Matteo Baldoni
Tran Cao Son
M. Birna van Riemsdijk
Michael Winikoff (Eds.)

Declarative Agent Languages and Technologies VI

LNAI 5397

6th International Workshop, DALT 2008
Estoril, Portugal, May 2008
Revised Selected and Invited Papers

Lecture Notes in Artificial Intelligence 5397

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

Subseries of Lecture Notes in Computer Science

Matteo Baldoni
Tran Cao Son
M. Birna van Riemsdijk
Michael Winikoff (Eds.)

Declarative Agent Languages and Technologies VI

6th International Workshop, DALT 2008
Estoril, Portugal, May 12, 2008
Revised Selected and Invited Papers

Springer

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada
Jörg Siekmann, University of Saarland, Saarbrücken, Germany
Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editors

Matteo Baldoni
Università di Torino
Dipartimento di Informatica
Via Pessinetto 12, 10149 Turin, Italy
E-mail: baldoni@di.unito.it

Tran Cao Son
New Mexico State University
Department of Computer Science
P.O. Box 30001, MSC CS, Las Cruces, NM 88003, USA
E-mail: tson@cs.nmsu.edu

M. Birna van Riemsdijk
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
E-mail: m.b.vanriemsdijk@tudelft.nl

Michael Winikoff
University of Otago
Higher Education Development Centre
P.O. Box 56, Dunedin, New Zealand
E-mail: michael.winikoff@otago.ac.nz

Library of Congress Control Number: Applied for

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 3-540-93919-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-93919-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12604487 06/3180 5 4 3 2 1 0

Preface

The workshop on Declarative Agent Languages and Technologies (DALT), in its *sixth edition* this year, is a well-established forum for researchers interested in sharing their experiences in combining declarative and formal approaches with aspects of engineering and technology of agents and multiagent systems.

DALT 2008 was held as a satellite workshop of AAMAS 2008, the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, in Estoril, Portugal. 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), and DALT 2007 in Honolulu (LNAI 4897), the workshop again provided 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) to bring the issue of designing complex agent systems to the attention of researchers working on declarative languages and technologies.

The aim of the DALT workshop is to stimulate research on formal and declarative approaches both for developing the foundations of multiagent systems as well as for all phases of engineering multiagent systems, i.e., for specification and modeling, for implementation, and for verification. By providing a forum for the presentation of ideas addressing both of these aspects, DALT encourages the integration of formal and declarative techniques and methods that are based on solid theoretical foundations in the engineering of multiagent systems.

Recent advances in the area of computational logic provide a strong foundation for declarative languages and technologies, increasingly allowing agents to be endowed with mechanisms for behaving flexibly and autonomously in open and dynamic environments. In this setting, it becomes more and more important that multiagent systems are engineered to ensure both adaptability *and* a certain level of predictability. While providing a certain level of predictability is important for any software, it is especially important for multiagent systems in which the agents are autonomous and adaptive. Formal and declarative technologies both for specification and verification as well as for implementation are arguably the most promising approach to providing this required predictability. Ensuring a certain level of predictability is important for the adoption of multiagent technology in practice, as users have to trust a multiagent system to behave as required even though the agents are autonomous and adaptive.

An ongoing challenge for the DALT community is the investigation of formal and declarative techniques for the specification and implementation of rational agents. Moreover, techniques for structuring a multiagent system and for facilitating cooperation among agents such as organizational views of agent systems, norms, teams, coordination mechanisms, and argumentation and negotiation techniques are becoming increasingly important and are challenges for the

DALT community. Further, there are several areas that have commonalities with multiagent systems and to which declarative agent languages and technologies can be applied, such as the Semantic Web, service-oriented systems, component-based systems, security, and electronic contracting.

There is thus an ongoing and even increasing demand for formal and declarative approaches for the development of multiagent systems. In this volume, we report on the latest results in this area, including papers on logical foundations, declarative programming approaches, and verification of multiagent systems. The DALT workshop received 24 submissions. Each paper was reviewed by at least 3 reviewers, and 14 papers were accepted: 10 as full papers, and 4 as short papers. Out of these 14 contributed articles that were selected for presentation at the workshop, 12 papers were selected by the Program Committee for publication in this volume, as well as three invited articles, originally presented as short papers at AAMAS 2008, that have been extended by their authors.

We would like to thank all authors for their contributions, 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 2008

Matteo Baldoni
Tran Cao Son
M. Birna van Riemsdijk
Michael Winikoff

Organization

Workshop Organizers

Matteo Baldoni	University of Turin, Italy
Tran Cao Son	New Mexico State University, USA
M. Birna van Riemsdijk	Delft University of Technology, The Netherlands
Michael Winikoff	University of Otago, New Zealand; and RMIT University, Australia

Program Committee

Thomas Agotnes	Bergen University College, Norway
Marco Alberti	University of Ferrara, Italy
Natasha Alechina	University of Nottingham, UK
Grigoris Antoniou	University of Crete, Greece
Matteo Baldoni	University of Turin, Italy
Cristina Baroglio	University of Turin, Italy
Rafael Bordini	University of Durham, UK
Federico Chesani	University of Bologna, Italy
Amit Chopra	North Carolina State University, USA
Keith Clark	Imperial College London, UK
Francesco M. Donini	University of Tuscia, Italy
Benjamin Hirsch	Technical University Berlin, Germany
Shinichi Honiden	National Institute of Informatics, Japan
John Lloyd	Australian National University, Australia
Viviana Mascardi	University of Genoa, Italy
Nicolas Maudet	University of Paris-Dauphine, France
John-Jules Ch. Meyer	Utrecht University, The Netherlands
Enrico Pontelli	New Mexico State University, USA
Birna van Riemsdijk	Delft University of Technology, The Netherlands
Chiaki Sakama	Wakayama University, Japan
Tran Cao Son	New Mexico State University, USA
Wamberto Vasconcelos	University of Aberdeen, UK
Mirko Viroli	University of Bologna, Italy
Marina De Vos	University of Bath, UK
Michael Winikoff	University of Otago, New Zealand; and RMIT University, Australia

Steering Committee

João Leite	New University of Lisbon, Portugal
Andrea Omicini	University of Bologna-Cesena, Italy
Leon Sterling	University of Melbourne, Australia
Paolo Torroni	University of Bologna, Italy
Pınar Yolum	Bogazici University, Turkey

Additional Reviewers

James Harland	Marco Montali	Viviana Patti
Inoue Katsumi	Jonty Needham	Sebastian Sardina
Emiliano Lorini	Naoyuki Nide	

Sponsoring Institutions

Matteo Baldoni has partially been funded by the MIUR PRIN 2005 "Specification and verification of agent interaction protocols" national project.

M. Birna van Riemsdijk has partially been supported by the EU project SENSORIA (IST-2005-016004), which is part of the 6th Framework Programme.

Tran Cao Son has been supported by the NSF grants HRD 0420407, CNS 0220590, and IIS 0812267.

Table of Contents

Invited Papers

Contributed Papers

Specifying and Enforcing Norms in Artificial Institutions*

Nicoletta Fornara[1] and Marco Colombetti[1,2]

[1] Università della Svizzera italiana, via G. Buffi 13, 6900 Lugano, Switzerland
{nicoletta.fornara,marco.colombetti}@lu.unisi.ch
[2] Politecnico di Milano, piazza Leonardo Da Vinci 32, Milano, Italy
marco.colombetti@polimi.it

Abstract. In this paper we investigate two related aspects of the formalization of open interaction systems: how to specify norms, and how to enforce them by means of sanctions. The problem of specifying the sanctions associated with the violation of norms is crucial in an open system because, given that the compliance of autonomous agents to obligations and prohibitions cannot be taken for granted, norm enforcement is necessary to constrain the possible evolutions of the system, thus obtaining a degree of predictability that makes it rational for agents to interact with the system. In our model, we introduce a construct for the definition of norms in the design of artificial institutions, expressed in terms of roles and event times, which, when certain activating events take place, is transformed into commitments of the agents playing certain roles. Norms also specify different types of sanctions associated with their violation. In the paper, we analyze the concept of sanction in detail and propose a mechanism through which sanctions can be applied.

1 Introduction

In our previous works [10, 11, 26] we have presented a metamodel of artificial institutions called *OCeAN* (Ontology, CommitmEnts, Authorizations, Norms), which can be used to specify at a high level and in an unambiguous way *open interaction systems*, where heterogeneous and autonomous agents may interact.

In our view open interaction systems and artificial institutions, used to model them, are a technological extension of human reality, that is, they are an instrument by which human beings can enrich the type and the frequency of their interactions and overcome geographical distance. Potential users of this kind of systems are artificial agents, that can be more or less autonomous in making decisions on behalf of their owners, and human beings using an appropriate interface. For example, it is possible to devise an electronic auction where the artificial agents are autonomous in deciding the amount of their bids, or an interaction system for the organization of conferences in which human beings (like

* Supported by the Hasler Foundation project number 2204 title "Artificial institutions: specification of open distributed interaction systems".

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 1–17, 2009.

the organizers, or the Program Committee members) act by means of artificial agents that have a very limited level of autonomy. In any case it is important to remark that in every type of system there is always a stage when the software agents have to interface with their human owners to perform certain actions in the real world. For these reasons artificial institutions have to reflect, with the necessary simplifications, crucial aspects of their human counterparts. Therefore in devising our model we draw inspiration from an analysis of social reality [22] and from human legal theory [14].

