a joint plan (as we will prove in the next theorem) similar to the correspondence between answer sets and plans in the case of a single agent. Observe also that $ok(r(.), T)$ is present only due to the successfulness of a request-action, not an offer-action. The conditions imposed on a set of compatible answer sets make sure that the collection of individual plans extracting from them satisfies the constraints of the planning problem and the requirement that satisfying requests must be matched with offers.

Theorem 2. *Let $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i\in\mathcal{AG}}, \mathcal{F}, \mathcal{IC}\rangle$ be a multiagent planning problem and n be an integer.*

- *a sequence of answer sets $M = \langle M_i\rangle_{i\in\mathcal{AG}}$ is compatible iff there exists a solution $S = \langle \alpha_i\rangle_{i\in\mathcal{AG}}$ such that $\alpha[M_i] = \alpha_i$ for every $i \in \mathcal{AG}$.*
- *if $\langle \Pi(\mathcal{P}_i, n)\rangle_{i\in\mathcal{AG}}$ does not have a set of compatible answer sets then $\mathcal{M}$ does not have a solution with length n.*

Proof. (Sketch) The conclusion of the first item can be derived from the definition of compatibility answer sets, Theorem 1, and the definition of a solution. The conclusion of the second item follows from the first item and Theorem 1. □

Example 10. Let $\mathcal{M}_1$ be the multiagent planning problem from Example 6. We can easily check the following:

- $\{\Pi(\mathcal{P}_i, n)\}_{i\in\{A,B,C\}}$ for $n \leq 2$ does not have compatible answer sets,
- for $n = 3$, the three answer sets M_A, M_B, and M_C of $\Pi(\mathcal{P}_A, 3)$, $\Pi(\mathcal{P}_B, 3)$, and $\Pi(\mathcal{P}_C, 3)$, where
  - M_A contains $o(give\_me\_nail(h\_nail, c), 0)$, $ok(give\_me\_nail(h\_nail, c), 1)$, $o(give\_me\_ham(h\_ham, b), 1)$, $ok(give\_me\_ham(h\_ham, b), 2)$, $o(hw\_nail, 2)$, and $o(\texttt{wait}, 3)$.
  - M_B contains $o(hw\_nail, 0)$, $o(get\_this\_ham(h\_ham, a), 1)$, $o(get\_this\_screw(h\_screw, c), 2)$, $o(\texttt{wait}, 3)$; and
  - M_C contains $o(get\_this\_nail(h\_nail, a), 0)$, $o(\texttt{wait}, 1)$, $o(give\_me\_screw(h\_screw, b), 2)$, $ok(give\_me\_screw(h\_screw, b), 2)$, and $o(hw\_screw, 3)$.

These answer sets are compatible and correspond to the solution in Example 5. ◇

The notion of joint plan can be specialized as follows.

Definition 8 (Optimal Joint Plan). *Let $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i\in\mathcal{AG}}, \mathcal{F}, \mathcal{IC}, \mathcal{C}\rangle$ be a multiagent planning problem and $\{S_i\}_{i\in\mathcal{AG}}$ be a plan for $\mathcal{M}$. We say that $\{S_i\}_{i\in\mathcal{AG}}$ is* optimal *if there exists no unsatisfied request actions in $\{S_i\}_{i\in\mathcal{AG}}$.*

Remark 3. The program $\Pi(\mathcal{P}_i)$ can be easily adapted to generate only optimal plans. Indeed, the only modification that needs to be done is to replace the rule (11) with

$$ok(r(\gamma, i), T + 1) \leftarrow o(r(\gamma, i), T).$$

Intuitively, this rule states that the request $r(\gamma, i)$ is satisfied. Thus, if a joint plan is found it will not contain any unsatisfied requests, i.e., it must be optimal.

Definitions 6 and 7 provide us with a way for computing joint plans of length n for a planning problem $\mathcal{M}$. The process involves (*i*) computing a set $\{M_i\}_{i\in\mathcal{AG}}$ of answer sets, where M_i is an answer set of $\Pi(\mathcal{P}_i, n)$; and (*ii*) checking the compatibility of $\{M_i\}_{i\in\mathcal{AG}}$. In what follows, we discuss a method for doing it. This method computes a joint plan by (*a*) forming a program representing $\mathcal{M}$ from the programs representing the individual plans and the set of constraints in $\mathcal{M}$; and (*b*) extracting joint plan from answer sets of the new program. This method is useful if the planning problem $\mathcal{M}$ is known to an agent or a manager.

4.4 Computing Joint Plans by Answer Set Programming

Let $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i\in\mathcal{AG}}, \mathcal{F}, \mathcal{IC}, \mathcal{C}\rangle$ be a planning problem. We will define a program $\Pi(\mathcal{M})$ whose answer sets represent the solutions of $\mathcal{M}$. $\mathcal{M}$ is constructed from the programs $\Pi(\mathcal{P}_i)$ for $i \in \mathcal{AG}$ as follows. For each $i \in \mathcal{AG}$, let $\Pi^i(\mathcal{P}_i)$, referred as the *tagged version* of $\Pi(\mathcal{P}_i)$, be the program obtained from $\Pi(\mathcal{P}_i)$ by replacing every literal x in $\Pi(\mathcal{P}_i)$ with the atom x^i (e.g., $action(a)^i$ for $action(a)$, $h(f, t)^i$ for $h(f, t)$, etc.). The program $\Pi(\mathcal{M})$ consists of

- for each $i \in \mathcal{AG}$, the tagged version $\Pi^i(\mathcal{P}_i)$ of $\Pi(\mathcal{P}_i)$;
- for each tuple (i, j, f^i, f^j) in $\mathcal{F}$, the constraints

$$\leftarrow h^i(f^i, T), h^j(\neg f^j, T) \tag{21}$$
$$\leftarrow h^i(\neg f^i, T), h^j(f^j, T) \tag{22}$$

ensure that shared variables maintain their consistency.

- for each set $S = \{(i_1, a_1), \ldots, (i_k, a_k)\}$ in $\mathcal{C}$, the constraint

$$\leftarrow 0\ \{o^{i_1}(a_1, T), \ldots, o^{i_k}(a_k, T)\}\ k-1 \tag{23}$$

which makes sure that if a part of S is executed, i.e., $o(i_j, a_j)$ belongs to an answer set, then the whole set S is executed.

- for each set $\{(i_1, a_1), \ldots, (i_k, a_k)\}$ in $\mathcal{IC}$, the constraints

$$\leftarrow o^{i_1}(a_1, T), \ldots, o^{i_k}(a_k, T) \tag{24}$$

This constraint guarantees that not all the actions $a_1, \ldots, a_k$ are executed at the same time.

- for every pair of instance $r(\gamma, j)$ and $p(\gamma, i)$ of a request-action r (for γ) of an agent i and an offer-action p (for γ) of an agent j, the following constraints

$$\leftarrow o^i(r(\gamma, j), T), ok^i(r(\gamma, j), T+1), not\ \ o^j(p(\gamma, i), T) \tag{25}$$
$$\leftarrow o^j(p(\gamma, i), T), not\ \ o^i(r(\gamma, j), T) \tag{26}$$
$$\leftarrow o^j(p(\gamma, i), T), not\ \ ok^i(r(\gamma, j), T+1) \tag{27}$$

The first constraint makes sure that if i requests for γ from j and it is satisfied then j does indeed offer the service. The last two rules guarantee the converse.

For a set X of literals in the language of $\Pi(\mathcal{M})$, let $X|_i = \{a \mid a$ is a literal in the language of $\Pi(\mathcal{P}_i)$ and $a^i \in X\}$. We have:

Theorem 3. *Let $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i \in \mathcal{AG}}, \mathcal{F}, \mathcal{IC}, \mathcal{C} \rangle$ be a multiagent planning problem. M is an answer set of $\Pi(\mathcal{M}, n)$ iff there exists a set of compatible answer sets $\{M_i\}_{i \in \mathcal{AG}}$ such that $M|_i = M_i$.*

The proof of Theorem 3 relies on the Splitting Theorem for logic programs [17]. It is divided into two steps. First, it is proved for program without the constraints (21)-(27). The significance of this proposition is that it allows us to compute the solution of a multiagent planning problem by computing a single answer set of $\mathcal{P}(\mathcal{M})$. Since the problem of determining whether a propositional program has an answer set or not is NP-complete, the following holds.

Corollary 1. *Determining whether a solution of polynomial bounded length of a multiagent planning problem $\mathcal{M}$ exists or not is NP-complete.*

5 Related Works

Multiagent planning could be viewed as a special case of distributed problem solving [9]. In this respect, our work could be viewed as one in the Centralized Planning for Distributed Plans group according to the classification in [9]. This is achieved by the program $\Pi(\mathcal{M})$. Alternatively, the individual plans can also be computed distributedly and coordinated using the program consisting of the constraints (21)-(27) and the tagged versions of the individual answer sets.

Our main goal is to generate a joint plan for the agents before its execution. In this regards, our work differs from many distributed continual planning systems that were discussed in the survey [7] and many papers presented in the recent AAMAS conferences which concentrate on planning and replanning or dealing with unexpected events during the plan execution.

Our approach to generating a joint plan in this paper blends the two components "planning" and "coordination" in the equation

$$\boxed{\text{Multiagent planning} = \text{Planning} + \text{Coordination}}$$

presented in [6] into a single step. Furthermore, we employ a plan representation that allows for the coordination to be done by using time-steps presented in individual plans. This is different from several other systems in which partial order plans are used for plan representation and refinement planning is used for coordination (e.g., [4,3] or earlier works such as the Partial Global Planning framework).

We use answer set programming [16], a method that has been used for single agent planning [10,26], in computing the joint plan. The declarativeness and modularity of answer set programming make the process of computing the joint plan fairly simple and simplify the coordination of the plans[7]. Our work is similar to the spirit of that in [8] where an attempt is made to construct joint plan using SAT-based single agent planner. Nevertheless, our use of answer set programming does not require the development of additional algorithms to assemble the final joint plan.

In [2], a language based on PDDL for modeling multiagent planning problems has been proposed that allows for the specification of and reasoning about several features important for multiagent planning and execution such as concurrency, individual and mutual beliefs of agents, planning and acting with incomplete information, communication, continuous events, etc. A special agent, called `env`, is present in all problems for modeling the environment which may act "unpredictably". Our language is less expressive than the above mentioned language as our focus is solely on the generation of a joint plan prior to its execution. On the other hand, the semantics provided in this paper can be used to prove formal properties of plans as well as the correctness of the logic program encoding of multiagent planning problem.

We note that collaborative actions presented in this paper is also suitable for the modeling of multiagent planning with resources. Requesting for a resource and offering a resource can be modeled in a similar fashion to that of asking for and offering of a nail (Example 4). Since our focus is the generation of joint plan before execution, the

[7] Recall that this is achieved by simply adding the rules (21)-(27).

proposed language is different from the resource logic introduced in [5], whose focus was on the plan merging phase. The requesting/offering actions can be seen as special case of *negotiation actions* discussed in [27].

We would like to point out that we use $\mathcal{A}$ because of its simple semantics and its close relationship to PDDL, the language developed for describing planning problems [14]. This means that other extensions or variations of $\mathcal{A}$ (e.g,. $\mathcal{B}$, $\mathcal{C}$ [13], $\mathcal{E}$ [15]) could also be extended to formalize cooperative actions. Observe that there are subtle differences between request actions and non-deterministic actions. First, a cooperative action changes the world of other agents while a non-deterministic action does not. Second, a cooperative action does not change the world of the agent executing this action, while a non-deterministic action does. In this sense, a cooperative action of an agent is like an exogenous action for other agents. Thus, modeling cooperative actions using non-deterministic actions might not be the most natural way.

Finally, we would like to note that an extension of the STRIPS language has been considered for multiagent planning in [1]. In this framework, a multiagent planning problem is formulated as a *single problem* and agent identifiers are attached to the actions, which is different from what we proposed here. As such, the framework in [1] is only appropriate for domains where no privacy among agents is required. This is not an issue in our formulation.

6 Conclusions and Future Works

We extend the action language $\mathcal{A}$ to define a language for representing and reasoning about actions and their effects in presence of cooperative actions between agents. We define the notion of a plan with cooperative actions and use it in formalizing the notion of a joint plan. We use answer set programming to generate joint plans. We introduce the notion of a set of compatible answer sets and provide a translation of a multiagent planning problem to a logic program whose answer sets represent joint plans.

The work so far has focused on the development of a theoretical framework for generating joint plans using answer set programming. The encoding of the examples are available in the technical report version of this paper [25]. It is worth noting that we have been able to extended the work to allow negotiation in multiagent planning [24]. Our immediate goal for the future is to investigate the scalability and efficiency of the proposed method. The use of answer set programming allows us to easily incorporate preferences or domain knowledge in the generation of the joint plans [22,23]. Additionally, we would like to explore the use of more expressive languages (e.g., action languages with constraints and sensing actions) in representing and reasoning about joint-plans of multiagents by addressing various questions mentioned in [2]. This is because the method provides in Section 2 has proved to be very effective in the single agent case (e.g. [26]).

Acknowledgment

The first author acknowledges the partial support from the NSF grants IIS-0812267.