In this paper we concentrate mainly on the definition of the constructs necessary for the specification of the normative component of artificial institutions, that is, of obligations, permissions and prohibitions of the interacting agents. The normative component is fundamental because it can be used to specify the expected behavior of the interacting agents, for example by means of flexible protocols [27]. We shall extend our *OCeAN* metamodel by defining a construct for the specification of norms for open systems, whose semantics is expressed by means of *social commitments*, the same concept that we have used to specify the semantics of a library of communicative acts [9, 10]. Commitments, having a well defined life-cycle, will be used at run-time to detect and react to the violation of the corresponding norms.

An important feature of our proposal, with respect to other ones [1, 5, 12, 19, 24], is that, using the construct of a commitment, it gives a uniform solution to two crucial problems: the specification of the semantics of norms and the definition of the semantics of an Agent Communication Language. Therefore a software agent able to reason on one construct is able to reason on both communicative acts and norms.

Moreover we present an innovative and detailed analysis of problem of defining a mechanism for enforcing obligations and prohibitions by means of sanctions that, that is, a treatment of the actions that have to be performed when a violation occurs, in order to deter agents from misbehaving and to secure and recover the system from an undesirable state. We speak of "obligation and prohibition enforcement" instead of "norm enforcement", as done in other approaches, because our proposal can be used to enforce obligations and prohibitions that derive either from predefined norms or from the autonomous performance of communicative acts.

The problem of managing sanctions has been tackled in a few other works: for example, López y López et al. [19] propose to enforce norms using the "enforcement norms" that oblige agents entitled to do so to punish misbehaving agents but does not treat the actions that the misbehaving agents may have to perform to repair to its violation; Vázquez-Salceda et al. [24] present, in the OMNI framework, a method to enforce norms described at a different level of abstraction but do not investigate in detail the mechanism to manage santctions; whereas Grossi et al. in [13] develop a high-level analysis of the problem of enforcing norms. Other interesting proposals introduce norms to regulate the interaction in open systems but, even when the problem of enforcement is considered to be crucial, do not investigate with sufficient depth why an agent ought

to comply with norms and what would happen if compliance does not occur. For instance, Esteva et al. [5, 12] propose ISLANDER, where a normative language with sanctions is defined but not discussed in detail, Boella et al. [3] model violations but do not analyze sanctions, and Artikis et al. [1] propose a model where the problem of norm enforcement using sanctions is mentioned but not fully investigated.

The paper is organized as follows: in Section 2 we briefly describe the part of metamodel for artificial institutions that we have presented in other works [10, 11, 26]. In Section 3 the reasons why in open interaction frameworks it makes sense to allow for the violation of obligations and prohibitions are discussed, and then in Section 4 a proposal on how to enforce obligations and prohibitions by means of sanctions is presented. In Section 5 our model of norms is described and the construct of commitment is extended, with respect to our previous works, by adding the treatment of sanctions. In Section 6 we briefly exemplify our proposal and finally in Section 7 we present conclusions.

2 The OCeAN Metamodel

In our previous works we have started to define the OCeAN metamodel [10, 11], that is, the set of concepts, briefly recalled in the sequel, that can be used to design artificial institutions. Examples of artificial institutions are the institution of language (that we call the *Basic Institution*, because we assume it will be used in the specification of every interaction system), the institution of English or Dutch auctions [10, 26], and the institution of organizations. In our view an open interaction system for autonomous agents can be specified using one or more artificial institutions. The state of the interaction system will then evolve on the basis of the events and actions that take place in the system, and whose effects are defined in the various institutions and on the basis of the life-cycle of the concepts defined in our model. (Investigating the relationships among the specification of different institutions is an interesting open problem [4], that we shall not tackle in this paper.) The concepts introduced by our metamodel are:

- The constructs necessary to define the *core ontology* of an institution, including: the notion of *entity*, used to define the concepts introduced by the institution (e.g., the notion of a run of an auction with its attributes introduced by the institution of auctions); the notion of *institutional action*, described by means of their preconditions and postconditions (e.g., the action of opening an auction, or declaring the current ask-price of an auction). The core ontology also defines the syntax of a list of *base-level actions*, like for instance the action of exchanging a message, whose function is to concretely execute institutional actions.
- Two fundamental concepts that are common to all artificial institutions: the notions of *role* and of *event*. In particular roles are used in the specification of authorizations and norms, while the happening of events is used to bring about the activation of a norm or to specify the initial or final instance of a time interval.

– The *counts-as relation* that is necessary for the concrete performance of institutional actions. In particular, such relation relies on a set of *conventions* that bind the exchange of a certain message, under a set of contextual conditions, to the execution of an institutional action. Contextual conditions include *authorizations* (called also *powers*) that specify what agents are authorized to perform certain institutional actions. Authorizations for the agent playing a given *role* to perform an institutional action *iaction* with a certain set of *parameters* if certain *conditions* are satisfied are represented with the following notation: *Auth(role,iaction(parameters), conditions)*.
– The construct of *norm* analyzed, discussed, and defined in Section 5, used to impose obligations and prohibitions to perform certain actions on agents interacting with the system.

3 Regimentation vs. Enforcement

In our model, as it will be discussed in more detail in Section 5, an active obligation is expressed by means of *commitments* to perform an action of a given type within a specified interval of time; similarly, an active prohibition is expressed by a commitment not to perform an action of a given type; moreover, every action is permitted unless it is explicitly forbidden. Note that a commitment can be created not only by the activation of a norm, but also by the performance of a communicative act [10], for instance by a promise.

In this section we briefly discuss the reasons why in open interaction systems it makes sense, and sometimes it is also inevitable, to allow for commitment violations, that happen when a prohibited action is performed or when an obligatory action is not performed within a predefined interval of time. The question is, why should we give an agent the possibility to violate commitments? why not adopt what in the literature is called "regimentation" [14], as proposed in [13], by introducing a control mechanism that does not allow agents to violate commitments?

To answer this question, it is useful to distinguish between obligations and prohibitions. With respect to obligations, there is only one way to "regiment" the performance of an obliged action, that is, by making the system performing the obliged action instead of a misbehaving agent. But this solution is not always viable, especially when the agent has to set the values of some parameters of the action. For instance, the auctioneer of a Dutch Auction is repeatedly obliged to declare a new ask price, lower than the one previously declared, but can autonomously decide the value of the decrement; therefore it would be difficult for the system to perform the action on behalf of the auctioneer. In any case it has to be taken into account that, even if the regimentation of obligations violates the autonomy of self-interested interacting agents, sometimes it can be adopted to recover the system from an undesirable state.

With respect to the regimentation of prohibitions, it is useful to introduce a further distinction between *natural* (or physical) actions (like opening a door or

physically delivering a product), whose effects take place thanks to physical laws, and *institutional* actions (like opening an auction or transferring the property of a product), whose effects take place thanks to the common agreement of the interacting agents (more precisely, of their designers). For our current purpose, the main difference between natural and institutional actions is that, under suitable conditions, the latter can be "voided", that is, their institutional effects can be nullified; on the contrary, this is not possible with natural actions. Consider for example the difference between destroying and selling an object: while in general a destroyed object cannot be brought back into existence, the transfer of ownership involved in selling it can always be nullified. The previous considerations imply that, in general, it is impossible to use regimentation to prevent the violation of a prohibition to perform a natural action. Concerning the prohibition of institutional actions, in our model it can be expressed using two different mechanisms: (i) through the lack of authorization: in fact, when an agent performs a base-level action bound by a convention to an institutional action a_i, but the agent is not authorized to perform a_i, neither the "counts-as" relation nor the effects of a_i take place; (ii) through a commitment not to perform such an action: in this case, if the action is authorized, its effects do take place but the corresponding commitment is violated. The solution to block the effects of certain actions by changing their authorizations during the life of the system is adopted for instance in AMELI (an infrastructure that mediates agent interactions by enforcing institutional rules) by means of *governors* [6], which filter the agents' actions letting only the allowed actions to be performed. However, this solution is not feasible when more than one institution contributes to the definition of an interaction system, as it happens for example when the Dutch Auction and the Auction-House institutions contribute to the specification of an interaction system as presented in [26] and briefly recalled in Section 6. In such cases, an action authorized by an institution cannot be voided by another institution, which can at most prohibit it.

It is moreover important to remark that in an open system, where heterogeneous agents interact exhibiting self-interested behavior based on a hidden utility function, it is impossible to predict at design phase all the interesting and fruitful behaviors that may emerge. To reach an optimal solution for all participants [28] it may be profitable to allow agents to violate their obligations and prohibitions.

We therefore conclude that regimenting an artificial system so that violations of commitments are completely avoided is often impossible and sometimes even detrimental, since it may preclude interesting evolutions of the system towards results that are impossible to foresee at design time. It is also true, however, that in order to make the evolution of the system at least partially predictable, misbehavior must be reduced to a minimum. But then, how is it possible to deter agents from violating commitments? An operational proposal to tackle this problem, based on the notion of sanction, is described in the following sections.

4 Sanctions

In this section we briefly discuss the crucial role played by *sanctions* in the specification of an open interaction system. In the Merriam-Webster On Line Dictionary[1] a sanction is defined as "the detriment, loss of reward, or coercive intervention annexed to a violation of a law as a means of enforcing the law". In an artificial system, even if the utility function of the misbehaving agent is not known, sanctions can be mainly devised to deter agents from misbehaving bringing about a loss for them in case of violation, under the assumption that the interacting heterogeneous agents are human beings or artificial agents able to reason on sanctions. Moreover sanctions can be devised to compensate the institution or other damaged agents for their loss due to the misbehavior of the agents; to contribute to the security of the system, for example by prohibiting misbehaving agents to interact any longer with the system; and to specify the acts that have to be performed to recover the system from an undesirable state [23].

When thinking about sanctions from an operational point of view, and in particular to the set of actions that have to be performed when a violation occurs, it is important to distinguish between two types of actions that differ mainly as far as their actors are concerned:

- One crucial type of action that deserves to be analyzed in detail, and that is not taken into account in other proposals [12, 19, 24], consists of the actions that the misbehaving agent itself has to perform when a violation occurs, and that are devised as a deterrent and/or a compensation for the violation. For instance, an unruly agent may have to pay a fine or compensate another agent for the damage. When trying to model this type of action it is important to take into account that it is also necessary to check that the compensating actions are performed and, if not, to sanction again the agent or, in some situations, to give it a new possibility to remedy the situation.
- Another type is characterized by the actions that certain agents are *authorized* to perform only against violations. In other existing proposals, for instance [19, 24], which do not highlight the notion of authorization (or power [15]), those actions are simply the actions that certain agents are obliged to perform against violations. From our point of view, instead, the obligation to sanction a violation should be distinguished from the authorization to do so. The reason why authorizations are crucial is obvious: sanctions can only be issued by agents playing certain specific roles in an institution. But an authorization does not always carry an obligation with it.

In some situations, and in particular when the sanction is crucial for the continuation of the interaction, one may want to express the obligation for authorized agents to react to violations by defining an appropriate new norm. For instance, in the organization of a conference if a referee does not meet the deadline for submitting a review, the organizers are not only authorized, but also obliged to reassign the paper to another referee. The norm that may be introduced to

[1] <http://www.m-w.com>

oblige the agents entitled to do so to manage the violation is similar to the "enforcement norm" proposed in [19]: it has to be activated by a violation and its content has to coincide with the sanctions of the violated obligation or prohibition. This norm may in turn be violated, and it is up to the designer of the system to decide when to stop the potentially infinite chain of violations and sanctions, leaving some violation unpunished.

Regarding this aspect, to make it reasonable for certain agents (or for their owner) to interact with an open system, it has to be possible to specify that certain violations will definitely be punished (assuming that there are not software failures). One approach is to specify that the actor of the actions performed as sanctions for those violations is the *interaction-system* itself, that therefore needs to be represented in our model as a "special agent". By "special" we mean that such an agent will not be able to take autonomous decisions, and will only be able to follow the system specifications that are stated before the interaction starts. We call this type of agents *heteronomous* (as opposite to autonomous). Given that the *interaction-system* can become, in an actual implementation, the actor of numerous actions performed as sanctions, it would be better to implement it in a distributed manner in order to avoid that it becomes a possible bottleneck.

Examples of reasonable sanctions that can be inflicted by means of norms in an open artificial system are the decrement of the trust or reputation level of the agent (similar to the reduction of the driving licence points that is nowadays applied in some countries), the revocation of the authorization to perform certain actions or a change of role (similar to confiscation of the driving licence) or, as a final action, the expulsion of the agent from the system. Another type of sanction typical of certain contracts (i.e., sets of correlated commitments created by performing certain communicative acts) is the authorization for an agent to break its part of the contract, without incurring a violation, if the counterpart has violated its own commitments.

5 Norms

Norms are taken as a specification of how a system ought to evolve. In an open system, they are necessary to impose obligations and prohibitions to the interacting agents, in order to make the system's evolution at least partially predictable [2, 20]. In particular, norms can be used to express interaction protocols as exemplified in [10, 26], where the English Auction and the Dutch Auction are specified by indicating what agents can do, cannot do, and have to do at each state of the interaction.

At design time, the main point is to guarantee that the system has certain crucial properties. This result can be achieved by formalizing obligations and prohibitions by means of logic and applying model checking techniques as studied in [17, 25]. At run time, and from the point of view of the interacting agents, norms can be used to reason about the relative utility of future actions [18]. Still at run time, but from the point of view of the open interaction system,

norms can be used to check whether the agents' behavior is compliant with the specifications and able to suitably react to violations. Our model of norms is mainly suited for the last task.

Coherent with other approaches [1, 5, 12, 19, 24], in our view norms have to specify who is affected by them, who is the creditor, what are the actions that should or should not be performed, when a given norm is active, and what are the consequences of violating norms. For instance, a norm of a university may state that a professor has to be ready to give exams any day from the middle to the end of February, otherwise the dean is authorized to lower the professor's public reputation level.

In the definition of our model it is crucial to distinguish between the definition of a construct for the specification of norms in the design phase, that will be used by human designers, and the specification of how such a construct will evolve during the run-time phase to make it possible to detect and react to norm violations. In particular we assume that during the run-time of the system the interacting agents cannot create new norms, but can create new commitments, directed to specific agents, by performing suitable communicative acts, for example by making promises or by giving orders.

During the phase of specification of the set of norms of a certain artificial institution the designer does not know the actual set of agents that will interact with the system at a given time. In this phase it is therefore necessary to define norms based on the notion of role. Moreover, the time instant at which a norm becomes active is typically not known at design time, being related to the occurrence of certain events; for example, the agent playing the role of the auctioneer in an English auction is obliged to declare the current ask-price after receiving each bid by a participant. Therefore at design phase it is only possible to specify the type of event that, if it happens, will activate the norm.

During the system run time such a construct of norm, expressed in terms of roles and times of events, must be transformed into an unambiguous representation of the obligations and prohibitions that every agent has at every state of the interaction. To tackle this problem we propose to use Event-Condition-Action (ECA) rules to transform the norms given in the design phase into concrete commitments, whose operational semantics is given in our previous work [10] and will be extended in Section 5.2. The main advantage of using commitments to express active obligations and permissions is that the same construct is also used in our model of institutions to express the semantics of numerous communicative acts [10]. Interacting agents may therefore be designed to reason on just one construct to make them able to reason on all their obligations and prohibitions, derived both from norms and from the performance of communicative acts.

5.1 The Construct of Norm

First of all a norm is used to impose a certain behavior on certain agents in the system. Therefore a norm is applied to a set of agents, identified by means of the *debtors* attribute, on the basis of the roles they play in the system.

Another fundamental component of a norm is its *content*, which describes the actions that the debtors have to perform (if the norm expresses an obligation) or not to perform (if the norm expresses a prohibition) within a specified interval of time. In our model *temporal propositions*, which are defined by the Basic Institution (for a detailed treatment see [8]), are used to represent the content of commitments and, due to the strict connection between commitments and norms, are also used to represent the content of norms. A temporal proposition binds a *statement* about a state of affairs or about the performance of an action to a specific *interval of time* with a certain *mode* (that can be $\forall$ or $\exists$). Temporal propositions are represented with the following notation:

$$TP(statement, [t_{start}, t_{end}], mode, truth\text{-}value),$$

where the *truth-value* could be undefined ($\perp$), true or false. In particular when the *statement* represents the performance of an action and the *mode* is $\exists$, the norm is an obligation and the debtors of the norms have to perform the action within the interval of time. When the *statement* represents the non-performance of an action and the *mode* is $\forall$ the norm is a prohibition and the debtors of the norms should not perform the action within the interval of time. In particular t_{start} is always equal to the time of occurrence of the event that activates the norm. Regarding the verification of prohibitions, in order to be able to check that an action has not been performed during an interval of time it is necessary to rely on the closure assumption that if an action is not recorded as happened in the system, then it has not happened.

A norm becomes active when the *activation event* e_{start} happens. Activation can also depend on some Boolean *conditions*, that have to be true in order that the norm can become active; for instance an auctioneer may be obliged to open a run of an auction at time t_{start} if at least two participants are present.

An agent can reason whether to fulfil or not to fulfil a norm on the basis of the sanctions/reward (as discussed later) and of whom is the *creditor* of the norm, as proposed also in [16, 19]. For example, an agent with the role of auctioneer may decide to violate a norm imposed by the auction house if it is in conflict with another norm that regulates trade transactions in a certain country. Moreover the creditor of a norm is crucial because, given that it becomes the creditor of the commitments generated by the norm (as described in next section), is the only agent authorized to cancel such commitment [10]. In particular the cancelation of the commitment generated by the activation of a norm coincides with the operation of *exempting* an agent from obeying the norm in certain circumstances. Like for the *debtors* attribute, it is useful to express the creditor of declarative norms by means of their role. For instance, a norm may state that an employee is obliged to report to his director on the last day of each month; this norm will become active on the last day of each month and will be represented by means of a set of commitments, each having an actual employee as the debtor, and the employee's director as creditor.

Sometimes it may be useful to take the creditor of norms to be an *institutionalized agent*, that typically represents a human organization, like a university, a hospital, or a company, which can be regarded as the creditors of their bylaws.

In the human world, an institutionalized agent is an abstract entity that can perform actions only through a human being, who is its legal representative and has the right *mandate* [21]. On the contrary, in an artificial system it is always possible to create an agent that represents an organization but can directly execute actions. Therefore we prefer to view an institutionalized agent as a special role that can be assigned to one and only one agent having the appropriate authorizations, obligations, and prohibitions.