References

1. Boutilier, C., Brafman, R.I.: Partial-order planning with concurrent interacting actions. JAIR 14, 105–136 (2001)
2. Brenner, M.: Planning for Multiagent Environments: From Individual Perceptions to Coordinated Execution. In: Work. on Multiagent Planning & Scheduling, ICAPS, pp. 80–88 (2005)
3. Cox, J.S., Durfee, E.H.: An efficient algorithm for multiagent plan coordination. In: AAMAS 2005, pp. 828–835 (2005)
4. Cox, J.S., Durfee, E.H., Bartold, T.: A Distributed Framework for Solving the Multiagent Plan Coordination Problem. In: AAMAS, pp. 821–827. ACM Press, New York (2005)
5. de Weerdt, M., Bos, A., Tonino, H., Witteveen, C.: A resource logic for multi-agent plan merging. Ann. Math. Artif. Intell. 37(1-2), 93–130 (2003)
6. de Weerdt, M., ter Mors, A., Witteveen, C.: Multi-agent planning: An introduction to planning and coordination. In: Handouts of the Euro. Agent Summer School, pp. 1–32 (2005)
7. desJardins, M., Durfee, E.H., Ortiz, C.L., Wolverton, M.: A survey of research in distributed, continual planning. AI Magazine 20(4), 13–22 (1999)
8. Dimopoulos, Y., Moraitis, P.: Multi-agent coordination and cooperation through classical planning. In: IEEE/WIC/ACM/IAT, pp. 398–402. IEEE Comp. Society, Los Alamitos (2006)
9. Durfee, E.: Distributed Problem Solving and Planning. In: Multiagent Systems (A Modern Approach to Distributed Artificial Intelligence), pp. 121–164. MIT Press, Cambridge (1999)
10. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: Answer Set Planning under Action Costs. Journal of Artificial Intelligence Research 19, 25–71 (2003)
11. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Fujisaki, T., Furukawa, K., Tanaka, H. (eds.) Logic Programming 1988. LNCS, vol. 383, pp. 1070–1080. Springer, Heidelberg (1989)
12. Gelfond, M., Lifschitz, V.: Representing actions and change by logic programs. Journal of Logic Programming 17(2,3,4), 301–323 (1993)
13. Gelfond, M., Lifschitz, V.: Action languages. ETAI 3(6) (1998)
14. Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL — the Planning Domain Definition Language. Ver. 1.2. TR1165, Yale (1998)
15. Kakas, A.C., Miller, R., Toni, F.: E-RES. Reasoning about Actions, Events and Observations. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS (LNAI), vol. 2173, pp. 254–266. Springer, Heidelberg (2001)
16. Lifschitz, V.: Action languages, answer sets and planning. In: The Logic Programming Paradigm: a 25-Year Perspective, pp. 357–373. Springer, Heidelberg (1999)
17. Lifschitz, V., Turner, H.: Splitting a logic program. In: ICLP, pp. 23–38 (1994)
18. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: The Log. Prog. Paradigm: a 25-year Perspective, pp. 375–398 (1999)
19. Niemelä, I.: Logic programming with stable model semantics as a constraint programming paradigm. AMAI 25(3,4), 241–273 (1999)
20. Niemelä, I., Simons, P., Soininen, T.: Stable model semantics for weight constraint rules. In: Proc. Logic Programming and NonMonotonic Rreasong, pp. 315–332 (1999)
21. Parsons, S., Sierra, C., Jennings, N.R.: Agents that reason and negotiate by arguing. J. of Log. and Comp. 8(3), 261–292 (1998)
22. Son, T.C., Baral, C., Tran, N., McIlraith, S.: Domain-Dependent Knowledge in Answer Set Planning. ACM Transactions on Computational Logic 7(4) (2006)

23. Son, T.C., Pontelli, E.: Planning with Preferences using Logic Programming. Journal of Theory and Practice of Logic Programming (TPLP) 6, 559–607 (2006)
24. Son, T.C., Pontelli, E., Sakama, C.: Logic Programming for Multiagent Planning with Negotiation. In: ICLP (accepted, 2009)
25. Son, T.C., Sakama, C.: Reasoning and Planning with Cooperative Actions for Multiagents Using Answer Set Programming. Technical Report (2008)
26. Tu, P.H., Son, T.C., Baral, C.: Reasoning and Planning with Sensing Actions, Incomplete Information, and Static Causal Laws using Logic Programming. TPLP 7, 1–74 (2006)
27. Wooldridge, M., Parsons, S.: Languages for negotiation. In: Proceedings of ECAI (2000)

Social Commitments in Time: Satisfied or Compensated

Paolo Torroni, Federico Chesani, Paola Mello, and Marco Montali

DEIS, University of Bologna. V.le Risorgimento 2, 40136 Bologna, Italy

Abstract. We define a framework based on computational logic technology and on a reactive axiomatization of the Event Calculus to formalize the evolution of commitments in time. We propose a new characterization of commitments with time that enables a rich modeling of the domain, various forms of reasoning, and run-time and static verification.

1 Introduction

Social commitments are commitments made from an agent to another agent to bring about a certain property. In broad terms, a social commitment represents the commitment that an agent, called *debtor*, has towards another agent, called *creditor*, to bring about some property or state of affairs, which is the *subject* of the commitment. In some instantiations of this idea, such as [7,16], the subject of a commitment is a temporal logic formula.

Commitments are a well-known concept in Multi-Agent Systems (MAS) research [2,14]. Representing the commitments that the agents have to one another and specifying constraints on their interactions in terms of commitments provides a principled basis for agent interactions [15]. From a MAS modelling perspective, a role can be modelled by a set of commitments. For example, a seller in an online market may be understood as committing to its price quotes and a buyer may be understood as committing to paying for goods received. Commitments also serve as a natural tool to resolve design ambiguities. The formal semantics enables verification of conformance and reasoning about the MAS specifications [6] to define core interaction patterns and build on them by reuse, refinement, and composition.

Central to the whole approach is the idea of manipulation of commitments: their creation, discharge, delegation, assignment, cancellation, and release, since commitments are stateful objects that change in time as events occur. Time and events are, therefore, essential elements. Some authors distinguish between *base-level* commitments, written $\mathsf{C}(x, y, p)$, and *conditional* commitments, written $\mathsf{CC}(x, y, p, q)$ (x is the debtor, y is the creditor, and p/q are properties). $\mathsf{CC}(x, y, p, q)$ signifies that if p is brought out, x will be committed towards y to bring about q.

In this work we give emphasis to temporal aspects of commitments. We build from previous research by Mallya et al. [12,11]. In our opinion, they represent the best articulated research on time-enhanced commitments to date. The main

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 228–243, 2010.

idea in these articles is to extend a commitment framework with a way to describe time points and intervals, and alternative outcomes due to commitments extending into the uncertain future. The perspective on commitment-related temporal issues proposed by [12] mainly aims to capture the idea of validity of a commitment. Thus the previous notation $\mathsf{C}(x, p, y)$ is extended with existential and universal *temporal quantifiers*, which become prefixes of p. There are two types of quantification. By an existential quantification, $[t_1, t_2]p$, we mean that p is true at one or more moments in the interval beginning at t_1 and ending at t_2. By a universal quantification, $\overline{[t_1, t_2]}p$, we indicate instead that p is true at every moment in the interval beginning at t_1 and ending at t_2. Nestings are possible. For example, a commitment from x to y that q is going to hold at every moment in a week beginning on some day between the 1st and 20th of February could be written as follows: $\mathsf{C}(x, y, [01.02.2009, 20.02.2009](\overline{[t_{start}, t_{start} + 7days]}q))$.

This is an elegant approach which decouples the temporal quantification from the proposition, enabling reasoning about the temporal aspect without regard to the propositions' meaning. However, there are still some cases in which such a characterization is difficult to use in practical applications. The main problems are due to the lack of variables in temporal logic expressions, and from the separation between such expressions and the other parts of the represented knowledge. Among the aims of this work there is our intention to identify such cases and discuss them.

Along with a notation to express commitments, we need a language to express operations on commitments. For example, Yolum and Singh propose a notation based on the Event Calculus temporal representation language to describe commitment manipulation inside an operational framework [16]. Moreover, from a design perspective, we need an architecture in which a commitment notation, a temporal representation language and a specification and verification framework are given a specific role.

In this paper, we discuss our ongoing research about commitment frameworks. We start by introducing some issues regarding social commitment modeling, and define a number of desiderata for social commitment frameworks. We then propose a new notation for commitments and commitment specification programs: the Commitment Modeling Language ($\mathcal{CML}$). Finally, we outline an abstract commitment framework architecture and a concrete instance of it that supports $\mathcal{CML}$. In such an instance, temporal reasoning with commitments is operationalized using a reactive implementation of the Event Calculus and various verification tasks can be accomplished thanks to an underlying declarative, computational logic-based framework.

2 Some Issues Regarding Modeling

The following informal discussion is example-driven. Examples are mainly taken from the literature. We start by observing that in some cases Mallya et al.'s notation can be simplified, considering that to represent an existentially quantified time interval it is sufficient to represent a time point using a variable with a

domain. We then sketch a new possible notation that accommodates variables with domains and temporal modeling in several dimensions. Again, based on literature examples, we demonstrate the possible usage of rules to model conditional commitments. Finally we discuss time associated with commitment state changes and the issue of compensation.

2.1 Time Variables, Rules and Constraints

Let us analyze the scenario proposed by Mallya et al. in [12].

Example 1. A travel agent wishes to book an airline ticket to a certain destination, a rental car to use while there, and a hotel room at which to stay. Consider four situations:

- *Situation 1.1.* The travel agent wants the passenger to fly on a particular day while still reserving the right to choose any flight on that day. If the airline offers such a deal, it becomes committed to maintaining a condition—a booked ticket—over an extended time period.
- *Situation 1.2.* The car rental company offers a one-week free rental in January.
- *Situation 1.3.* A hotel offers an electronic discount coupon that expires today, but text on the coupon states that it can only be used during a future spring break. Note that in this case the commitment violates a constraint about time. In fact, the coupon expires before it can be used.
- *Situation 1.4.* The car rental company guarantees that its cars will not break down for at least two days, promising an immediate replacement if one does. However, if the company is closed on weekends, then a customer who rents a car on a Friday would not benefit from the warranty if the car broke down on Saturday. Thus in this case the car rental company offers a warranty that cannot be used during the period in which the warranty is valid. □

Following [12], we use the symbols h for hotel, g for guest, r for rental company, c for customer, a for airline and p for the proposition, subject of the commitment. How do we model the situations above using commitments?

Situation 1.1. Let p represent that a ticket booking is guaranteed. Thus, using an existential temporal quantifier, $[t_1, t_1 + 24hrs]p$, we express that a ticket booking is guaranteed for 1 day, as of t_1 [12].

However, in practical applications, it may be interesting to explicitly model the time at which the commitment is satisfied (e.g., when the ticket is issued). To this end, we could use an alternative notation, which associates p with a variable, and binds such a variable to a domain interval: $[T]p, T \in [t_1, t_1 + 24hrs]$. We write the commitment as follows:

$$\mathsf{C}(a, g, [T]p), t_1 \leq T, T \leq t_1 + 24hrs. \tag{1}$$

In this way, the commitment is satisfied if there is a possible value of T which falls in the range $[t_1, t_1 + 24hrs]$, and such a value can be used for further inferences.

Situation 1.2. Let p denote free rental, and t_1 January 1st. Thus, using a universal temporal quantifier, guaranteed free rental for 7 days as of time t_3 is denoted by $\overline{[t_3, t_3 + 7days]}p$. Then to express that such an interval $\overline{[t_3, t_3 + 7days]}$ is inside January, Mallya et al. [12] use a condition on t_3, namely $t_1 \leq t_3 \leq t_1 + 24days$, and they attach an existential temporal quantifier outside of the quantifier above: $[t_1, t_1 + 31days](\overline{[t_3, t_3 + 7days]}p), t_1 \leq t_3 \leq t_1 + 24days$.

Let us now use the notation introduced above, instead of existential temporal quantification. We obtain $[T, T + 7days]p, t_1 \leq T, T \leq t_1 + 24days$. Note that we simplified the notation. In particular, we do not need do distinguish any more between existentially/universally quantified time intervals, because all intervals are universally quantified, and we can drop the over-line notation. The resulting commitment is:

$$\mathsf{C}(r, c, [T, T + 7days]p), t_1 \leq T, T \leq t_1 + 24days. \tag{2}$$

Situation 1.3. Mallya et al. propose the following solution:

$$\mathsf{C}(h, c, [t_1, t_1 + 24hrs](\overline{[t_3, t_3 + 7days]}p)), t_1 + 24hrs < t_3,$$

where $t_1, t_1 + 24hrs$ is "today" (before spring break) and spring break starts on t_3 and lasts one week. In this way, we obtain two disjoint intervals. The commitment should be resolved before the end of the first interval in order not to be breached, however it can only be resolved during the second interval, which implies that it will be necessarily breached. An alternative notation for the same commitment is the following:

$$\mathsf{C}(h, t, [T_s, T_e]p), t_1 \leq T_s, T_e \leq t_1 + 7days, t_3 \leq T_s, T_e \leq t_3 + 24hrs. \tag{3}$$

In this way, we eliminate the need for nested intervals, and unresolvability can automatically be discovered by basic CLP inference [9].

Situation 1.3 shows that in a specific case, we can do away with nesting. In general, all existential temporal quantifiers can be mapped onto CLP domain restrictions, so the need for nesting intervals is only due to nested universal temporal quantifiers. An example of such a situation is the following:

Example 2. The car rental company offers a one-week free rental every month, for the whole 2009. □

In this case, we cannot do away with nested intervals. It is possible to extend Mallya et al.'s Solution 2 and write

$$\overline{[t_1, t_1 + 12months]}(\overline{[t_3, t_3 + 7days]}p), t_1 \leq t_3 \leq t_1 + 24days,$$

however that does not capture the "every month" concept, due to lack of domain variables. A different notation is needed. For example, we may use nested commitments, instead of nested intervals. Alternatively, if the "every month" concept is often used in the domain, we could define a new (non-binary) constraint and a dedicated propagation algorithm which ensures a very compact

notation and an efficient inference process. Non-binary (global) constraints are one of the prominent features of constraint programming frameworks. It may well be that a global constraints that we can use is already available off-the shelf.[1]

Situation 1.4. Mallya et al. propose the following solution:

$$\mathsf{C}(r, c, (\overline{[t_1, t_1 + 2days]}great\_car \vee [t_1, t_2]replace\_car)), t_2 < t_1 + 2days,$$

where *great_car* means that the car has not broken down, and *replace_car* represents the warranty that the rental company gives on the quality of the car, t_1 denotes the instant at which the car is rented on Friday and t_2 denotes the closing of the rental company on Friday. Using the framework presented in [12] is it possible to reason on this "warranty paradox" using CTL and realize that the warranty cannot be enjoyed if the car is rented on a Friday and it breaks down on Saturday.

Note that this modeling, however intuitive, may give rise to some counter-intuitive results. For example, c may decide to satisfy the commitment by replacing a perfectly functioning car with a broken car.

If we wish to follow Situation 1.4's description more literally, we should opt for a different formalization. For example, the commitment about the replacement car should only be a consequence of the car breaking down:

$$\mathsf{C}(r, c, [T]replace\_car) \leftarrow t_1 \leq T, T \leq t_2, \mathbf{H}(break\_down, T), T \leq t_1 + 2days \quad (4)$$

where by $\mathbf{H}(break\_down(T))$ we denote an event occurred ("**H**appened") at time T. Again, it is possible to reason on the "warranty paradox" using basic CLP inference. The result of such a reasoning would be a "real" validity interval for the warranty, which excludes Saturday.

Thus using a rule-based notation we can express many situations in a faithful way. In particular, it would be possible to express conditional commitments. However, there is another possible solution, based on the concept of *compensation*. A compensation is an action to be taken to recover from a situation of violation. To this end, we need to explicitly denote the state of commitments. For instance, we can write $[t]viol(\mathsf{C}(x, y, p))$ to indicate that a commitment has been violated at time t (due to an event occurring at time t which falsifies p, or due to the elapsing at time t of a time interval in which p was supposed to be verified). We obtain:

$$\mathsf{C}(r, c, [t_1, t_2]great\_car). \quad (5)$$

$$\mathsf{C}(r, c, [T_r]replace\_car) \leftarrow [T_b]viol(\mathsf{C}(r, c, [T_s, T_e]great\_car)), \quad T_s \leq T_b, T_b \leq T_s + 2days, T_b \leq T_e, T_b \leq T_r. \quad (6)$$

[1] See Beldiceanu and Carlsson's global constraints catalog, http://www.emn.fr/x-info/sdemasse/gccat/

(T_b is the time of break down, T_r is the time of replacement). Alternatively, using an extended notation, we could write:

$$compensate(r, c, [T_r]replace\_car, \mathsf{C}(r, c, [T_s, T_e]great\_car), T_b)) \leftarrow \\ T_s \leq T_b, T_b \leq T_s + 2days, T_b \leq T_e, T_b \leq T_r. \tag{7}$$

More on compensations below.

Note that we can easily refine the rules above to specify what "immediate replacement" means (1 day? 3 hours?), by posing another constraint between T_b and T_r, other than $T_b \leq T_r$.

2.2 Time of Commitments

Another temporal dimension could be interesting in many applications. It is the time of the commitment itself. To the best of our knowledge, this dimension has not been considered by previous research.

Example 3. Consider a university-like agent organization, in which agent x and faculty f belong to. There are roles with social responsibilities, which we can express by way of commitments. One such role is that of director of studies (*dos*). x has been appointed *dos* at Faculty f on October 29, 2008.

We can express that x has been appointed *dos* for 2009 at Faculty f, using a notation like:

$$\mathsf{C}(x, f, [01.01.2009, 31.12.2009]dos). \tag{8}$$

But how can we express that x has been appointed *dos* on October 29th 2008? This could be an important element of the domain. Consider a regulation that says that *a Faculty member that has been appointed director of studies cannot take more commitments in the Faculty.* The notation above does not permit to reason at this level. The closest approximation is probably: *a Faculty member cannot take more commitments in the Faculty while he is director of studies.* Or, we could resort to an additional commitment to express the appointment, beside the *dos* commitment. But this would complicate the model by increasing the number of commitments. A simple solution is to attach the duration of the commitment to the commitment itself:

$$[29.10.2008, T_{end}]active(\mathsf{C}(x, f, [01.01.2009, 31.12.2009]dos)). \tag{9}$$

2.3 Compensations

Contracts often involve deadlines and compensations. Usually, compensation actions are possibilities given to recover from a situation of violation. In a typical setting, a commitment not satisfied in time will not become satisfied by actions taken after the deadline, but it will instead incur in a further commitment from the debtor's side to take a compensation action. The extent of the compensation required may be subject to context-dependent conditions and be directly related to the time spent after the deadline before the compensation action is taken.

Example 4. According to the Italian civil code, the owners of real estate must pay taxes to the municipality (I.C.I.) between June 1 and June 16 of every tax year, for the property owned during the previous solar year. The Law allows the debtor who did not pay by the deadline to recover, by their own initiative, from such a violation by a procedure called *spontaneous revision*. The spontaneous revision procedure is permitted only if the debtor has not yet been officially notified about ongoing investigations related to such a violation. A spontaneous revision's compensation of a previous violation amounts to 3% of the amount not paid, which counts as a sanction, plus the legal interests on the amount not paid, which depend on the time elapsed between the I.C.I. payment deadline and the payment by spontaneous revision. □

To model this example, we can resort to a *compensate* notation like we did above. Let t_1 be June 1st, t_2 be June 16th, c a citizen, m a municipality, and let domain-dependent knowledge such as the interest rate $IRate$ and the amount of taxes to be paid by a citizen be defined by rules or facts such as $interest\_rate(0.025)$ and $ici(c, 100euro)$. A possible solution of Example 4 is the following:

$$\mathsf{C}(c, m, [T]pay\_ICI(Amt)), t_1 \leq T, T \leq t_2 \leftarrow ici(c, Amt). \quad (10)$$

$$compensate(c, m, [T_p]pay\_ICI(Amt), \mathsf{C}(c, m, [T_r]s\_rev(Amt_{new}))) \leftarrow$$
$$interest\_rate(IRate), Amt_{new} = Amt \times (1.03 + IRate \times (T_r - T_p)), \quad (11)$$
$$\neg \mathbf{H}(official\_notification(pay\_ICI(Amt)), T_n), T_n < T_r.$$

(s_rev stands for spontaneous revision, Amt for amount, and "=" is a CLP equality constraint).

Such examples are ubiquitous in legislation bodies, and in many domains in which contracts are used to establish rights and obligations of interacting parties. To be able to model such situations and reason about them, a notation should accommodate variables inside commitments and allow us to relate such variables with domains and expressions containing other variables.

Note that in this case *compensate* is syntactic sugar for an alternative and equally expressive notation. One could resort for instance to conditional commitments, and include original content (pay_ICI) and compensating content (s_rev) in one single CC-like fact. However, compensations could be defined in different ways depending on the domain. For example, one can introduce various degrees of violation, such as mild/serious violation, depending on whether an official notification has been issued or not. A commitment modeling framework should be flexible enough to accommodate all these needs. This notation helps to abstract away from the specific compensation semantics.

3 Desiderata for a Commitment Modeling Framework

The considerations made above suggest a number of desiderata for a commitment modeling framework that enables reasoning with time. The first two desiderata are taken from [12].

Time intervals. Contracts often involve time bounds. It should be possible to express such time bounds, in order to enable reasoning about satisfaction or breach of commitments in general.

Achievement and maintenance. Two kinds of commitment conditions are possible: achievement conditions (a ticket will be issued by the end of the day), and maintenance conditions (the car will work without breaking down for 2 days). They should both be accommodated.

Degrees of violation. It should be possible to reason about the extent of breach of a commitment, to capture ideas such as partial or mild violation of a contract.

Compensation. The language should enable associating commitments with compensation actions.

Time of commitment state changes. It should be possible to reason about the time a commitment assumes a particular state, e.g., the time a commitment is created, violated or discharged. The framework should enable reasoning about the state of commitments along time.

Meta-level reasoning. There could be commitments about commitments (and further nesting). The notation should accommodate contracts in which a commitment is about another commitment that will be created at some later point, or about some existing commitment.

Simplicity. The notation should be easy and at the same time rigorous. It should be possible to run automated reasoning tasks on commitment-based contract specifications. Some interesting reasoning tasks are: contract analysis, commitment tracking, and compliance verification.

Modularity. It should be possible to extend the commitment notation or modify the underlying theories and reasoning procedures in a modular way. Moreover, it should be possible to integrate a commitment framework with other domain knowledge, so as to enable reasoners and agents using commitments to reason using all available knowledge, possibly including ontological knowledge. Such an integration should preserve the modularity principle.

4 A New Notation for Social Commitments: $\mathcal{CML}$

We propose a new notation for social commitments. We call it $\mathcal{CML}$ (Commitment Modeling Language). To enable reasoning, we consider commitments as a part of a knowledge base. In particular, social commitments are specified inside $\mathcal{CML}$ programs ($CPrograms$), which could describe for example a contract.

A $\mathcal{CML}$ program is made of rules. A rule in the form

$$CRuleHead \leftarrow CRuleBody. \tag{12}$$

is used to define effects of events on the state of commitments. More specifically, the user can use such rules to define for instance which events create, discharge,

or break which commitments, in the style of [16]. The body defines the context, i.e., the conditions that must hold in order for an event to have an effect on the state of a commitment. If no context is defined, the rule is said to be a fact.

Atoms in the body of rules may be external predicates, not defined in the commitment specification program (this should be allowed because of the modularity principle), or they can be fluents modeling the state of commitments. Such a state can be associated with existentially quantified temporal variables or with universally quantified intervals.

The $\mathcal{CML}$ syntax, shown in Figure 1, is easily extensible to accommodate various models of commitment evolution. For example, *p_viol* has been introduced alongside *viol* and *active* as a possible new commitment state, and a new type of commitment operation, e.g., *compensate*, could be added to the language to provide the user with such a high-level abstraction.

$$
\begin{aligned}
CMLProgram &::= CRules \\
CRules &::= CRule[CRules] \\
CRule &::= CRuleHead\text{"."} \mid CRuleHead\text{" } \leftarrow \text{ "}CRuleBody\text{"."} \\
CRuleHead &::= OP_{\mathsf{C}}\text{"("}Terms\text{","}Commitment\text{")"} \\
OP_{\mathsf{C}} &::= =\text{ "}create\text{"}\mid\text{"}discharge\text{"}\mid\text{"}cancel\text{"}\mid\text{"}release\text{"}\mid\text{"}assign\text{"}\mid\text{"}delegate\text{"}\mid\ldots \\
CRuleBody &::= CRuleBodyElem[\text{","}CRuleBody] \\
CRuleBodyElem &::= holds\text{"("}Interval\ State\text{"("}Commitment\text{","}Time\text{"))"}\mid \\
&\quad\ Atom \mid Constraint \\
Commitment &::= \mathsf{C}\text{"("}Agent\text{","}Agent\text{","}[Interval]CAtom\text{")"} \\
Interval &::= \text{"["}TExpr[\text{","}TExpr]\text{"]"} \\
TExpr &::= Time\ OP_T\ Time \mid Time\ OP_T\ Duration \\
OP_T &::= \text{ "} + \text{"}\mid\text{" } - \text{ "} \\
Time &::= Date \mid Numeral \mid Variable \mid TExpression \\
Duration &::= Numeral\ Granularity \\
Granularity &::= \text{"}hrs\text{"}\mid\text{"}days\text{"}\mid\text{"}weeks\text{"}\mid\text{"}months\text{"}\mid\ldots \\
Agent &::= Term \\
Atom &::= Ident \mid Ident\text{"("}Terms\text{")"} \\
Term &::= Atom \mid Numeral \mid Variable \\
Terms &::= Term[\text{","}Term] \\
CAtom &::= Atom \mid Commitment \\
Constraint &::= Variable\text{" } \in \text{ "}Domain \mid Variable\ OP_{CLP}\ TExpr \\
State &::= \text{"}viol\text{"}\mid\text{"}p\_viol\text{"}\mid\text{"}active\text{"}\mid\ldots \\
OP_{CLP} &::= \text{ " } = \text{ "}\mid\text{" } \neq \text{ "}\mid\text{" } \leq \text{ "}\mid\text{" } \geq \text{ "}\mid\text{" } < \text{ "}\mid\text{" } > \text{ "} \\
Domain &::= Interval \mid Set
\end{aligned}
$$

Fig. 1. Syntax of $\mathcal{CML}$ programs

A sample $CProgram$, modeling Situation 1.4, is the following:

$$create(rent\_a\_car(T_c, T_e), \mathsf{C}(r, c, [T_c, T_c + 2days]great\_car)). \tag{13}$$

$$create(car\_broken(T_b), \mathsf{C}(r, c, [T_r]replace\_car)) \leftarrow T_r \leq T_b + 24hours, holds([T_b]viol(\mathsf{C}(r, c, [T_s, T_e]great\_car), T_b)). \tag{14}$$

Renting a car at time T_c until T_e creates a commitment that for 2 days as of T_c the car does not break down. The car breaking down at a time T_b creates a commitment that the car must be replaced within 24 hours of the incident, if the breakdown has caused a breach of commitment.

While $\mathcal{CML}$ provides very expressive constructs such as variables with domains, constraints and rules, on the other hand it does not explicitly accommodate temporal logic expressions, such as $p\mathcal{U}q$ or $\bigcirc p$. We are currently investigating whether and how temporal logic formulae can be mapped onto $\mathcal{CML}$ expressions.

Two fundamental aspects of commitment frameworks are manipulation and reasoning [15]. Manipulation operates on the state of commitments.

4.1 States of Commitments

Recently, many authors proposed different possible evolutions of the commitment state, from an initial state after creation, down to its satisfaction through discharge, delegation or cancellation operations, or else to its violation due to the occurrence of events that contradict the subject of the agreement represented by the commitment itself. For instance, in [7], Fornara and Colombetti propose the following set of states for a commitment: empty (e), cancelled (c), precommitment (p), conditional (cc), active (a), fulfilled (f), and violated (v). The states and their transitions are depicted in Figure 2.

5 Commitment Manipulation and Reasoning

Usually, once the conditions specified in the commitment are either satisfied (for an achievement commitment) or violated (for a maintenance commitment), the commitment assumes a final state. However, as discussed above, if we consider also relevant temporal aspects, such as deadlines, we could define a finer-grained

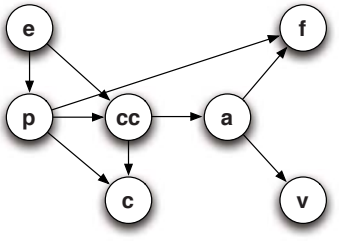

Fig. 2. Fornara & Colombetti's commitment state transitions [7]

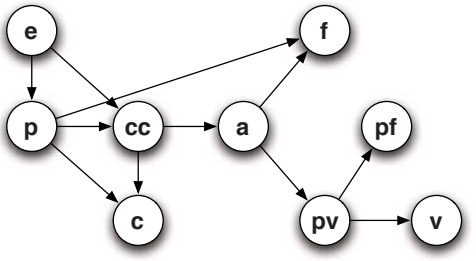

Fig. 3. A possible extension to a commitment state transitions accounting for partial violation (pv) and partial fulfillment (pf) of commitments

characterization of the state of a commitment. For example, after a deadline has passed, the debtor may still opt for a belated action that partially makes up for the violation. It may be interesting to distinguish among (1) commitment satisfied in time, (2) commitment "violated" before the deadline but "satisfied" after the deadline (partial violation/partial satisfaction), and (3) violated commitment. Such a distinction is depicted in Figure 3, which is an extended version of Figure 2.