In order to enforce norms it is necessary to specify sanctions. More precisely, as discussed in the previous section, it is necessary to specify what actions have to be performed, when a violation occurs, by the debtors of a norm and by the agent(s) in charge of norm enforcement. These two types of actions, that we respectively call *a-sanctions* (active sanctions) and *p-sanctions* (passive sanctions) are sharply dissimilar, and thus require a different treatment. More specifically, to specify an *a-sanction* means to describe an action that the violator should perform in order to extinguish its violation; therefore, an *a-sanction* can be specified through a temporal proposition representing an action. On the contrary, to specify a *p-sanction* means to describe what actions the norm enforcer is authorized to perform in the face of a violation; therefore, a *p-sanction* can be specified by representing a suitable set of authorizations.

Regarding *a-sanctions*, it is necessary to consider that a violating agent may have more than one possibility to extinguish its violation. For example, an agent may have to pay a fine of x euro within one month, and failing to do so may have to pay a fine of $2 * x$ euro within two months. In principle we may regard the second sanction as a compensation for not paying the first fine in due time, but this approach would require an unnecessarily complex procedure of violation detection. Given that any Boolean combination of temporal propositions is still a temporal proposition, and that the truth-value of the resulting temporal proposition can be obtained from the truth-values of its components using an extended truth table to manage the indefinite truth-value [7], a more viable solution consists in specifying every possible action with a different temporal proposition, and combining them using the *OR* operator.

In summary, in our model the construct of norm is characterized by the following attributes having the specified domains:

debtors:	*role*;
creditor:	*role*;
content:	*temporal proposition*;
e_{start}:	*event-template*;
conditions:	*Boolean expression*;
a-sanctions:	*temporal proposition*;
p-sanctions:	*authorization*;

5.2 Commitments with Sanctions

In order to give an intuitive operational semantics to the construct of norms introduced so far, we now describe a mechanism to transform them, at run time,

into *commitments* relative to specific agent and time interval. The transformation of norms defined at design time in commitments at run time is crucial because they are the mechanisms used to detect and react to violations. Moreover given that the activation event of norms may happen more than once in the life of the system, it is possible to distinguish between different activations and, in case, violations of the same norm. Given that our previous treatment of *commitment* [8, 10] does not cover sanctions, in this section we extend it to cover this aspect.

In our model a special institution, the Basic Institution, defines the construct of commitment, which is represented with the following notation:

$$Comm(state, debtor, creditor, content).$$

The content of commitments is expressed using *temporal propositions* (briefly recalled in Section 5). The *state* of a commitment, as described in Figure 1, can change as an effect of the execution of institutional actions (solid lines) or of environmental events (dotted lines). Relevant events for the life cycle of commitments are due to the change of the truth-value of the commitment's content. If the content becomes true an event-driven routine (as discussed in detail in [26]) automatically changes the commitment's state to fulfilled, otherwise it becomes violated. In particular the *unset* state is used to represent commitments created by means of a request communicative act and that have not been already accepted by their debtor.

In our view an operational model of sanctions has to specify how to detect: (i), that a commitment has been violated (a mechanism already introduced in our model of commitment); (ii), that the debtor of the violated commitment performs the compensating actions; and (iii), that the agents entitled to enforce the norms have managed the violation by performing certain actions.

Regarding the necessity to check that the debtor performs the compensating actions, one solution may be to create a new commitment to perform those actions. But this solution brings in the problem of taking trace that the violation of a given commitment is extinguished by the fulfillment of another commitment. A simpler and more elegant solution consists in adding two new attributes, *a-sanctions* and *p-sanctions*, to commitments, and two new states, *extinguished* and *irrecoverable*, to their life-cycle. The value of the *a-sanctions* attribute is a temporal proposition describing the actions that the debtor of the commitment has to perform, within a given interval of time, to remedy the violation. If the actions indicated in the *a-sanctions* attribute are performed, the truth-value of the related temporal proposition becomes true and an event driven routine automatically changes the state of the violated commitment to *extinguished*, as reported in Figure 1. Analogously, if the debtor does not perform those actions, at the end of the specified time interval the truth-value of the temporal proposition becomes false and the state of the commitment becomes *irrecoverable*. Similarly to what we did for norms, the actions that certain agents are authorized to perform against the violation of the commitment are represented in the *p-sanctions* attribute. Note that whether such actions are or are not performed does not affect the life cycle of the commitment; this depends on the fact that

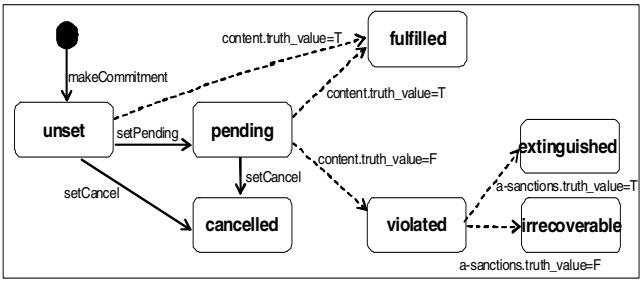

Fig. 1. The life-cycle of commitments

the agent that violated a commitment cannot be held responsible for a possible failure of other agents to actually carry out the actions they are authorized to perform.

Finally, for proper management of violation it may be necessary to trace the source of a commitment, either deriving it from the activation of a norm or from the performance of a communicative act. In order to represent this aspect we add to commitments an optional attribute called *source*. Our enriched notion of commitment is therefore represented with the following notation:

$Comm(state, debtor, creditor, content, a\text{-}sanctions, p\text{-}sanctions, source)$.

In our model we use *ECA-rules* (Event-Condition-Action rules), inspired by Active Database models, to specify that certain *actions* are executed when an event identified by an *event-templates* happens, provided that certain Boolean *conditions* are true. The semantics of ECA rules is given as usual: when an event matching the event template occurs in the system, the variable e is filled with the event instance and the condition is evaluated; if the condition is satisfied, the set of actions are executed and their effects are brought about in the system. The *interaction-system* agent (see Section 4) is the actor of the actions performed by means of ECA-rules, and has to have the necessary authorization in order to perform them. In our model, ECA rules are specified according to the following notation:

on e: *event-template*
if *condition* **then**
  do $action(parameters)^+$

In particular the following two ECA-rules have to be present in every interaction system. One in necessary to transform at run time norms into commitments: when the activation event of the norm happens, the *makePending-Comm* institutional action is performed and creates a pending commitment for each agent playing one of the roles specified in the *debtors* attribute of the norm:

on e_{start}
if $norm.conditions$ **then**
do foreach $agent \mid agent.role$ **in** $norm.debtors$
do $makePendingComm(agent, norm.creditor, norm.content$
$norm.a\text{-}sanctions, norm.p\text{-}sanctions, norm\text{-}ref)$

The other is necessary to give the authorizations expressed in the *p-sanctions* attributes to the relevant agents when a commitment is violated:

on e: $AttributeChange(comm.state, violated)$
if $true$ **then**
do foreach $auth$ **in** $comm.p\text{-}sanctions$
do $createAuth(auth.role, auth.iaction)$

The *createAuth(role,iaction)* institutional action creates the authorization for the agents playing a certain role to perform a certain institutional action. We assume that the *interaction-system* (the actor of ECA-rules) is always authorized to create new authorizations. A similar ECA-rule has also to be defined to remove such authorizations once *iaction* has been performed.

In certain systems, to guarantee that the *interaction-system* actually performs the actions specified in the *p-sanctions* attribute, it is also possible to create the following ECA-rule that reacts to commitment violations performing those actions:

on e: $AttributeChange(comm.state, violated)$
if $true$ **then**
do foreach $auth$ **in** $comm.p\text{-}sanctions$
if $auth.role = interaction\text{-}system$
do $auth.iaction(parameters)$

6 Example

An interesting example that highlights the importance of a clear distinction between permission and authorization, which becomes relevant when more than one institution is used to specify the interaction system, is the specification of the Dutch Auction as discussed in [26].

One of the norms of the Dutch Auction obliges the auctioneer to declare a new ask-price (within λ seconds) lowering the previous one by a certain amount κ, on condition that δ seconds have elapsed from the last declaration of the ask-price without any acceptance act from the participants. If the auctioneer violates this norm the interaction-system is authorized to declare the ask-price and to lower the auctioneer's public reputation level (obviously there is no need of an authorization to change a private reputation level), while the auctioneer has to pay a fine (within h seconds) to extinguish its violation. Such a norm can be expressed in the following way:

$$
\begin{array}{ll}
debtors= & auctioneer; \\
creditor= & auction\text{-}house; \\
content= & TP(setAskPrice(DutchAuction.LastPrice\text{-}\kappa), \\
 & \quad [time\text{-}of(e_{start}), time\text{-}of(e_{start}) + \lambda], \exists, \bot); \\
e_{start}= & TimeEvent(DutchAuction.timeLastPrice + \delta); \\
conditions= & DutchAuction.offer.value = null; \\
a\text{-}sanctions= & TP(pay(ask\text{-}price, interaction\text{-}system), \\
 & \quad [time\text{-}of(e), time\text{-}of(e) + h], \exists, \bot); \\
p\text{-}sanctions= & Auth(interaction\text{-}system, setAskPrice(DutchAuction.LastPrice\text{-}\kappa)), \\
 & Auth(interaction\text{-}system, ChangeReputation(auctioneer, value));
\end{array}
$$

where variable e refers to the event that happens if the commitment generated at run-time by this norm is violated.