This issue is discussed in depth by Fornara and Colombetti in [8], where the authors propose a new commitment life-cycle accommodating two states after *violated*: *extinguished* and *irrecoverable*. Our contribution here is not in the theoretical underpinning of sanctions and violations, but rather in building a framework where different theories of violation and sanction may be instantiated and operationalized.

5.1 Reasoning about Commitments

Yolum and Singh [16] propose to reason about commitments using the Event Calculus ($\mathcal{EC}$) [10]. The $\mathcal{EC}$ was introduced by Kowalski and Sergot as a logic programming framework for representing and reasoning about events and their effects. Basic concepts are that of *event*, happening at a point in time, and *property* (or *fluent*), holding during time intervals. Fluents are initiated/terminated by occurring events. There are many different formulations of the $\mathcal{EC}$ axioms. A simple one, taken from [5], is the one below (F stands for *Fluent*, Ev for *Event*).

$$holds\_at(F,T) \leftarrow initiates(Ev,F,T_{Start}) \wedge T_{Start} < T \wedge \neg clipped(T_{Start},F,T). \quad (ec_1)$$

$$clipped(T_1,F,T_3) \leftarrow terminates(Ev,F,T_2) \wedge T_1 < T_2 \wedge T_2 < T_3. \quad (ec_2)$$

$$initiates(Ev,F,T) \leftarrow happens(Ev,T) \wedge holds(F_1,T) \wedge ... \wedge holds(F_N,T). \quad (ec_3)$$

$$terminates(Ev,F,T) \leftarrow happens(Ev,T) \wedge holds(F_1,T) \wedge ... \wedge holds(F_N,T). \quad (ec_4)$$

Axioms ec_1 and ec_2 are the general ones of $\mathcal{EC}$, whereas ec_3 and ec_4 are user-defined, domain-specific axiom schemas. The $\mathcal{EC}$ is a suitable formalism to specify the effects of commitment manipulation, and reason from such operations. As a sample fragment of Yolum and Singh's formalization, consider a *create* operation, whose purpose is to establish a commitment, and can only be performed by the debtor. To express that an event $e(x)$ carried out by x at time t creates a commitment $\mathsf{C}(x, y, p)$, Yolum and Singh define the operation $create(e(x), \mathsf{C}(x, y, p))$ in terms of $happens(e(x), t) \wedge initiates(e(x), \mathsf{C}(x, y, p), t)$.

In the same way, the semantics of $\mathcal{CML}$ can be given in terms of $\mathcal{EC}$ programs. This helps *simplicity*, because the language of $\mathcal{EC}$ is very simple, and *modularity*, because for different domains we can define different theories of commitments.

The $\mathcal{EC}$ is an effective framework for temporal reasoning. It has been extensively used in the literature to carry out two main reasoning tasks: deductive *narrative verification*, to check whether a certain fluent holds given a narrative (set of events), and abductive *planning*, to simulate a possible narrative which satisfies some requirements [13]. Chittaro and Montanari [4] proposed a way to use the $\mathcal{EC}$ for run-time monitoring and verification. It is based on a mechanism to cache the outcome of the inference process every time the knowledge base is updated by a new event. In a nutshell, the *Cached Event Calculus* ($\mathcal{CEC}$) computes and stores fluents' *maximum validity intervals* (MVIs), which are the maximum time intervals in which fluents hold, according to the known events. The set of cached validity intervals is then extended/revised as new events occur or get to be known. Therefore, the $\mathcal{EC}$ can be used as a basis for reasoning on commitments in many ways, including not only planning and static verification, but also tracking, depending on the $\mathcal{EC}$ implementation used (abductive, deductive, reactive).

6 Social Commitment Framework Architecture

We propose an abstract, layered architecture that enables modeling and reasoning with social commitments. It consists of:

- a user application layer;
- a commitment modeling layer;
- a temporal representation and reasoning layer;
- a reasoning and verification layer.

On the top layer, the user can define contracts or agent social interaction rules using commitments. Such definitions are based on a language provided by the layer below. The commitment modeling language is implemented using a temporal representation and reasoning framework, which is in turn built on top of a more general reasoning and verification framework, which lies at the bottom layer. It is important to rely on a formal framework that accommodates various forms of verification, because in this way commitments can be operationalized and the user can formally analyze commitment-based contracts, reason on the state of commitments, plan for actions needed to reach states of fulfillment, and track the evolution of commitments at run-time.

Indeed, the underlying reasoning and verification layer must be powerful enough to implement a temporal representation and reasoning layer. We propose a concrete instance of such an architecture, represented in Figure 4.

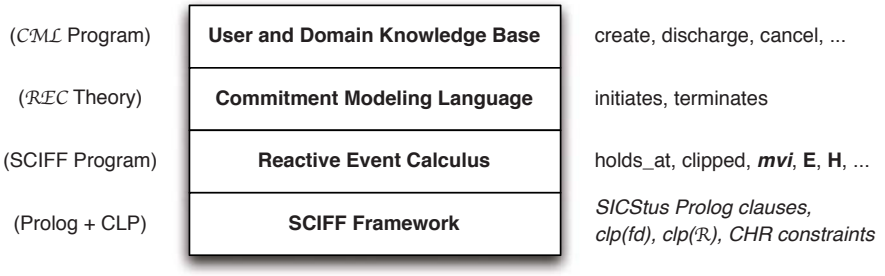

Fig. 4. Social commitment framework architecture

At the bottom layer, we find a number of Prolog+CLP modules which implement the SCIFF family of proof-procedures and provide the SCIFF language to the layer above [1]. The SCIFF framework is based on abductive logic programming and it consists of a declarative specification language and a family of proof-procedures for reasoning from SCIFF specifications. Some kinds of reasoning are: deduction, hypothetical reasoning, static verification of properties, compliance checking and run-time monitoring. In general, SCIFF comes in hand for a number of useful tasks in the context of agent interaction. A high-level description of SCIFF and of its usage is given in [15], also in relation with commitments. The CLP solvers integrated in SCIFF can work with discrete and dense domains, depending on the application needs, and they are particularly useful for reasoning along the temporal dimension.

On top of the SCIFF layer there is a SCIFF implementation of the $\mathcal{EC}$, which uses ideas taken from $\mathcal{CEC}$ and thus enables runtime verification. It is called the Reactive Event Calculus ($\mathcal{REC}$). This layer provides to the layer above the $\mathcal{REC}$ language, which consists of domain-dependent axioms with schemas ec_3 and ec_4.

In the third layer, the constructs that define the Commitment Modeling Language ($\mathcal{CML}$), i.e., the notation proposed above, are written by way of $\mathcal{REC}$ theories. Thus this layer will provide the top layer with the language to write a $CProgram$. The top layer consists of user and domain-dependent knowledge encoded into a $CProgram$. An example of a program for the top layer was given in Section 4.

We believe that such an architecture, and its instantiation based on SCIFF, $\mathcal{REC}$, and the $\mathcal{CML}$, can successfully address the desiderata identified above. Modularity is achieved in two directions: in the vertical direction, by making $\mathcal{CML}$ programs, $\mathcal{EC}$ theory, and commitment theories independent of each other, and in the horizontal direction, by letting the user refer to external inputs by way of simple atoms. Atoms can be mapped into function calls via suitable interfaces such as those available in most Prolog engines. $\mathcal{CML}$ is a simple and easily

extensible language, which consists of primitives such as *create*, *discharge*, etc., in the style of [16]. The language is expressive enough to express time intervals, achievement and maintenance conditions, and time of commitment state change. Thanks to the expressivity of the language and to the modularity of the architecture, it is possible to extend the framework to model different kinds of violation and powerful new constructs such as compensation. In fact, the states of commitments and manipulation operations are not hard-wired in the architecture, but they can be (re)defined by the user. Finally, $\mathcal{CML}$ accommodates meta-lavel reasoning on commitments, and the underlying $\mathcal{REC}$ engine can reason about commitments at all levels by treating a commitment state as a fluent which holds for a period of time.

7 Conclusion

We identified some issues regarding the representation of commitments which are still open, and we formulated a number of desiderata for a commitment modeling framework. To the best of our knowledge, in the state of the art there is no framework that satisfies all the desiderata. We believe that a possible answer could come from a commitment framework organized into four layers.

On top of the stack, at the user level, contracts can be specified by way of commitment programs. We identified in SCIFF a potential candidate for the bottom layer and we defined a notation for top-level programs. A prototypical implementation exists of all the four layers.[2] Its usage for commitment tracking purposes is demonstrated in a companion paper [3].

Our discussion was informal and example-driven. We gave emphasis to temporal aspects of commitments in relation with deadlines. However, deadlines are only a special case of temporal constraints and CLP constraints in general. Surely there are many other types of constraint that could be very useful for modeling the domain correctly and compactly. In particular, global constraints capture common patterns and help specify complex and recurring constraints in a simple way. Each global constraint comes with an effective and efficient propagation algorithm capable of powerful inference. A useful activity could be to isolate a subset of CLP constraints of interest for commitment-related domains. A concrete implementation of the commitment modeling framework should include a library of such CLP constraints. To the best of our knowledge, the inference potential of such a technology, unlocked by the architecture we propose, is unprecedented in the domain of commitments.

Currently we are evaluating the $\mathcal{CML}$ language and framework empirically on a number of case studies. Some of the formal properties of the $\mathcal{CML}$ framework, in particular insofar as reasoning is concerned, are discussed in [3]. We plan to focus not on the expressivity of the top-level commitment programming language and on the notion of sanctions. To this extent, we found more recent work by Fornara and Colombetti [8] inspiring and very relevant to our approach. Along this line, we also intend to investigate how $\mathcal{CML}$ and the abstract architecture

[2] http://lia.deis.unibo.it/research/sciff/

fit into the concrete application domain of electronic institutions. Commitments or obligations are also included in their modeling, and often they work with deadlines of events taking place instead of time.

Acknowledgments. We thank the anonymous reviewers for their useful comments. This work has been partially supported by the FIRB project *TOCAI.IT*.

References

1. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. ACM Transactions on Computational Logic 9(4), 1–43 (2008)
2. Castelfranchi, C.: Commitments: From individual intentions to groups and organizations. In: Lesser, V.R., Gasser, L. (eds.) Proceedings of the First International Conference on Multi-Agent Systems, pp. 41–48. The MIT Press, Cambridge (1995)
3. Chesani, F., Mello, P., Montali, M., Torroni, P.: Commitment tracking via the Reactive Event Calculus. In: Boutilier, C. (ed.) Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI), pp. 91–96. AAAI Press, Menlo Park (2009)
4. Chittaro, L., Montanari, A.: Efficient temporal reasoning in the cached event calculus. Computational Intelligence 12(2), 359–382 (1996)
5. Chittaro, L., Montanari, A.: Temporal representation and reasoning in artificial intelligence: Issues and approaches. Annals of Mathematics and Artificial Intelligence 28(1-4), 47–106 (2000)
6. Fisher, M., Bordini, R.H., Hirsch, B., Torroni, P.: Computational logics and agents: A road map of current technologies and future trends. Computational Intelligence 23(1), 61–91 (2007)
7. Fornara, N., Colombetti, M.: Operational specification of a commitment-based agent communication language. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 536–542. ACM Press, New York (2002)
8. Fornara, N., Colombetti, M.: Specifying and enforcing norms in artificial institutions. In: Normative Multi-Agent Systems. Dagstuhl Seminar Proceedings, vol. 07122 (2007), http://drops.dagstuhl.de/opus/volltexte/2007/909
9. Jaffar, J., Maher, M.: Constraint logic programming: a survey. Journal of Logic Programming 19-20, 503–582 (1994)
10. Kowalski, R.A., Sergot, M.: A logic-based calculus of events. New Generation Computing 4(1), 67–95 (1986)
11. Mallya, A.U., Huhns, M.N.: Commitments among agents. IEEE Internet Computing 7(4), 90–93 (2003)
12. Mallya, A.U., Yolum, P., Singh, M.P.: Resolving commitments among autonomous agents. In: Dignum, F.P.M. (ed.) ACL 2003. LNCS (LNAI), vol. 2922, pp. 166–182. Springer, Heidelberg (2004)
13. Shanahan, M.: An abductive event calculus planner. Journal of Logic Programming 44(1-3), 207–240 (2000)

14. Singh, M.P.: An ontology for commitments in multiagent systems: Toward a unification of normative concepts. Artificial Intelligence and Law 7, 97–113 (1999)
15. Torroni, P., Yolum, P., Singh, M.P., Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P.: Modelling interactions via commitments and expectations. In: Dignum, V. (ed.) Handbook of Research on MAS: Semantics and Dynamics of Organizational Models, Hershey, Pennsylvania, March 2009, pp. 263–284. IGI Global (2009)
16. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: applying event calculus planning using commitments. In: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 527–534. ACM Press, New York (2002)

Verifying Dribble Agents

Doan Thu Trang, Brian Logan, and Natasha Alechina

The University of Nottingham
School of Computer Science

Abstract. We describe a model-checking based approach to verification of programs written in the agent programming language Dribble. We define a logic (an extension of the branching time temporal logic CTL) which describes transition systems corresponding to a Dribble program, and show how to express properties of the agent program in the logic and how to encode transition systems as an input to a model-checker. We prove soundness and completeness of the logic and a correspondence between the operational semantics of Dribble and the models of the logic.

1 Introduction

BDI-based agent-oriented programming languages [5] facilitate the implementation of cognitive agents by providing programming constructs to implement concepts such as beliefs, goals, and (pre-defined) plans. In such languages, an agent selects a plan to achieve one or more goals based on its beliefs about the environment. However in anything other than toy environments, selecting an appropriate plan does not guarantee that it can be successfully executed. The beliefs used to select a particular plan for a given goal is only a heuristic, and cannot capture the preconditions of all the actions in the plan (some of which may be false when the plan is selected and will only be made true by actions in the plan). Moreover, in dynamic environments, an agent's beliefs (and hence the best way of achieving a goal) may change in unanticipated ways during plan execution, and a rational agent must be prepared to revise its plans at run time to take advantage of 'fortuitous' changes in the environment (e.g., which allow some steps in the plan to be skipped) or to recover from 'adverse' changes in the environment (e.g., when a precondition of an action is discovered not to hold).

Many BDI-based agent programming languages provide facilities to drop plans if the corresponding goal is 'unexpectedly' achieved or when execution of the plan fails [6,16,17]. More advanced languages, e.g., [18,9] provide support for arbitrary modification of plans during their execution. However, while such meta-level capabilities simplify the development of rational agents, they make it more difficult to reason about the execution of agent programs, e.g., to verify their correctness. In addition to reasoning about the agent's beliefs, goals and plans, we need to model the current state of plan execution, and the evolution of the agent's program at run time in response to interactions between the effects the agent's actions in its environment and its plan revision capabilities.

In this paper we present an approach to verifying agent programs which admit arbitrary revisions at run time. We focus on the BDI agent programming language Dribble

M. Baldoni et al. (Eds.): DALT 2009, LNAI 5948, pp. 244–261, 2010.

introduced in [18]. Dribble allows the implementation of agents with beliefs, (declarative) goals, actions, abstract actions (procedural goals), plans, and rules for selecting and revising plans. Although relatively simple and abstract, it is representative of a wider class of BDI agent programming languages which support plan revision, and presents significant challenges for verification. Our approach is based on model-checking. We define a logic (an extension of the branching time temporal logic CTL) which describes transition systems corresponding to a Dribble program, and show how to express properties of the program in the logic and how to encode transition systems as an input to a model-checker. We prove soundness and completeness of the logic and a correspondence between the operational semantics of Dribble and the models of the logic.

The rest of the paper is organised as follows. In the next section, we describe the syntax and operational semantics of Dribble. In section 3 we introduce a logic for expressing properties of Dribble programs, and give a complete axiomatisation of the set of models corresponding to the operational semantics of a Dribble agent program. We discuss the use of the logic for verification in section 4, where we describe model-checking of Dribble programs and give a simple example of a program and a property which can be model-checked. We give a brief survey of related work in section 5.

2 Dribble

In this section, we briefly review the syntax and operational semantics of Dribble.

2.1 Beliefs and Goals

Let $Prop$ be a finite set of propositional variables and $\mathcal{L}$ the set of propositional formulas. In order to make $\mathcal{L}$ finite, we allow $\mathcal{L}$ to contain only formulas in Disjunctive Normal Form (DNF). A formula is said to be in DNF iff it is a disjunction of conjunctive clauses in which a conjunctive clause is a conjunction of literals. As usual, a literal is either p or $\neg p$ for any $p \in Prop$. Moreover, formulas of $\mathcal{L}$ satisfy the following conditions:

1. formulas do not contain duplicates of conjunctive clauses;
2. conjunctive clauses of a formula do not contain duplicates of literals; and
3. literals in a conjunctive clause of a formula only occur in some fixed order.

A Dribble agent has both a belief base and a goal base which are finite subsets of $\mathcal{L}$. The agent's beliefs and goals are expressed in a language $\mathcal{L}_{BG}$. The syntax of $\mathcal{L}_{BG}$ is defined as follows:

$$\beta \leftarrow \mathbf{B}\alpha \mid \mathbf{G}\alpha \mid \neg\beta \mid \beta_1 \wedge \beta_2 \text{ where } \alpha \in \mathcal{L}.$$

The meaning of $\mathbf{B}\alpha$ is that α can be propositionally derived from the belief base of an agent, and $\mathbf{G}\alpha$ means that α is the consequence of some single goal in the goal base of an agent. For convenience, a formula β is of $\mathcal{L}_B$ ($\mathcal{L}_G$) iff it does not contain any subformula of the form $\mathbf{G}\alpha$ ($\mathbf{B}\alpha$, respectively).

A formula of $\mathcal{L}_{BG}$ is interpreted by a pair of a belief base and a goal base $\langle \delta, \gamma \rangle$, in which both δ and γ are finite subsets of formulas of $\mathcal{L}$. The truth of a formula β is defined inductively as follows.

- $\langle \delta, \gamma \rangle \models_{BG} \mathbf{B}\alpha \Leftrightarrow \delta \models_{Prop} \alpha$
- $\langle \delta, \gamma \rangle \models_{BG} \mathbf{G}\alpha \Leftrightarrow \exists g \in \gamma : g \models_{Prop} \alpha$
- $\langle \delta, \gamma \rangle \models_{BG} \neg\varphi \Leftrightarrow \langle \delta, \gamma \rangle \not\models_{BG} \varphi$
- $\langle \delta, \gamma \rangle \models_{BG} \beta \wedge \beta' \Leftrightarrow \langle \delta, \gamma \rangle \models_{BG} \beta$ and $\langle \delta, \gamma \rangle \models_{BG} \beta'$

2.2 Plans

A Dribble plan consists of basic actions and abstract plans composed by sequence and conditional choice operators. The sequence operator, ';', takes two plans, π_1, π_2, as arguments and states that π_1 should be performed before π_2. The conditional choice operator allows branching and generates plans of the form '`if` ϕ `then` π_1 `else` π_2'. The syntax of plans is defined as follows:

$$\pi \leftarrow a \mid b \mid \textit{if } \beta \textit{ then } \pi'_1 \textit{ else } \pi'_2 \mid \pi'_1; \pi'_2$$

where a is an abstract plan, b is a basic action and $\beta \in \mathcal{L}_B$.

We depart from [18] in that we do not have an empty plan (denoted by E in [18]) as a special kind of plan which can occur as part of other plans. Below, we will use E as a marker for an empty plan base, but not as a plan expression, to avoid introducing rewriting rules for $E; E$ to E and $\pi_1; E; \pi_2$ to $\pi_1; \pi_2$, etc.

We define length of a plan π, $len(\pi)$, inductively as follows:

$$\begin{aligned} len(a) &= 1 \\ len(b) &= 1 \\ len(\textit{if } \beta \textit{ then } \pi'_1 \textit{ else } \pi'_2) &= len(\pi'_1) + len(\pi'_2) + 4 \\ len(\pi'; \pi) &= len(\pi') + len(\pi) \end{aligned}$$

Notice that in the case of the if-then-else statement, the length is the sum of lengths of the plans π'_1 and π'_2 together with the number of extra symbols of the statement, i.e. if, $then$, $else$ and β.

Since in reality, agents can hold a plan up to some fixed length, we make an assumption that all plans have length smaller than a certain preset number. Restricting the length of plans also makes the set of plans finite. This is necessary for the axiomatisation of the logic later in the paper.

In the rest of this paper, we denote by $Plans$ the set of all plans whose lengths are smaller than len_{MAX}, where len_{MAX} is a natural number.

$$Plans = \{\pi \mid len(\pi) \leq len_{MAX}\}$$

2.3 Dribble Agents

Writing a Dribble agent means writing a number of goal rules and practical reasoning rules. The syntax of goal rules (PG) and practical reasoning (PR) rules is given below.

- PG rules: $\beta \rightarrow \pi$ where $\beta \in \mathcal{L}_{BG}$ and $\pi \in Plans$
- PR rules: $\pi_1 \mid \beta \rightarrow \pi_2$ where $\beta \in \mathcal{L}_B$ and $\pi_1, \pi_2 \in Plans$, and π_2 may be empty.

One writes a PG rule to intend that an agent with an empty plan base will generate a plan π if its current belief and goal bases satisfy the condition encoded in β. If the agent has certain goals in its goal base, it will generate a plan based on its beliefs to hopefully achieve those goals. A PR rule proposes a possible revision π_2 to (the prefix of) a plan π_1 which is applicable if the belief base satisfies the condition encoded in β. That is, if the agent has certain beliefs which imply that the current plan will be unable to achieve the intended goal(s) or that the plan is redundant and can be simplified, it can modify the plan. Note that π_2 can be empty, allowing the agent to drop part or all of a plan.

We have slightly modified the meaning of PR rules given in [18]. In Dribble, these rules apply to complete plans (π_1 is the agent's plan in its entirety, not a plan prefix, for example a name for an abstract plan). In contrast we allow π_1 to be a prefix of the agent's plan, which is replaced by π_2 followed by the continuation of the original plan. We could have written PR rules as $\pi_1'; \pi \mid \beta \rightarrow \pi_2; \pi$ where π is a plan variable. In cases where π_1 matches the entire plan, our PR rules are equivalent to those in [18]. We believe that our generalisation is justified programmatically, and it presents an interesting challenge for logical formalisation, in particular model-checking. To enforce our assumption about the length of plans, we require that Dribble agents consist of PG and PR rules which do not produce plans of length more than len_{MAX}.

A Dribble agent only has one intention at a time, i.e., its plan base contains at most one plan and it can apply a goal rule only when its plan is empty, and is strongly committed to its goals, i.e., an agent drops a goal only when it believes that the goal has been achieved.

A **Dribble agent** is a tuple $\langle \delta, \gamma, \Gamma, \Delta \rangle$ in which Γ is a set of goal rules, Δ is set of practical reasoning rules, δ and γ are the initial belief base and goal base and both satisfy the following conditions:

1. δ is consistent
2. $\forall \alpha \in \gamma, \delta \not\models_{Prop} \alpha$
3. $\forall \alpha \in \gamma, \alpha$ is consistent

that is, the agent's beliefs are consistent, it does not have as a goal any formula it already believes to be the case, and each of its goals is consistent (though its goals may be inconsistent with each other as they can be achieved at different times).

2.4 Operational Semantics

In this section, we describe how a Dribble agent operates. A Dribble program P is a pair (Γ, Δ) of PG rules and PR rules.

A configuration of an agent is a tuple $\langle \delta, \gamma, \{\pi\} \rangle$ where δ, γ and π are the agent's current belief base, goal base and plan base (where π is the current plan, possibly partially executed), respectively. In what follows, we will omit the set brackets around the plan for readability, as in $\langle \delta, \gamma, \pi \rangle$. The plan base can also be empty, which we will write as $\langle \delta, \gamma, \emptyset \rangle$. The *initial configuration* of an agent is $\langle \delta_0, \gamma_0, \emptyset \rangle$.

We specify the operational semantics of a Dribble agent as a set of transition rules. Each transition corresponds to a single execution step and takes the system from one configuration/state to another. In the cases corresponding to applying PG and PR rules,

we have additional conditions to guarantee that we do not produce a plan of length more than len_{MAX}. Notice that transitions from a configuration in which the plan base begins with an abstract plan are included in application of PR rules.

Application of a goal rule

$$\frac{\varphi \rightarrow \pi \in \Gamma \qquad len(\pi) \leq len_{MAX} \qquad \langle \delta, \gamma \rangle \models_{BG} \varphi}{\langle \delta, \gamma, \emptyset \rangle \quad \longrightarrow_{apply(\varphi \rightarrow \pi)} \quad \langle \delta, \gamma, \pi \rangle}$$

Application of a plan revision rule

$$\frac{\pi_1 \mid \beta \rightarrow \pi_2 \in \Delta \qquad len(\pi_2; \pi) \leq len_{MAX} \qquad \langle \delta, \gamma \rangle \models_{BG} \beta}{\langle \delta, \gamma, \pi_1; \pi \rangle \quad \longrightarrow_{apply(\pi_1 \mid \beta \rightarrow \pi_2)} \quad \langle \delta, \gamma, \pi_2; \pi \rangle}$$

Basic action execution

$$\frac{\mathcal{T}(b, \delta) = \delta' \qquad \gamma' = \gamma \setminus \{g \in \gamma \mid \delta' \models_{Prop} g\}}{\langle \delta, \gamma, b; \pi \rangle \quad \longrightarrow_{execute(b)} \quad \langle \delta', \gamma', \pi \rangle}$$

where $\mathcal{T}$ is a belief update function which takes an action and a belief base and returns the resulting belief base. $\mathcal{T}$ is a partial function since an action may not be applicable in some situations.

Conditional statement

$$\frac{\langle \delta, \gamma \rangle \models_{BG} \beta}{\langle \delta, \gamma, \textit{if } \beta \textit{ then } \pi_1 \textit{ else } \pi_2; \pi \rangle \rightarrow_{execute(if)} \langle \delta, \gamma, \pi_1; \pi \rangle}$$

$$\frac{\langle \delta, \gamma \rangle \not\models_{BG} \beta}{\langle \delta, \gamma, \textit{if } \beta \textit{ then } \pi_1 \textit{ else } \pi_2; \pi \rangle \rightarrow_{execute(if)} \langle \delta, \gamma, \pi_2; \pi \rangle}$$

Note that in the last three rules, π may be absent (or be an empty string), in which case for example executing $b; \pi$ will result in $\langle \delta', \gamma', \emptyset \rangle$.

For technical reasons, if in a configuration $\langle \delta, \gamma, \pi \rangle$ no transition rule is applicable, we assume that there is a special 'stall' transition to the same configuration: $\langle \delta, \gamma, \pi \rangle \rightarrow_{stall} \langle \delta, \gamma, \pi \rangle$.

A computation tree $CT(c_0, P)$ for a Dribble agent with a program $P = (\Gamma, \Delta)$ is a tree with root c_0 where each node is a configuration, such that for each node c and each child c' of c, $c \rightarrow c'$ is a transition in the transition system for P. The meaning of a Dribble agent $\langle \delta_0, \gamma_0, P \rangle$ is a tree $CT(\langle \delta_0, \gamma_0, \emptyset \rangle, P)$.

3 A Logic of Dribble Programs

In this section, we introduce a logic which allows us to formalize the properties of Dribble agent programs. Formulas of the logic will be used as input to the model-checker. In addition, we give a complete axiomatisation of the models of the logic. Axiomatisation is, of course not necessary for model-checking, but it helps us to understand the logic and its models; for example, the axioms may be more intuitive or clearer than the semantic conditions on models.