At the same time, the seller of a product can fix the minimum price ($minPrice$) at which the product can be sold, for example by means of an act of proposal [7]. The auction house, by means of its auctioneer, sells the product in a run of the Dutch Auction where the auctioneer is authorized to lower the price to a predetermined *reservation price*. The reservation price fixed by the auction house can be lower than $minPrice$, for example because in previous runs of the auction the product remained unsold. If the auctioneer actually sells the product at a price ($winnerPrice$) lower than $minPrice$, the sale is valid but the auction house violates its commitment with the seller of the product and will incur the corresponding sanctions; for example, it may have to refund the seller, while the seller is authorized to lower the reputation of the auction house. This situation can be modelled by the following commitment between the seller and the auction house:

$$
\begin{array}{ll}
state= & pending; \\
debtor= & auction\text{-}house; \\
creditor= & seller; \\
content= & TP(not\ setCurPrice(p) \mid p < minPrice, [now, +\infty)], \forall, \bot); \\
a\text{-}sanctions= & TP(pay(seller, minPrice\text{-}winnerPrice), \\
 & \quad [time\text{-}of(e), time\text{-}of(e)+15days], \exists, \bot); \\
p\text{-}sanctions= & Auth(seller, ChangeReputation(auction\text{-}house, value));
\end{array}
$$

where variable e refers to the event that happens if the commitment is violated.

7 Conclusions

In this paper we have discussed the importance of formalizing and enforcing obligations and prohibitions in the specification of open interaction frameworks. We have proposed a construct to define norms in the design of institutions expressed in terms of roles and event times. The operational semantics of norms is defined by the commitments they generate through ECA-rules.

The innovative aspects of our proposal are the definition of different types of sanctions and of the operational mechanisms for monitoring the behavior of the agents and reacting to commitment violations. In particular, an interesting feature of our proposal is that the construct of commitment is uniformly used to

model the semantics of communicative acts and of norms. Differently from [19] our model of norms specifies the interval of time within which norms are active. Thanks to their transformation into commitments, it is possible to apply certain norms (whose activation event may happen many times) more than once in the life of the system. Another crucial aspect of our norms is that, differently from [19], they are activated by the occurrence of events and not simply if a certain state holds. Regarding the treatment of sanctions our model is more in-depth with respect to other proposals [13, 19, 24] because we distinguish the actions of the debtors from the actions of the other agents that are entitled to react to violations. In particular, regarding the actions of the debtors, we propose an effective solution for managing multiple sanctions, that is, multiple possibilities to compensate the violation (for example, paying an increasing amount of money), without entering in an infinite loop of checking violations and applying punishments. Regarding the sanctions applied by other agents, we discussed the reasons why a norm expresses what actions are authorized against violations and the reasons why some norms may be enforced by the interaction-system itself, which is treated as a special heteronomous agent.

Acknowledgements

We would like to thank Bertil Cottier, professor of Law at Università della Svizzera italiana, for helping us to improve our knowledge on the legal aspects of 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. Barbuceanu, M., Gray, T., Mankovski, S.: Coordinating with obligations. In: Sycara, K.P., Wooldridge, M. (eds.) Proceedings of the 2nd International Conference on Autonomous Agents (Agents 1998), pp. 62–69. ACM Press, New York (1998)
3. Boella, G., van der Torre, L.: Contracts as legal institutions in organizations of autonomous agents. In: Dignum, V., Corkill, D., Jonker, C., Dignum, F. (eds.) Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004), pp. 948–955. IEEE Computer Society, Los Alamitos (2004)
4. Cliffe, O., Vos, M.D., Padget, J.: Specifying and Reasoning About Multiple Institutions. In: Noriega, P., Vázquez-Salceda, J., Boella, G., Boissier, O., Dignum, V., Fornara, N., Matson, E. (eds.) COIN 2006. LNCS (LNAI), vol. 4386, pp. 67–85. Springer, Heidelberg (2007)
5. Esteva, M., Padget, J., Sierra, C.: Formalizing a language for institutions and norms. In: Meyer, J.-J.C., Tambe, M. (eds.) ATAL 2001. LNCS, vol. 2333, pp. 348–366. Springer, Heidelberg (2002)

6. Esteva, M., Rodríguez-Aguilar, J.A., Rosell, B., Arcos, J.L.: AMELI: An Agent-based Middleware for Electronic Institutions. In: Jennings, N.R., Sierra, C., Sonenberg, L., Tambe, M. (eds.) Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004), pp. 236–243. ACM Press, New York (2004)
7. Fornara, N.: Interaction and Communication among Autonomous Agents in Multiagent Systems. PhD thesis, Faculty of Communication Sciences, University of Lugano, Switzerland (2003), http://doc.rero.ch
8. Fornara, N., Colombetti, M.: A commitment-based approach to agent communication. Applied Artificial Intelligence an International Journal 18(9-10), 853–866 (2004)
9. Fornara, N., Viganò, F., Colombetti, M.: Agent communication and institutional reality. In: van Eijk, R., Huget, M., Dignum, F. (eds.) AC 2004. LNCS, vol. 3396, pp. 1–17. Springer, Heidelberg (2005)
10. Fornara, N., Viganò, F., Colombetti, M.: Agent communication and artificial institutions. Autonomous Agents and Multi-Agent Systems 14(2), 121–142 (2007)
11. 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)
12. Garcia-Camino, A., Noriega, P., Rodriguez-Aguilar, J.A.: Implementing norms in electronic institutions. In: Proceedings of the 4th International Joint Conference on Autonomous agents and Multi-Agent Systems (AAMAS 2005), pp. 667–673. ACM Press, New York (2005)
13. Grossi, D., Aldewereld, H., Dignum, F.: Ubi lex, ibi poena: Designing norm enforcement in e-institutions. In: Noriega, P., Vázquez-Salceda, J., Boella, G., Boissier, O., Dignum, V., Fornara, N., Matson, E. (eds.) COIN 2006. LNCS (LNAI), vol. 4386, pp. 101–114. Springer, Heidelberg (2007)
14. Hart, H.L.A.: The Concept of Law. Clarendon Press, Oxford (1961)
15. Jones, A., Sergot, M.J.: A formal characterisation of institutionalised power. Journal of the IGPL 4(3), 429–445 (1996)
16. Kagal, L., Finin, T.: Modeling Conversation Policies using Permissions and Obligations. In: van Eijk, R., Huget, M., Dignum, F. (eds.) AC 2004. LNCS, vol. 3396, pp. 123–133. Springer, Heidelberg (2005)
17. Lomuscio, A., Sergot, M.: A formulation of violation, error recovery, and enforcement in the bit transmission problem. Journal of Applied Logic (Selected articles from DEON 2002 - London) 1(2), 93–116 (2002)
18. López y López, F., Luck, M., d'Inverno, M.: Normative Agent Reasoning in Dynamic Societies. In: Jennings, N.R., Sierra, C., Sonenberg, L., Tambe, M. (eds.) Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004), pp. 535–542. ACM Press, New York (2004)
19. López y López, F., 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)
20. Moses, Y., Tennenholtz, M.: Artificial social systems. Computers and AI 14(6), 533–562 (1995)
21. Pacheco, O., Carmo, J.: A Role Based Model for the Normative Specification of Organized Collective Agency and Agents Interaction. Autonomous Agents and Multi-Agent Systems 6(2), 145–184 (2003)
22. Searle, J.R.: The construction of social reality. Free Press, New York (1995)

23. Vázquez-Salceda, J., Aldewereld, H., Dignum, F.: Implementing Norms in Multiagent Systems. In: Lindemann, G., Denzinger, J., Timm, I.J., Unland, R. (eds.) MATES 2004. LNCS (LNAI), vol. 3187, pp. 313–327. Springer, Heidelberg (2004)
24. Vázquez-Salceda, J., Dignum, V., Dignum, F.: Organizing multiagent systems. Autonomous Agents and Multi-Agent Systems 11(3), 307–360 (2005)
25. Viganò, F.: A Framework for Model Checking Institutions. In: Edelkamp, S., Lomuscio, A. (eds.) MoChArt IV. LNCS, vol. 4428, pp. 129–145. Springer, Heidelberg (2007)
26. Viganò, F., Fornara, N., Colombetti, M.: An Event Driven Approach to Norms in Artificial Institutions. In: Boissier, O., Padget, J., Dignum, V., Lindemann, G., Matson, E., Ossowski, S., Simao Sichman, J., Vázquez-Salceda, J. (eds.) ANIREM 2005 and OOOP 2005. LNCS (LNAI), vol. 3913, pp. 142–154. Springer, Heidelberg (2006)
27. 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)
28. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The Gaia methodology. ACM Transactions on Software Engineering and Methodology (TOSEM) 12(3), 317–370 (2003)

Social Norm Emergence in Virtual Agent Societies

Bastin Tony Roy Savarimuthu, Maryam Purvis, Martin Purvis, and Stephen Cranefield

Department of Information Science, University of Otago, Dunedin, P.O. Box 56, Dunedin, New Zealand
{tonyr,tehrany,mpurvis,scranefield}@infoscience.otago.ac.nz

Abstract. The advent of virtual environments such as SecondLife call for a distributed approach for norm emergence and spreading. In open virtual environments, monitoring various interacting agents (avatars), using a centralized authority might be computationally expensive. The number of possible states and actions of an agent could be huge. An approach for sustaining order and smoother functioning of these environments can be facilitated through norms. Agents can generate norms based on interactions. In particular, those social norms that incur certain cost to an individual agent but benefit the whole society are more interesting than those benefit both the agent and the society. The problem is that the selfish agents might not be willing to share the norm adherence cost. In this work, we experiment with notion proposed by Axelrod that social norms are best at preventing small defections where the cost of enforcement is low. We also study how common knowledge can be used to facilitate the overall benefit of the society. We believe our work can be used to facilitate norm emergence in virtual online societies.