The language of our logic $\mathcal{L}_D$ is based on Computation Tree Logic (CTL) [7] which is a logic for reasoning about branching time. The syntax of CTL is as follows:

$$\mathcal{L}_{CTL}: \quad \varphi \leftarrow p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid EX\varphi \mid E(\varphi U \psi) \mid A(\varphi U \psi) \quad \text{where } p \in \mathcal{L}$$

The meaning of the temporal operators is as follows: $EX\varphi$ means there is a successor state which satisfies φ; $E(\varphi U \psi)$ means that there is a branch where φ holds until ψ becomes true; $A(\varphi U \psi)$ means that on all branches, φ holds until ψ.

3.1 Syntax

$\mathcal{L}_D$ extends L_{CTL} with belief, goal and plan operators ($\mathbf{Bs}$, $\mathbf{Gs}$ and $\mathbf{P}$).

$$\mathcal{L}_D: \quad \varphi \leftarrow \mathbf{Bs}\delta \mid \mathbf{Gs}\gamma \mid \mathbf{P}\pi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid EX\varphi \mid E(\varphi U \psi) \mid A(\varphi U \psi)$$

where $\delta, \gamma \subseteq \mathcal{L}$; $\pi \in Plans \cup \{E\}$. $\mathbf{Bs}$ and $\mathbf{Gs}$ describe the belief base and goal base of the agent. Note that these operators apply to *sets* of formulas. $\mathbf{P}$ is used to describe the plan base of the agent. If the agent's plan is π, this is expressed as $\mathbf{P}\pi$, and if the plan base is empty, this is expressed as $\mathbf{P}E$.

We will use the usual abbreviation:

$AX\varphi = \neg EX\neg\varphi$ (in all successor states, φ)
$AF\varphi = A(\top U \varphi)$ (on all branches, in some future state, φ)
$EF\varphi = E(\top U \varphi)$ (there exists a branch, where in some future state, φ)
$AG\varphi = \neg EF\neg\varphi$ (on all branches, in all states, φ)
$EG\varphi = \neg AF\neg\varphi$ (there is a branch, where in all states, φ).

3.2 Semantics

In this section we define models for the logic. We show in section 4 that they correspond exactly to the computation trees for Dribble agents generated by the operational semantics.

Given a Dribble agent program $P = (\Gamma, \Delta)$, a Dribble model of P is a triple $M_P = (S, R, V)$ in which:

- S is a nonempty set of states
- $R \subseteq S \times S$ satisfies the properties below
- $V = (V_b, V_g, V_p)$ a collection of three functions, $V_b(s) : S \rightarrow 2^{\mathcal{L}}$, $V_g(s) : S \rightarrow 2^{\mathcal{L}}$ and $V_p(s) : S \rightarrow 2^{Plans}$ satisfying the following conditions, for all $s \in S$:
  1. $V_b(s)$ and $V_g(s)$ are finite subsets of propositional formulas
  2. $V_b(s)$ is consistent
  3. $\forall \alpha \in V_g(s) : V_b(s) \not\models_{Prop} \alpha$
  4. $\forall \alpha \in V_g(s) : \alpha$ is (propositionally) consistent
  5. $V_p(s)$ is a singleton or an empty set.

To simplify the definition, we use the following conventions: $\forall g \in \Gamma$ of the form $\varphi \rightarrow \pi$ then $guard(g) = \varphi$, $body(g) = \pi$, i.e. $guard(g)$ specifies the mental condition for which situation is a good idea to execute the plan, and $body(g)$ is the plan generated

after firing the goal rule. Also, $\forall r \in \Delta$ of the form $\pi_1 \mid \beta \rightarrow \pi_2$ then $head(r) = \pi_1$, $guard(r) = \beta$ and $body(r) = \pi_2$. If $V_p(s) = \{\pi\}$, we will write $V_p(s) = \pi$ for readability.

Further requirements for R are listed below.

EPG: For all $s \in S$, if $V_p(s) = \emptyset$ and there exists $g \in \Gamma$ such that $\langle V_b(s), V_g(s)\rangle \models_{BG} guard(g)$ then there is $s' \in S$ such that $(s, s') \in R$ with $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and $V_p(s') = body(g)$

APG: For all $(s, s') \in R$ such that $V_p(s) = \emptyset$, then $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and there is $g \in \Gamma$ such that $\langle V_b(s), V_g(s)\rangle \models_{BG} guard(g)$ and $V_p(s') = body(g)$

EBA: For all $s \in S$, if $V_p(s) = b; \pi$ then there is $s' \in S$ such that $(s, s') \in R$ with $V_b(s') = T(b, V_b(s))$, $V_g(s') = V_g(s) \setminus \{g \in V_g(s) | V_b(s') \models_{Prop} g\}$ and $V_p(s') = \pi$

EIF: For all $s \in S$, if $V_p(s) = \pi_{if}; \pi$, where $\pi_{if} = \textit{if } \beta \textit{ then } \pi_1 \textit{ else } \pi_2$, then there is $s' \in S$ such that $(s, s') \in R$ with $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and

$$V_p(s') = \begin{cases} \pi_1; \pi & \text{if } \langle V_b(s), V_g(s)\rangle \models_{BG} \beta \\ \pi_2; \pi & \text{otherwise} \end{cases}$$

EPR: For all $s \in S$, if $V_p(s) = \pi_1; \pi$ and there exists $r \in \Delta$ such that $head(r) = \pi_1$ and $\langle V_b(s), V_g(s)\rangle \models_{BG} guard(r)$ then there is $s' \in S$ such that $(s, s') \in R$ with $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$ and $V_p(s') = body(r); \pi$

ABAvPR: For all $(s, s') \in R$, such that $V_p(s) = b; \pi'; \pi$, where π' might be empty, then either of the following is true:

1. $V_b(s') = T(b, V_b(s))$, $V_g(s') = V_g(s) \setminus \{g \in V_g(s) \mid V_b(s') \models_{Prop} g\}$ and $V_p(s') = \pi'; \pi$
2. $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$, and there is $r \in \Delta$ such that $\langle V_b(s), V_g(s)\rangle \models_{BG} guard(r)$, $head(r) = b; \pi'$ and $V_p(s') = body(r); \pi$

AIFvPR For all $(s, s') \in R$ such that $V_p(s) = \pi_{if}; \pi'; \pi$ (π' might be empty), then either of the following is true:

1. $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$, and

$$V_p(s') = \begin{cases} \pi_1; \pi'; \pi & \text{if } \langle V_b(s), V_g(s)\rangle \models_{BG} \beta \\ \pi_2; \pi'; \pi & \text{otherwise} \end{cases}$$

2. $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$, and there is $r \in \Delta$ such that $\langle V_b(s), V_g(s)\rangle \models_{BG} guard(r)$, $head(r) = \pi_{\text{if}}; \pi'$ and $V_p(s') = body(r); \pi$

APR For all $(s, s') \in R$ such that $V_p(s) = a; \pi'; \pi$ (π' might be empty), then $V_b(s') = V_b(s)$, $V_g(s') = V_g(s)$, and there is $r \in \Delta$ such that $\langle V_b(s), V_g(s)\rangle \models_{BG} guard(r)$, $head(r) = a; \pi'$ and $V_p(s') = body(r); \pi$

For any state s such that there are no transitions $R(s, s')$ required by the conditions above, we stipulate $R(s, s)$. This is required to make the transition relation serial.

No other transitions apart from those required by the conditions above exist in the model.

Given a model M_P and a state s of M_P, the truth of a $\mathcal{L}_D$ formula is defined inductively as follows:

- $M_P, s \models \mathbf{Bs}\delta \Leftrightarrow V_b(s) = \delta$
- $M_P, s \models \mathbf{Gs}\gamma \Leftrightarrow V_g(s) = \gamma$
- $M_P, s \models \mathbf{P}\pi \Leftrightarrow V_p(s) = \pi$
- $M_P, s \models \mathbf{P}E \Leftrightarrow V_p(s) = \emptyset$
- $M_P, s \models \neg\varphi \Leftrightarrow M_P, s \not\models \varphi$
- $M_P, s \models \varphi_1 \wedge \varphi_2 \Leftrightarrow M_P, s \models \varphi_1$ and $M_P, s \models \varphi_2$
- $M_P, s \models EX\varphi \Leftrightarrow \exists s' : (s, s') \in R : M_P, s' \models \varphi$
- $M_P, s \models E(\varphi U \psi) \Leftrightarrow \exists$ path $(s_0, s_1, ..., s_n)$ such that: $s_0 = s; n \geq 0; (s_i, s_{i+1}) \in R\ \forall 0 \leq i < n$ and $M_P, s_n \models \varphi$, and for all $i < n$, $M_P, s_i \models \psi$
- $M_P, s \models A(\varphi U \psi) \Leftrightarrow \forall$ paths $(s_0, s_1, ...)$ such that: $s_0 = s$ and $\forall i \geq 0\ (s_i, s_{i+1}) \in R$, exists $n \geq 0$: $M_P, s_n \models \varphi$, and for all $i < n$, $M_P, s_i \models \psi$

Note that the formulas of CTL are evaluated in state s on a tree corresponding to an unravelling of M_P with the root s. Without loss of generality, we can assume that each model of P is a tree with the root which intuitively corresponds to the initial configuration of the agent.

3.3 Axiomatization

We will refer to the axiom system below as the Dribble logic of a program P, DL_P. To simplify the axioms, we use $guard(g)$, $body(g)$, $head(r)$, $guard(r)$ and $body(r)$ with the same meanings as in the model. Finally, we use π_{if} for *if* β *then* π_1 *else* π_2.

CL classical propositional logic
CTL axioms of CTL
A1a $\bigvee_{\delta \subseteq \mathcal{L}} \mathbf{Bs}\delta$
A1b $\mathbf{Bs}\delta \rightarrow \neg\mathbf{Bs}\delta', \forall\delta' \neq \delta$
An agent has only one belief base.
A2a $\bigvee_{\gamma \subseteq \mathcal{L}} \mathbf{Gs}\gamma$
A2b $\mathbf{Gs}\gamma \rightarrow \neg\mathbf{Gs}\gamma', \forall\gamma' \neq \gamma$
An agent has only one goal base.
A3a $\bigvee_{\pi \in Plans \cup \{E\}} \mathbf{P}\pi$
A3b $\mathbf{P}\pi \rightarrow \neg\mathbf{P}\pi'$, where $\pi, \pi' \in Plans \cup \{E\}, \forall\pi' \neq \pi$
An agent has only one plan.
A4 $\neg\mathbf{Bs}\delta, \forall\delta$ such that $\delta \models_{Prop} \bot$
Belief base is consistent.
A5 $\mathbf{Bs}\delta \rightarrow \neg\mathbf{Gs}\gamma$ for all γ such that $\exists g \in \gamma : \delta \models_{Prop} g$
All goals in goal base are not consequences of belief base.
A6 $\neg\mathbf{Gs}\gamma$ for all γ such that $\exists g \in \gamma : g \models_{Prop} \bot$
Each goal in goal base is consistent.

EPG $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}E \rightarrow EX(\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\pi)$ if $\exists g \in \Gamma$ such that $\langle \delta, \gamma \rangle \models_{BG} guard(g)$ and $\pi = body(g)$

In a state s where some planning goal rule is applicable, i.e. the current plan is empty, there exists a next state s' where its plan is the one generated by firing the planning goal rule.

APG $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}E \rightarrow AX(\bigvee_{g \in \Gamma'} (\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\pi_g))$ where Γ' is a set of planning goal rules g that satisfies the following two conditions: $\langle \delta, \gamma \rangle \models_{BG} guard(g)$ and $\pi_g = body(g)$, provided $\Gamma' \neq \emptyset$.

In a state s where the current plan is empty, all possible next states from s are only reachable by applying some PG rule, i.e. its plan is generated by firing the PG rule.

EBA $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(b;\pi) \rightarrow EX(\mathbf{Bs}\delta' \wedge \mathbf{Gs}\gamma' \wedge \mathbf{P}\pi)$ where $\delta' = T(b, \delta)$ and $\gamma' = \gamma \setminus \{g \mid \delta' \models_{Prop} g\}$

In a state s where a basic action is applicable, there exists a next state s' in which the basic action is removed from its plan, and the belief base is updated according to the basic action (the goal base, therefore, also has to be changed in order to maintain the disjointness with the belief base).

EIF $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(\pi_{if};\pi) \rightarrow EX(\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(\pi_i;\pi))$
where

$$\pi_i = \begin{cases} \pi_1 & \text{if } \langle V_b(s), V_g(s) \rangle \models_{BG} \beta \\ \pi_2 & \text{otherwise} \end{cases}$$

In a state s where the current plan begins with a conditional plan, there exists a next state s' in which the conditional plan is replaced by one of its two sub plans depending on whether its condition is derivable or not from the belief base in s, respectively.

EPR $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(\pi_1;\pi) \rightarrow EX(\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(\pi_2;\pi))$
if $\exists r \in \Delta$ such that $\langle \delta, \gamma \rangle \models_{BG} guard(r)$, $head(r) = \pi_1$ and $\pi_2 = body(r)$

In a state s where a plan revision rule is applicable, i.e. the head of the rule is the beginning of the current state and the guard of the rule is derivable from the current belief and goal base, there exists a next state s' in which the beginning of the plan in s is replaced by the body of the rule.

ABAvPR $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(b;\pi';\pi) \rightarrow AX((\mathbf{Bs}\delta' \wedge \mathbf{Gs}\gamma' \wedge \mathbf{P}(\pi';\pi)) \vee \bigvee_{r \in \Delta'} (\mathbf{Bs}\delta' \wedge \mathbf{Gs}\gamma' \wedge \mathbf{P}(\pi'';\pi)))$ where Δ' is a set of plan revision rules r that satisfies the following three conditions: $head(r) = b;\pi'$, $\langle \delta, \gamma \rangle \models_{BG} guard(r)$ and $body(r) = \pi''$, provided $\Delta' \neq \emptyset$ or $\mathcal{T}(\lfloor, \delta)$ is defined.

In a state s where a basic action b is the first element of the plan, we can only transit to another state by executing the action or applying an applicable practical reasoning rule.