1 Introduction

Norms are expectations of an agent about the behaviour of other agents in the society. The human society follows norms such as tipping in restaurants and exchange of gifts at Christmas. Norms are of interest to researchers because they help to improve the predictability of the society. They also reduce the computations required by an agent to make a decision. Norms have been of interest in different areas of research such as Sociology, Economics, Psychology and Computer science [1]. Norms have been shown to facilitate co-ordination and co-operation among the members of the society [2, 3]. Some of the well established norms may become laws.

While the discussion on how norms emerge and spread remains a research issue among scientists in Sociology, the advent of new ways of human interactions proxied through software agents in virtual 3D worlds such as SecondLife [4] have created interest among researchers in MAS to work on the applicability of the concept of norms in these digital societies.

We believe that software agents that operate autonomously or on behalf of human users in these virtual worlds cannot be effectively monitored and controlled

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 18–28, 2009.

through centralized policing mechanisms. The explosion of possible action states for an agent in a open environment is huge. It would be computationally expensive and taxing for a centralized monitor to enforce behavioural regularities to ensure smoother functioning of these systems. We believe that an alternative approach based on norms could be effectively used in such scenarios where norms can be derived and built using a bottom-up approach through interactions between the agents. Our objective in this paper is to experiment with the bottom-up approach where we observe whether a norm against littering a park spreads in an agent society.

The paper is organized as follows. Section 2 provides a brief overview on the work on norms with a focus on the norm emergence addressed in the field of multi-agent systems. The experimental set up used for conducting simulations on norm emergence is described in Section 3. The results are presented in Section 4. The discussion and future work are presented in Section 5.

2 Related Work on Norms

Due to multi-disciplinary interest in norms, several definitions for norms exist. Habermas [5], a renowned sociologist, identified norm regulated actions as one of the four action patterns in human behaviour. A norm to him means *fulfilling a generalized expectation of behaviour*, which is a widely accepted definition for social norms. When members of a society violate the societal norms, they may be punished (and even ostracized in some cases). Many social scientists have studied why norms are adhered. Some of the reasons for norm adherence include a) fear of authority b) rational appeal of the norms and c) feelings such as shame, embarrassment and guilt that arise because of non-adherence.

Social scientists have categorized norms into several categories [1]. One such categorization is based on the the cost-benefit analysis of a norm from a perspective of an individual agent and how that relates to the society as the whole, proposed by Horne [6]. There are four categories of norms according to Horne. There are norms which benefit both the individual agents and the society and some norms incur cost to both the agent and the society. There are some norms that benefit the agent but cost the society. Some norms cost the agent but are beneficial to the society. The last category of norms where the whole society is benefitted when an agent incurs a cost, is more interesting than the others. For these norms to be established there should be agents in the society that will help in the enforcement process. In the realm of digital societies such as that of SecondLife, it is important that there are these distributed enforcing agents that are helpful in regulating social order which is beneficial to the society.

2.1 Normative Multi-agent Systems

Norms have been of interest to multi-agent system researchers for over a decade [3,7,8]. Norms in multi-agent systems are treated as constraints on behaviour, goals to be achieved or as obligations [9]. The research on norms in sociology and multi-agent systems complement each other. Researchers in multi-agent systems

have borrowed ideas from sociology such as speech act theory and autonomy to model software agents. Sociologists on the other hand have used multi-agent systems for testing their theories through simulations.

The definition of normative multi-agent systems as described by the researchers involved in Normas 2008 workshop is as follows [10]: *A normative multi-agent system is a multi-agent system organized by means of mechanism to represent, communicate, distribute, detect, create, modify and enforce norms, and mechanism to deliberate about norms and detect norm violation and fulfillment.*

While some aspects of normative multi-agent systems such as normative system architectures, norm representations, norm adherence and violation detections [11,12,13] have received a lot of interest, areas such as norm creation and modification have not been explored in-depth. Our work in this paper borders the area of norm distribution (i.e. norm spreading) and the detection of the norm that has been created (through emergence).

2.2 Norm Spreading and Emergence

The concepts of norm spreading and norm emergence are related. A norm emerges in a society as a result of norm spreading mechanism. According to Boyd and Richerson [14], there are three ways by which a social norm can be propagated from one member of the society to another. They are a) Vertical transmission (from parents to offspring), b) Oblique transmission (from a leader of a society to the followers) and c) Horizontal transmission (from peer to peer interactions). Norm propagation is achieved by spreading and internalization of norms. Boman and Verhagen [7,15] have used the concept of normative advice (advice from the leader of a society) as one of the mechanisms for spreading and internalizing norms in an agent society. The work done by Savarimuthu et al. [16] uses a distributed approach for normative advise based on the notion of leadership. Another recent development is the consideration of the role of network topologies on norm emergence [17,18]. Sen et al. [19] have experimented with the emergence of traffic norm using social learning. In his well known work, Axelrod [2] has shown the role of metanorms to be effective in norm emergence. He also discusses several other approaches that might be useful for norm establishment which include the role of power, reputation, internalization and punishment. The contribution of this paper to this area are two fold. Firstly, we investigate Axelrod's statement that social norms are better suited for preventing smaller defections when the enforcement costs are low using social simulations in the context of norm emergence. Secondly, we introduce the notion of common knowledge that can help sustain norms in agent societies.

3 Experimental Setup and Parameters

Multi-Agent Based Simulation (MABS) is a inter-disciplinary research area which brings together researchers within the Agent-Based Social Simulation community (ABSS) [20] and the Multiagent Systems community (MAS). MABS

researchers use simulation as a mechanism to experiment with and validate new hypotheses (both social and computational theories). Simulations are used as tools that provide explanations for behaviours exhibited by complex systems. Simulation tools help researchers to investigate the implications of a strategies adopted by participating agents by running simulations starting from different initial conditions [21]. Adopting this approach, we have set up a simulation environment with multiple agents that interact in a social context.

We model agents in our virtual society as particles moving in a 2 dimensional space of linear size L. This virtual environment can be considered as a communal region such as a park. The agents explore and enjoy the park by moving around. Collisions of these particles in the virtual space represent interactions between agents in a social space as shown in figure 1. Each collision corresponds to two agents observing each other's actions. When two agents interact (when they meet each other within certain area of the park) they can observe each other performing one of the two actions, Litter (L) or Not Litter (NL), i.e. keep the environment clean. The payoff matrix that corresponds to the littering scenario is given in table 1.

An agent in our society starts with a score (s) of 100. When an agent litters, it gets a payoff of 0.5 while the cost associated with non littering is -0.5. These are the payoffs to an individual agent. When an agent litters, it pollutes the area

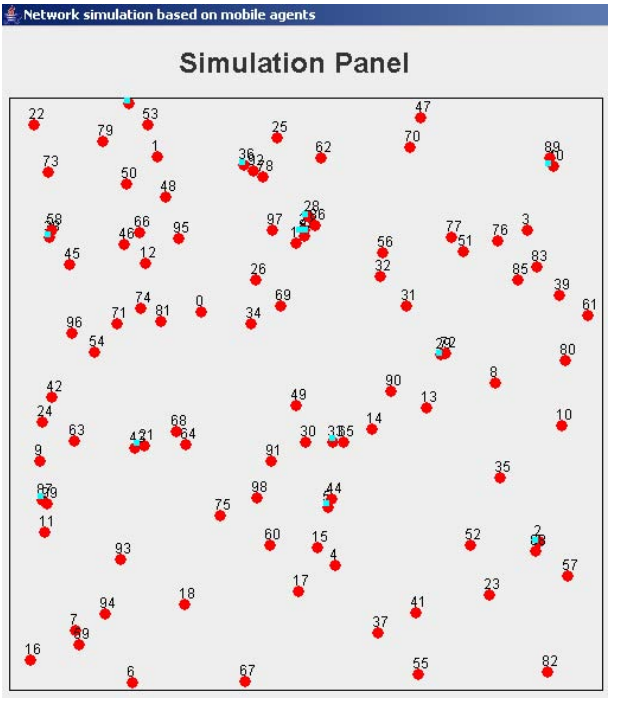

Fig. 1. Simulation of the park scenario (100 agents)

Table 1. Payoff matrix

	L	NL
L	0.5, 0.5	0.5,-0.5
NL	-0.5, 0.5	-0.5,-0.5

that belongs to the commons. So, this can be considered defecting the entire society. This impacts the productivity of the society. Productivity refers to the benefits that the park users will receive in using the park. This has been modeled using a variable *prod* that holds values from 0 to 1. The value of 1 represents the clean state of the park while 0 represents a littered, unusable park. Initially the productivity of the park is set to 1. When an agent litters, the productivity of the park goes down. For every X littering actions the productivity value is reduced by 0.01. The value of X was set to 10 in our system. The final payoff value is the sum of the individual payoff and the productivity of the system.

Let us now assume that the society does not have a law against littering and hence there is no centralized policing mechanism. In this scenario, any agent that believes that there should be no littering in the society, might choose to punish the other agent whom it observes littering. A punishment cost (P_{cost}) is incurred by the non-litterer when punishing a littering agent. Every agent in the society is initialized with an autonomy value from 0 to 9 based on a uniform distribution. Autonomy refers to the stubbornness of the agent. This value governs the number of punishments required by an agent to move from L to NL (change of strategies).

Another parameter that we have defined in the system is the minimum Survival score (S_{score}).When an agent's score, s goes below S_{score} or the productivity of the system goes below 0.5 ($prod < 0.5$), it changes its strategy (moves from L to NL). S_{score} is set to 50 in our experiments.

4 Experiments and Results

4.1 Role of Punishments with Low Enforcement Cost

In the first experiment there are 100 agents, 50 agents of type L and 50 of type NL. In every society there will be certain percentage of agents that are vengeful enough to punish another agent when they observe certain behaviour that they consider to be inappropriate. Let us assume that there are certain percentage of non-littering agents that are punishers (p=0.05, 0.10 and 0.25). P_{cost} is kept low (0.01) in these experiments.

In each iteration two agents are randomly chosen to interact. We conducted experiments over 6000 iterations. At the end of the simulation, we observe whether a littering or non-littering norm emerges. In our experiments we consider a norm to have emerged if all the agents are either of type L or NL (100% norm emergence). In other works, the value for norm emergence has varied from 70% to 100%.

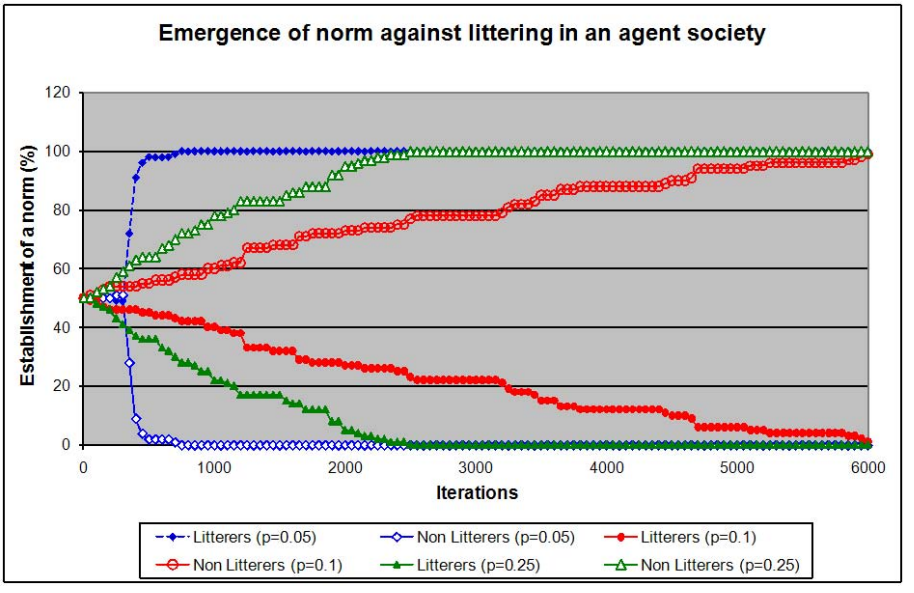

Fig. 2. Emergence of norm against littering in an agent society

Figure 2 shows 6 different lines. For each p value, there are two lines, one respresenting the number of litterers and the other representing the number of non-litterers. As the number of litterers decreases, the number of non-litterers increases (and vice versa). For this reason, these lines for a given p value, are the mirror images of each other. It is of interest to observe whether the littering or non-littering group reaches the value of 100. All the 6 lines start from a value of 50. Note that non-litterers are represented using hollow symbols while the litterers are shown using solid symbols.

It can be observed from figure 2 that as the number of punishers increases, the norm against littering is established. When the values of p are 0.10 and 0.25, the system converges to a norm against littering. When the value of p is 0.05, there aren't enough punishers in the system. Hence, the productivity of the society drops gradually and falls below the minimum level which results in the establishment of littering norm. Once there are adequate number of punishers, the cost of punishment is spread across the non-littering punishers, hence their individual scores do not reach the minimum threshold and they are successful in converting the litterers to become non-litterers.

Wikipedia is a good example of a system where more number of enforcers are needed to maintain the quality of the articles at a reasonable level. Members of Wikipedia have been successful in establishing a norm of collaboration using a peer to peer mechanism based on careful scrutiny of the content.

Figure 3 shows three different productivity lines that correspond to the results reported in figure 2. It can be observed that when there were adequate number of punishers $p = 0.1, 0.25$, the productivity of the society gradually improved

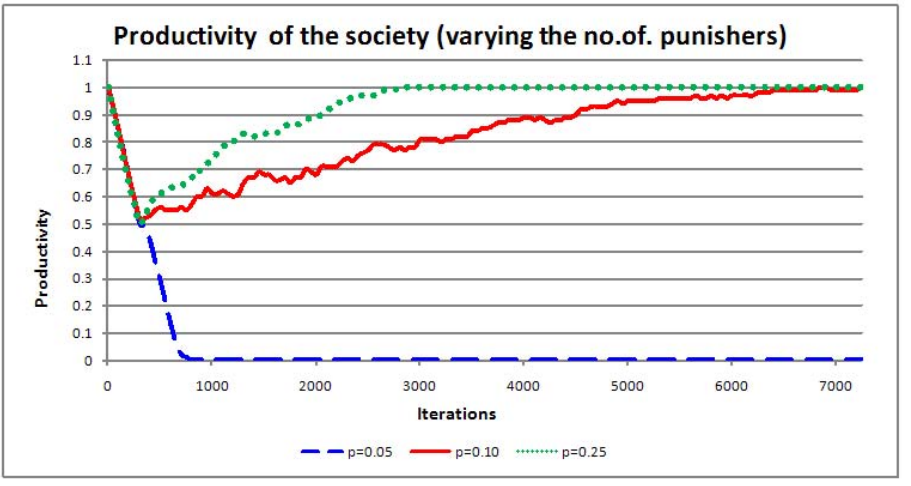

Fig. 3. Productivity of the society

and was sustained ($prod = 1$) in the end. When the litterers took over ($p = 0.05$) due to lack of enough punishers, the value of productivity plummeted to 0.

So, the important characteristics that governs this change are the autonomy of the individual agents (littering agents), the minimum score of productivity and the minimum threshold for survival (S_{score}). If a society has large autonomy values and high threshold for survival or productivity, the system will end up with litterers.

This type results can be observed in many social interactions. For example, when you go to a restaurant, if you were the first ones, you might be polite and keep your voice low when interacting with your friends. As the restaurant becomes crowded, you might observe the noise levels rising. As the noise level increases, there is no incentive for you to keep your voice down. Moreover, you might be forced to speak out loud as that is the only way you might be heard by others. This case is similar to the litterers becoming non-litterers after certain threshold is surpassed.

4.2 Role of Punishments with High Enforcement Cost

This experiment shows that when the enforcement cost increases the system drives all the agents to litter. This experiment shows the behaviour of the system for three different values of punishment cost ($P_{cost} = 0.1, 1, 10$). There are 25% punishers in a society for all the three experiments. Again, note that non-litterers are represented using hollow symbols while the litterers are shown using solid symbols.

It can be observed from figure 4 that lower enforcement costs ($P_{cost} = 0.1$, 1) resulted in a non-littering norm while the higher cost (P_{cost} =10) resulted in littering norm. When P_{cost} was set to 10 the society initially had more number of non-litterers (till iteration 1000). Soon, the survival score(S_{score}) of punishers

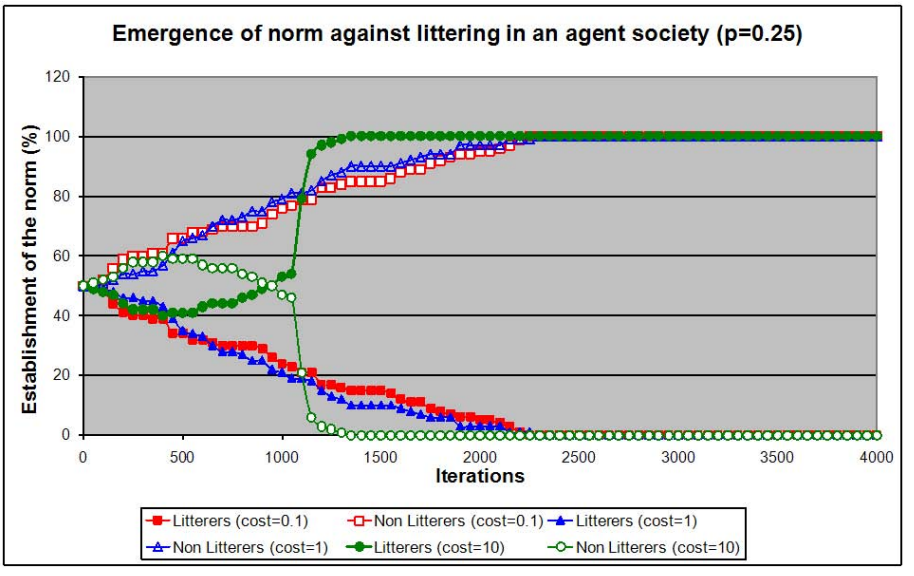

Fig. 4. Emergence of norm against littering in an agent society (p=0.25)

fell below the minimum threshold due to high punishment cost. The non-littering punishers then became litterers and hence the littering norm was established in the society.

From this experiment it can be inferred that social norms can successfully be established and sustained against smaller defections when the costs of enforcements are low. But for norms that require larger costs of punishment (e.g. honour killing), social norms might not be very useful. In those cases, institutionalized mechanisms such as laws would be best suited.