AIFvPR $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(\pi_{if};\pi';\pi) \rightarrow AX((\mathbf{Bs}\delta' \wedge \mathbf{Gs}\gamma' \wedge \mathbf{P}(\pi_i;\pi';\pi)) \vee \bigvee_{r \in \Delta'} (\mathbf{Bs}\delta' \wedge \mathbf{Gs}\gamma' \wedge \mathbf{P}(\pi'';\pi)))$ where Δ' is a set of plan revision rules r that

satisfy three following conditions: $head(r) = \pi_{if}; \pi'$, $\langle \delta, \gamma \rangle \models_{BG} guard(r)$ and $body(r) = \pi''$; and

$$\pi_i = \begin{cases} \pi_1 & \text{if } \langle V_b(s), V_g(s) \rangle \models_{BG} \beta \\ \pi_2 & \text{otherwise} \end{cases}$$

In a state s where an if-then-else statement is the first element of the plan, we can only transit to another state by executing the if-then-else statement or applying an applicable practical reasoning rule.

APR $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}(a; \pi_1; \pi) \rightarrow AX \bigvee_{r \in \Delta'} (\mathbf{Bs}\delta' \wedge \mathbf{Gs}\gamma' \wedge \mathbf{P}(\pi_2; \pi))$ where Δ' is a set of plan revision rules r that satisfy three following conditions: $head(r) = a; \pi_1$, $\langle \delta, \gamma \rangle \models_{BG} guard(r)$ and $body(r) = \pi_2$; provided $\Delta' \neq \emptyset$.

In a state s where an abstract plan is the first element of the plan, we can only transit to another state by applying a practical reasoning rule.

Stall $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\pi \rightarrow AX(\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\pi)$ where $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\pi$ describes a configuration from which no normal transitions are available.

We have the following result.

Theorem 1. *DL_P is sound and complete with respect to the class of models of the program P.*

Proof. The proof of soundness is straightforward and is omitted. In the rest of this section, we show the completeness of DL_P. Most of the proof is from that of CTL [13].

Let $BGP = 2^{\mathcal{L}} \times 2^{\mathcal{L}} \times (Plans \cup \{E\})$. BGP intuitively corresponds to the set of all possible configurations. Note that this is a finite set.

Given a consistent formula φ_0, we construct the generalised Fischer-Ladner closure of φ_0, $FL(\varphi_0)$, as the least set H of formulas containing φ_0 such that:

1. $\mathbf{Bs}\delta \in H$ for all $\delta \subseteq \mathcal{L}$
2. $\mathbf{Gs}\gamma \in H$ for all $\gamma \subseteq \mathcal{L}$
3. $\mathbf{P}\pi \in H$ for all $\pi \in Plans \cup \{E\}$
4. $EX(\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\psi)$ for all $(\delta, \gamma, \pi) \in BGP$
5. $EX(\bigvee_{(\delta,\gamma,\pi) \in BGP'} (\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}\psi))$ for all $BGP' \subseteq BGP$
6. $\neg\varphi \in H$, then $\varphi \in H$
7. $\varphi \wedge \psi \in H$, then $\varphi, \psi \in H$
8. $E(\varphi U \psi) \in H$, then $\varphi, EXE(\varphi U \psi) \in H$
9. $A(\varphi U \psi) \in H$, then $\varphi, AXA(\varphi U \psi) \in H$
10. $EX\varphi \in H$, then $\varphi \in H$
11. $AX\varphi \in H$, then $\varphi \in H$
12. $\varphi \in H$ and φ is not of the form $\neg\psi$, then $\neg\varphi \in H$

It is obvious that $FL(\varphi_0)$ is finite. As usual, we define a subset s of $FL(\varphi_0)$ that is maximally consistent if s is consistent and for all $\varphi, \neg\varphi \in FL(\varphi_0)$, either φ or $\neg\varphi$ is in s. Repeat the construction of a model M for φ_0 as in [13] based on the set of maximally consistent sets of $FL(\varphi_0)$, with the condition that the assignments are as follows:

- $V_b(s) = \delta$ for any δ such that $\mathbf{Bs}\delta \in s$
- $V_g(s) = \gamma$ for any γ such that $\mathbf{Gs}\gamma \in s$
- $V_p(s) = \pi$ for any π such that $\mathbf{P}\pi \in s$ (and $V_p(s) = \emptyset$ if $\mathbf{P}E \in s$).

The above definition is well-defined because axioms **A1x**, **A2x** and **A3x** guarantee that there are exactly one $\mathbf{Bs}\delta \in s$, $\mathbf{Gs}\gamma \in s$ and $\mathbf{P}\pi \in s$. Our remaining task is to show that M is, in fact, a model of P.

EPG: Assume that $V_b(s) = \delta$, $V_g(s) = \gamma$ and $V_p(s) = \emptyset$, then we have $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}E \in s$. Furthermore, assume that there is $g \in \Gamma$ such that $\langle \delta, \gamma \rangle \models_{BG} guard(g)$. By axiom **EPG** and modus ponens (MP), $EX(\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}body(g)) \in s$. According to the construction of M, there is s' such that $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}body(g) \in s'$ and $(s, s') \in R$. It is obvious that $V_b(s') = \delta$, $V_g(s') = \gamma$ and $V_p(s') = body(g)$.

APG: Assume that $V_b(s) = \delta$, $V_g(s) = \gamma$ and $V_p(s) = \emptyset$, then we have $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}E \in s$. Let Γ' is the set of PG rules $g \in \Gamma$ such that $\langle \delta, \gamma \rangle \models_{BG} guard(g)$. By axiom **APG** and modus ponens (MP), $AX(\bigvee_{g \in \Gamma'} (\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}body(g))) \in s$. According to the construction of M, for any s' such that $(s, s') \in R$,

$$\bigvee_{g \in \Gamma'} (\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}body(g)) \in s$$

Then, there exists $g \in \Gamma'$ such that $\mathbf{Bs}\delta \wedge \mathbf{Gs}\gamma \wedge \mathbf{P}body(g) \in s'$. It is obvious that $V_b(s') = \delta$, $V_g(s') = \gamma$ and $V_p(s') = body(g)$.

The proof of other conditions on R is similar to the two cases above and are omitted. This shows that M is a model of the program P.

4 Verification

We can express properties of Dribble programs using CTL operators in the usual way. For example, a safety property that 'nothing bad will happen' can be expressed as $AG\neg\phi$, where ϕ is a description of the 'bad' situation. Similarly, a liveness property that 'something good will happen', for example the agent will achieve its goal, can be expressed as $EF\phi$, where ϕ is a description of the 'good' situation. It is essential however that we know that we are verifying the properties with respect to the computation trees which precisely correspond to the operational semantics of the agent. We prove in the next section that models of the logic correspond to computation trees, and hence that a CTL formula is true at the root of a Dribble model for P if, and only if, the corresponding property holds for the initial configuration c_0 of its computation tree $CT(c_0, P)$.

4.1 Correspondence Theorem

We say that a state s of some model M_P is corresponding to a configuration $c \in CT(c_0, P)$ (notation: $s \sim c$) iff $c = \langle V_b(s), V_g(s), V_p(s) \rangle$.

Consider a Dribble agent $\langle \delta_0, \gamma_0, P\rangle$ and its computational tree $CT(\langle \delta_0, \gamma_0, \emptyset\rangle, P)$. We claim that it is isomorphic to the Dribble model of P with the root s_0 such that $s_0 \sim \langle \delta_0, \gamma_0, \emptyset\rangle$ and for every state s, all children of s are distinct. The last condition is required for isomorphism; there may be Dribble models for P where there are duplicated transitions to identical states. Such duplication of identical successors does not affect the truth of DL_P formulas.

Theorem 2. *$CT(c_0, P)$ is isomorphic to the Dribble model M_P of P with the root s_0 such that $s_0 \sim c_0$ satisfying the condition that for every state s, all children of s are distinct.*

Proof. We are going to show that $\sim$ defines a bijection between the states of $CT(c_0, P)$ and M_P. We prove the theorem by induction on the distance from the root of the tree.

Base case: Assume that $(s_0, s) \in R$ in M_P. We will show that there exists a unique c_0 with $c_0 \rightarrow c$ in $CT(c_0, P)$ and $s \sim c$. The other direction (if $c_0 \rightarrow c$ then there is a unique s such that $R(s_0, s)$ and $s \sim c$) is similar.

Case 1: Assume that $V_p(s_0) = \emptyset$, by **APG**, there is $g \in \Gamma$ such that

$$\langle V_b(s_0), V_g(s)\rangle \models_{BG} guard(g)$$

Furthermore, $V_b(s) = V_b(s_0)$, $V_g(s) = V_g(s_0)$ and $V_p(s) = body(g)$. Let $c = \langle V_b(s), V_g(s), V_p(s)\rangle$. By the operational semantics, $c_0 \rightarrow_{apply(g)} c$.

Case 2: Assume that $V_p(s_0) = b; \pi'; \pi$ (π' might be empty). As $(s_0, s) \in R$, **ABAvPR** implies that there are two cases to consider. In the first case, $V_b(s) = T(b, V_b(s_0))$, $V_g(s) = V_g(s_0) \setminus \{\alpha \in V_g(s_0) \mid V_b(s) \models_{Prop} \alpha\}$ and $V_p(s) = \pi'; \pi$. Simply let $c = \langle V_b(s), V_g(s), \pi'\pi\rangle$, we have that $c_0 \rightarrow_{execute(b)} c$. In the second case, we have $V_b(s) = V_b(s_0)$, $V_g(s) = V_g(s_0)$ and there is $r \in \Delta$ such that $head(r) = b; \pi'$ and

$$\langle V_b(s_0), V_g(s_0)\rangle \models_{BG} guard(r)$$

and $V_p(s) = body(r); \pi$. Let $c = \langle V_b(s), V_g(s), body(r); \pi\rangle$, then we have $c_0 \rightarrow_{apply(r)} c$.

For the other cases of $V_p(s_0)$, the proof is done in a similar manner by using the suitable conditions of R.

Induction step: Assume that the path from s_0 to s has length $n > 1$. That means there are $s_1, \ldots, s_n = s$ in M_P such that $(s_i, s_{i+1}) \in R$ for all $i > 0$. By the induction hypothesis, there are $c_1, \ldots, c_{n-1}$ such that $s_i \sim c_i$ for all $i = 0, \ldots, n-1$ and $c_i \rightarrow_{x_i} c_{i+1}$ by some

$$x_i \in \{execute(b), execute(if), apply(g), apply(r)\}$$

By repeating the proof of the base case, we have that there is c_n such that $s_n \sim c_n$ and $c_{n-1} \rightarrow_x c_n$ by some

$$x \in \{execute(b), execute(if), apply(g), apply(r)\}.$$

4.2 Automated Verification

In this section, we show how to encode DL_P models for a standard CTL model checker to allow the automated verification of properties of Dribble programs. For the examples reported here, we have used the MOCHA model checker [3], due to the ease with which we can specify Dribble programs in *reactive modules*, the description language used by MOCHA.[1]

States of the DL_P models correspond to an assignment of values to state variables in the model checker. In particular, the agent's mental state is encoded as a collection of state variables. The agent's goals and beliefs are encoded as boolean variables with the appropriate initial values. The agent's plan is encoded as an array of *steps* of length $len_{MAX} + 1$. *step* is an enumeration type which includes all basic actions and abstract plans declared in the agent's program, and a set of special *if-tokens*. Each if-token (β, u, v) corresponds to an if-then-else construct appearing one of the agent's plans, and encodes the belief(s) tested, β and the lengths of the 'then' and 'else' branches (denoted by u and v respectively). All elements of the plan array are initially assigned the value *null*.

The execution of plans and the application of goal and practical reasoning rules are encoded as a MOCHA *atom* which describes the initial condition and transition relation for the variables corresponding to the agent's belief and plan bases. A basic action is performed if the corresponding step token is the first element in the plan array. Executing the action updates the agent's beliefs appropriately and advances the plan, i.e., for each plan element $i > 0$, the step at location i is moved to location $i - 1$. If the first element of the plan array is an if-token, a test is performed on the appropriate belief(s) and the plan advanced accordingly. For example, if the first element of the plan array is (β, u, v) and β is false, the plan is advanced $u+1$ steps. Goal rules can be applied when the agent has no plan, i.e., when the first element of the plan array contains 'null', and the rule's mental condition holds (i.e., the agent has the appropriate beliefs and goals). Firing the rule writes the plan which forms the body of the goal rule into the plan array. Practical reasoning rules can be applied when the plan to be revised matches a prefix of the plan array and the rule's belief condition is believed by the agent. Firing the rule writes the plan which forms the body of the rule into the plan array and appends the suffix of the original plan (if any). In the case in which the first element of the plan array is an abstract action, application of the appropriate practical reasoning rule inserts the plan corresponding to the abstract action at the beginning of the plan array. An additional atom encodes the agent's commitment strategy, and drops any goals the agent has come to believe as a result of executing a basic action.

The evolution of the system's state is described by an initial round followed by an infinite sequence of update rounds. State variables are initialised to their initial values in the initial round and new values are assigned to the variables in the subsequent update rounds. At each update round, MOCHA non-deterministically chooses between executing the next step in the plan (if any) and firing any applicable goal and practical reasoning rules.

[1] Note that model checkers such as MCMAS [15] intended for the verification of multi-agent systems are not appropriate, as they assume that epistemic modalities are defined in terms of accessibility relations, rather than syntactically as in $\mathcal{L}_D$.

4.3 Example

As an illustration, we show how to prove properties of a simple agent program written in Dribble. In the interests of brevity, we do not consider abstract plans in this example. Consider the following Dribble program for a simple 'vacuum cleaner' agent. The agent's environment consists of two rooms, room1 and room2, and its goal is to clean both rooms. The agent has actions which allow it to move between rooms and to clean a room, and goal rules which allow it to select an appropriate plan to clean a room. To clean a room the agent's battery must be charged and cleaning discharges the battery. The agent has two practical reasoning rules which revise a plan which is not executable because the battery has become discharged or because charging the battery is only possible in room2. The agent's beliefs and goals are expressed using the following propositional variables: c_1, c_2 which mean that room1 and room2 are clean, r_1, r_2 which mean that the agent is in room1 and room2, and b means that the agent's battery is charged. The agent has the following five basic actions:

- mR for 'move right'. It is applicable if the agent's belief base δ is such that $\delta \models_{prop} r_1$, for example $\delta = \{b \wedge \neg c_1 \wedge \neg c_2 \wedge r_1 \wedge \neg r_2\}$. For this particular δ, $\mathcal{T}(mR, \delta) = \{b \wedge \neg c_1 \wedge \neg c_2 \wedge \neg r_1 \wedge r_2\}$, that is, the agent no longer believes that it is in r_1 but believes that it is in r_2. In general, $\mathcal{T}(mR, \delta)$ is defined as follows: for every $\alpha \in \delta$, in every disjunct in α, add to the conjunction $\neg r_1 \wedge r_2$ and remove r_1 and $\neg r_2$ (if they were in the conjunction).[2]
- mL for 'move left'. The belief update function is defined analogously.
- cR for 'clean room'. It is applicable if the agent's belief base δ is such that $\delta \models_{prop} b$. If the action is executed in room1, it makes c_1 true, similarly for executing cR in room2. In both cases b becomes false.
- cB for 'charge battery'. It is only applicable in r_2 (intuitively because this is where the charger is) and it makes b true.

The agent's program consists of the following two PG rules:

$$pg_1 = Gc_1 \;\rightarrow\; \texttt{if}\ Br_1\ \texttt{then}\ cR\ \texttt{else}\ mL; cR$$
$$pg_2 = Gc_2 \;\rightarrow\; \texttt{if}\ Br_2\ \texttt{then}\ cR\ \texttt{else}\ mR; cR$$

and two PR rules:

$$pr_1 = cR; \pi \mid \neg Bb \rightarrow cB; cR; \pi$$
$$pr_2 = cB; \pi \mid Br_1 \rightarrow mR; cB; mL; \pi$$

[2] The intuition underlying the belief update function $\mathcal{T}$ is as follows. Think of every $\alpha \in \delta$ as a description of a set of states the agent may believe that it is in. Each disjunct is a (possibly partial) description of a state. For example, if the agent believes that its battery is not charged, that it is in room2 and that one of the rooms is clean, then its belief base may be $\delta_1 = \{(\neg b \wedge c_1 \wedge r_2) \vee (\neg b \wedge c_2 \wedge r_2)\}$ or $\delta_2 = \{\neg b \wedge r_2,\ c_1 \vee c_2\}$ (other belief bases representing the same information are also possible). After an update, for every partial description, we should add to the description the effect of the action and remove any beliefs inconsistent with this effect. For example, the effect of executing cB is that the battery is charged. After executing cB the agent should believe that it battery is charged, so we add b to each disjunct (and remove $\neg b$ if it is there). δ_1 would therefore become $\{(b \wedge c_1 \wedge r_2) \vee (b \wedge c_2 \wedge r_2)\}$ and δ_2 would become $\{b \wedge r_2, (b \wedge c_1) \vee (b \wedge c_2)\}$.

In this example, the agent has two PR rules to revise a plan in case it is not executable. In the first case, if the agent intends to clean a room but its battery is not charged, pr_1 will add an additional action cB before cR. However, charging the battery (cB) is applicable only if the agent is in room2. If the agent is in room1, pr_2 will revise a plan containing a charge battery action by inserting a sequence of actions to move to room2, then charge the battery, and then move back to room1.

The program of the vacuum cleaner agent realises the following simple strategy: if the agent has a goal to clean some room, it moves to that room to clean it. If the agent's battery is discharged and it cannot clean the room, it postpones cleaning the room and goes to charge the battery. Similarly, if agent wants to charge its battery and it is not in room2, it moves to room2 before attempting to charge the battery before returning to room1. While this strategy could be implemented without using PR rules, the resulting set of PG rules would be more complex.

We would like to verify that, starting in a state in which the agent believes that it is in room1, r_1, and its battery is charged, b, the above program results in the agent achieving its goals c_1 and c_2. This can be achieved by verifying, for the corresponding model, that $AF(Bc_1 \wedge Bc_2)$ is true in the initial configuration where b, $\neg c_1$, $\neg c_2$, r_1 and $\neg r_2$ are true.

The MOCHA encoding for the vacuum cleaner example is given in appendix A.

5 Related Work

A logic for proving properties of Dribble agents is presented in [18], based on dynamic logic rather than on CTL. However, no axiomatisation of the logic or automated verification procedure is given. In [1,2], Alechina et al. introduced a logic which is also based on dynamic logic for verification of agent programs written in a sublanguage of 3APL, but this work does not consider rules to revise plans. In [10] Dastani et al. prove a correspondence between an APL-like agent programming language and a specification language based on CTL, and show that any agent implemented in the language satisfies some desirable properties, e.g., relating to commitment strategies. In contrast, our aim in this paper is to verify whether a given property holds for a particular agent program. In addition, the APL considered by Dastani et al. does not include practical reasoning rules, and hence their results are confined to agents which are unable to revise their plans.

A strand of work on model-checking properties of agent programming languages is represented by [4] and continued by [11,12] who use Java Path Finder (JPF) which is a model checker for Java programs. This approach requires writing a Java interpreter for the language using Agent Infrastructure Layer (AIL). After that, verification proceeds as for a standard Java program, without exploiting any features specific to agent programming languages.

A model-checking approach to automated verification of ConGolog programs was described in [8]. The paper proposes a very expressive logic which includes first-order language for specifying properties of programs and defines a model-checking algorithm for the logic. Due to the first-order sublanguage, the algorithm is not guaranteed to terminate. While ConGolog is a very expressive language, it differs from the APL and Dribble family of languages in that it lacks an explicit mechanism for revising plans.

MetateM [14] is another language for agent programming which is based on executable temporal statements. Although it is easy to verify automatically properties of agents written in MetateM, the language is very different from agent programming languages such as 3APL, AgentSpeak and Dribble, where plan constructs are based on conventional imperative languages (e.g. plans with branching constructs and loops).

6 Conclusion

This paper describes a temporal logic which allows us to axiomatise the set of transition systems generated by the operational semantics of a Dribble program, and formulate properties of a Dribble agent, such as that the agent is guaranteed to achieve its goals or is not going to violate some safety restrictions. One of the interesting properties of Dribble are practical reasoning or plan rewriting rules, and we believe that they pose interesting challenges for logical formalisation; in particular, we had to introduce explicit 'plan operators' in the language to model those rules. We show how to encode the models of the logic as input to a standard model-checker, which gives an automatic procedure for verifying properties of a Dribble program.

References

1. Alechina, N., Dastani, M., Logan, B., Meyer, J.-J.C.: A logic of agent programs. In: Proc. of the 22nd National Conf. on Artificial Intelligence (AAAI 2007), pp. 795–800. AAAI Press, Menlo Park (2007)
2. Alechina, N., Dastani, M., Logan, B., Meyer, J.-J.C.: Reasoning about agent deliberation. In: Proc. of the 11th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2008), pp. 16–26. AAAI, Menlo Park (2008)
3. Alur, R., Henzinger, T.A., Mang, F.Y.C., Qadeer, S., Rajamani, S.K., Tasiran, S.: MOCHA: Modularity in model checking. In: Computer Aided Verification, pp. 521–525 (1998)
4. Bordini, R., Fisher, M., Visser, W., Wooldridge, M.: State-space reduction techniques in agent verification. In: Proceedings of the 3rd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 2004), pp. 896–903. ACM Press, New York (2004)
5. Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E. (eds.): Multi-Agent Programming: Languages, Platforms and Applications. Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 15. Springer, Heidelberg (2005)
6. Bordini, R.H., Hübner, J.F., Vieira, R.: Jason and the Golden Fleece of agent-oriented programming. In: Multi-Agent Programming: Languages, Platforms and Applications, ch. 1. Springer, Heidelberg (2005)
7. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
8. Claßen, J., Lakemeyer, G.: A logic for non-terminating Golog programs. In: Proceedings of the 11th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2008), pp. 589–599. AAAI, Menlo Park (2008)
9. Dastani, M., van Riemsdijk, M.B., Dignum, F., Meyer, J.-J.C.: A programming language for cognitive agents goal directed 3apl. In: Dastani, M.M., Dix, J., El Fallah-Seghrouchni, A. (eds.) PROMAS 2003. LNCS (LNAI), vol. 3067, pp. 111–130. Springer, Heidelberg (2004)

10. Dastani, M., van Riemsdijk, M.B., Meyer, J.-J.C.: A grounded specification language for agent programs. In: Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), pp. 578–585. ACM Press, New York (2007)
11. Dennis, L.A., Farwer, B., Bordini, R.H., Fisher, M.: A flexible framework for verifying agent programs. In: 7th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), IFAAMAS, pp. 1303–1306 (2008)
12. Dennis, L.A., Fisher, M.: Programming verifiable heterogeneous agent systems. In: 6th Int. Workshop on Programming in Multi-Agent Systems (ProMAS 2008), pp. 27–42 (2008)
13. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. J. Comput. Syst. Sci. 30(1), 1–24 (1985)
14. Fisher, M.: Metatem: The story so far. In: Bordini, R.H., Dastani, M.M., Dix, J., El Fallah Seghrouchni, A. (eds.) PROMAS 2005. LNCS (LNAI), vol. 3862, pp. 3–22. Springer, Heidelberg (2006)
15. Lomuscio, A., Raimondi, F.: MCMAS: A model checker for multi-agent systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 450–454. Springer, Heidelberg (2006)
16. Morley, D., Myers, K.: The SPARK agent framework. In: Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2004), pp. 714–721. IEEE Computer Society, Los Alamitos (2004)
17. Thangarajah, J., Harland, J., Morley, D., Yorke-Smith, N.: Aborting tasks in bdi agents. In: Proc. of the 6th Int. Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS 2007), pp. 8–15 (2007)
18. van Riemsdijk, B., van der Hoek, W., Meyer, J.-J.C.: Agent programming in dribble: from beliefs to goals using plans. In: Proc. of the 2nd Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2003), pp. 393–400. ACM Press, New York (2003)

A MOCHA Encoding for the Vacuum Cleaner Agent Example

```
module vacuum
  interface G_c1, G_c2, B_c1, B_c2, B_r1, B_r2, B_b : bool
  interface Pi : array (0..20) of
     { null, mL, mR, cR, cB, ifBr1_then1_else2, ifBr2_then1_else2 }
  interface Pilen : (0..100)

  atom actions
    controls B_c1, B_c2, B_r1, B_r2, B_b, Pi, Pilen
    reads    B_c1, B_c2, B_r1, B_r2, B_b, Pi, Pilen, G_c1, G_c2

    init
      [] true -> B_c1' := false; B_c2' := false;
                 B_r1' := true; B_r2' := false;
                 B_b' := true;
                 Pilen' := 0;
                 forall i Pi'[i] := null

    update
    -- PG rules
      [] G_c1 & Pilen = 0 ->
                 Pi'[0] := ifBr1_then1_else2;
                 Pi'[1] := cR; Pi'[2] := mL;Pi'[3] := cR;
                 Pilen' := 4
      [] G_c2 & Pilen = 0 ->
                 Pi'[0] := ifBr2_then1_else2;
                 Pi'[1] := cR; Pi'[2] := mR; Pi'[3] := cR;
                 Pilen' := 4
    -- PR rules
```

```
      [] Pilen > 0 & Pi[0] = cR & ~B_b ->
                forall i Pi'[i]:= if (i>0) then Pi[i-1] else Pi[i] fi;
                Pi'[0] := cB;
                Pilen' := Pilen + 1
      [] Pilen > 0 & Pi[0] = cB & B_r1 ->
                forall i Pi'[i]:= if (i>1) then Pi[i-2] else Pi[i] fi;
                Pi'[0] := mR; Pi'[1] := cB; Pi'[2] := mL;
                Pilen' := Pilen + 2
    -- If then else statement
      [] Pilen > 0 & Pi[0]=ifBr1_then1_else2 & B_r1 ->
                forall i Pi'[i]:= if (i<1) then Pi[i+1] else
                                     if (i<18) then Pi[i+3] else Pi[i] fi
                                  fi;
                Pilen' := Pilen - 3
      [] Pilen > 0 & Pi[0]=ifBr1_then1_else2 & ~B_r1 ->
                forall i Pi'[i]:= if (i<19) then Pi[i+2] else Pi[i] fi;
                Pilen' := Pilen - 2
      [] Pilen > 0 & Pi[0]=ifBr2_then1_else2 & B_r2 ->
                forall i Pi'[i]:= if (i<1) then Pi[i+1] else
                                     if (i<18) then Pi[i+3] else Pi[i] fi
                                  fi;
                Pilen' := Pilen - 3
      [] Pilen > 0 & Pi[0]=ifBr2_then1_else2 & ~B_r2 ->
                forall i Pi'[i]:= if (i<19) then Pi[i+2] else Pi[i] fi;
                Pilen' := Pilen - 2
    -- Belief update actions
      [] Pilen > 0 & Pi[0]=mR -> B_r1' := false; B_r2' := true;
                forall i Pi'[i]:= if (i<20) then Pi[i+1] else Pi[i] fi;
                Pilen' := Pilen - 1
      [] Pilen > 0 & Pi[0]=mL -> B_r1' := true;  B_r2' := false;
                forall i Pi'[i]:= if (i<20) then Pi[i+1] else Pi[i] fi;
                Pilen' := Pilen - 1
      [] Pilen > 0 & Pi[0]=cR & B_r1 & B_b -> B_c1' := true; B_b' := false;
                forall i Pi'[i]:= if (i<20) then Pi[i+1] else Pi[i] fi;
                Pilen' := Pilen - 1
      [] Pilen > 0 & Pi[0]=cR & B_r2 & B_b -> B_c2' := true; B_b' := false;
                forall i Pi'[i]:= if (i<20) then Pi[i+1] else Pi[i] fi;
                Pilen' := Pilen - 1
      [] Pilen > 0 & Pi[0]=cB & B_r2 ->
                B_b' := true;
                forall i Pi'[i]:= if (i<20) then Pi[i+1] else Pi[i] fi;
                Pilen' := Pilen - 1
  endatom

  atom goals
    controls G_c1, G_c2
    reads  G_c1, G_c2
    awaits B_c1, B_c2

    init
      [] true -> G_c1' := true; G_c2' := true

    update
      [] B_c1' & G_c1 -> G_c1' := false
      [] B_c2' & G_c2 -> G_c2' := false
  endatom

endmodule
```

Author Index