4.3 Conditional Punishment Based on Common Knowledge

In human societies we tend to gather information about the state of the world through certain common knowledge sources such as newspapers, television, radio and even from some influential, well connected people. At any point of time, not everyone in the society might know about the current state of an issue or a problem. But, once some information is available through the common knowledge sources, it can be assumed that there would be an increase in the awareness of the situation in the society. Chwe [22] describes the role of common knowledge in solving coordination problems. In this section we describe our experiment on how common knowledge helps with in the context of social littering.

Let us assume that a common knowledge source is available (e.g. a newspaper). This common knowledge source periodically informs the agents about the state of the park. The agents in the society can choose to look at the information available from the knowledge source periodically. Based on the information available, the agents can choose to react. For example, whenever the park's

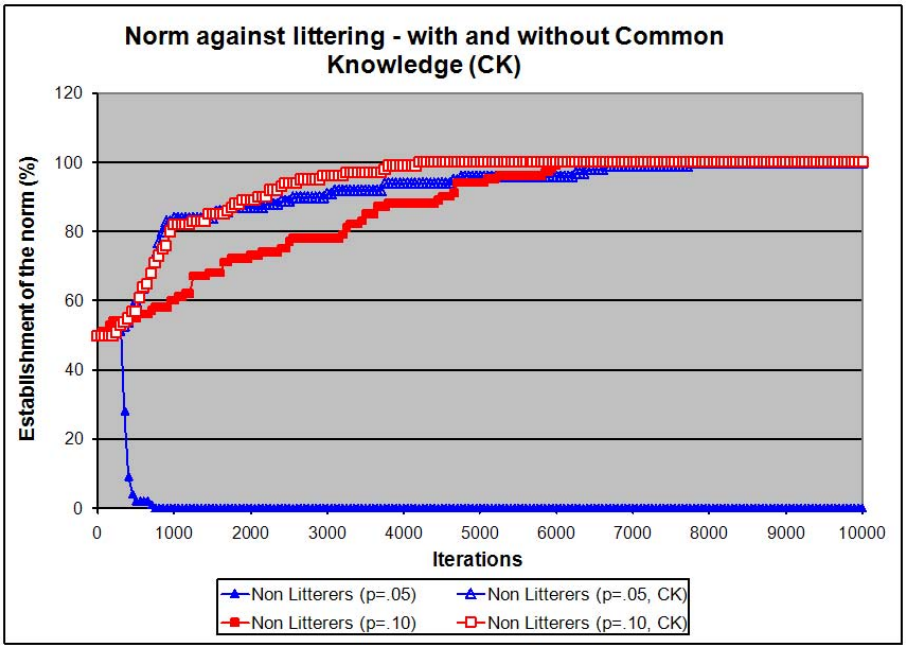

Fig. 5. Comparison of emergence of norm against littering - with and without Common Knowledge (CK)

productivity is less than certain value ($prod < 0.75$ in our experiment), the non littering agents can choose to punish. For example, say there were only 5% of the non-litterers were punishers originally. After the information is known to all the other non litterers (remaining 45% of non-litterers), can choose to punish the litterers based on a conditional probability which is their vengefulness value. Each non-littering punisher agent has a vengefulness value (V) which is similar to the autonomy value and this is initialized at the start of the experiments. A non-littering punisher with vengefulness value of 8 will punish a litterer 8 out of 10 times.

Figure 5 shows a comparison of punishment mechanisms with and without the use of common knowledge keeping P_{cost}=0.1 as a constant. The figure shows four lines that correspond to the establishment or fading of the non-littering norm. We have omitted the lines that show the trend lines of the littering norm for the sake of clarity of the diagram.

When p=0.05, the punishment mechanism that makes use of common knowledge results in a non-littering norm (hollow triangles) while the mechanism that does not use common knowledge results in a littering norm (solid triangles). When p=0.10, the punishment mechanism that uses common knowledge (hollow squares) converges faster than the one that does not use it (solid squares). So, it can be inferred that the availability of common knowledge has increased the rate of establishment of a norm against littering in one case (p=0.10) and has resulted in the emergence of a new norm in another (p=0.05).

5 Discussion and Future Work

The results presented in this paper demonstrate that norms can be established through a bottom-up process based on a distributed, peer to peer punishment mechanism. The results obtained in our experiments are in agreement with Axelrod's statement that the norms are best in preventing smaller defections when the cost of enforcement is low. Also, it was shown that common knowledge can be used as a mechanism for improving norm establishment when used in conjunction with the punishment mechanism.

We agree that our results are preliminary. However, this work is aimed towards experimenting with mechanisms that might be suitable for generating norms in a bottom-up approach than a prescriptive top-down approach. In particular, our work is relevant for virtual online societies where behavioural norms should be derived by the agents themselves rather than adhering to an enforced law.

We are currently extending our simulation scenario to include agents of different personality types. Another extension to our system is to experiment with the emergence of different norms among different sub-groups within an agent society. To achieve that we need an application domain that has more states than a simple co-ordination game. Another important extension is to test the model on network topologies as agents evolve norms based on the influence from agents that they are connected to. The concept of distributed norm emergence is applicable to many applications in agent societies (e.g. buyer-seller scenarios in supply chain management, file sharing). A fertile ground for the study and experimentation of new mechanisms for norm emergence are the social networking applications.

Furthermore, norm emergence in digital societies can be studied using real data collected from social networking sites. New mechanisms for norm emergence can be experimented and applied on services such as file sharing on the Internet. Further investigations on norm emergence mechanisms are important because we believe that the future of the digital world will be played in digital environments where the software agents that act on behalf of human users might want to emerge norms rather than accepting an enforced norm.

References

1. Elster, J.: Social norms and economic theory. The Journal of Economic Perspectives 3(4), 99–117 (1989)
2. Axelrod, R.: An evolutionary approach to norms. The American Political Science Review 80(4), 1095–1111 (1986)
3. Shoham, Y., Tennenholtz, M.: On social laws for artificial agent societies: Off-line design. Artificial Intelligence 73(1-2), 231–252 (1995)
4. Second life, http://secondlife.com/
5. Habermas, J.: The Theory of Communicative Action: Reason and the Rationalization of Society, vol. 1. Beacon Press (1985)
6. Horne, C.: Sociological perspectives on the emergence of norms. In: Hechter, M., Opp, K.D. (eds.) Social Norms, pp. 3–34 (2001)

7. Boman, M.: Norms in artificial decision making. Artificial Intelligence and Law 7(1), 17–35 (1999)
8. Conte, R., Falcone, R., Sartor, G.: Agents and norms: How to fill the gap? Artificial Intelligence and Law 7(1), 1–15 (1999)
9. Castelfranchi, C., Conte, R.: Cognitive and social action. UCL Press, London (1995)
10. Boella, G., Torre, L., Verhagen, H.: Introduction to the special issue on normative multiagent systems. Autonomous Agents and Multi-Agent Systems 17(1), 1–10 (2008)
11. López y López, F., Márquez, A.A.: An architecture for autonomous normative agents. In: Fifth Mexican International Conference in Computer Science (ENC 2004), pp. 96–103. IEEE Computer Society, Los Alamitos (2004)
12. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: Norm-oriented programming of electronic institutions. In: Proceedings of The Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems, AAMAS, pp. 670–672. ACM Press, New York (2006)
13. Boella, G., van der Torre, L.: An architecture of a normative system: counts-as conditionals, obligations and permissions. In: Proceedings of The Fifth International Joint Conference on Autonomous Agents and Multi Agent Systems, AAMAS, pp. 229–231. ACM Press, New York (2006)
14. Boyd, R., Richerson, P.J.: Culture and the evolutionary process. University of Chicago Press, Chicago (1985)
15. Verhagen, H.: Norm Autonomous Agents. PhD thesis, Department of Computer Science, Stockholm University (2000)
16. Savarimuthu, B.T.R., Purvis, M., Cranefield, S., Purvis, M.: Mechanisms for norm emergence in multi-agent societies. In: Sixth International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2007), pp. 1097–1099 (2007)
17. Pujol, J.M.: Structure in Artificial Societies. PhD thesis, Software Department, Universitat Politénica de Catalunya (2006)
18. Savarimuthu, B.T.R., Cranefield, S., Purvis, M., Purvis, M.K.: Role Model Based Mechanism for Norm Emergence in Artificial Agent Societies. In: Sichman, J.S., Padget, J., Ossowski, S., Noriega, P. (eds.) COIN 2007. LNCS, vol. 4870, pp. 203–217. Springer, Heidelberg (2008)
19. Sen, S., Airiau, S.: Emergence of norms through social learning. In: Proceedings of Twentieth International Joint Conference on Artificial Intelligence (IJCAI), Hyderabad, India, pp. 1507–1512. MIT Press, Cambridge (2006)
20. Davidsson, P.: Agent based social simulation: a computer science view. Journal of Artificial Societies and Social Simulation 5 (2002)
21. López-Sánchez, M., Noria, X., Rodríguez, J.A., Gilbert, N.: Multi-agent based simulation of news digital markets. IJCSA 2(1), 7–14 (2005)
22. Chwe, M.: Rational Ritual: Culture, Coordination and Common Knowledge. Princeton University Press, Princeton (2001)

A Distributed Normative Infrastructure for Situated Multi-agent Organisations

Fabio Y. Okuyama[1], Rafael H. Bordini[2], and Antônio Carlos da Rocha Costa[3]

[1] Universidade Federal do Rio Grande do Sul, Brazil
okuyama@inf.ufrgs.br
[2] University of Durham, United Kingdom
R.Bordini@durham.ac.uk
[3] Universidade Católica de Pelotas, Brazil
rocha@atlas.ucpel.tche.br

Abstract. In most of the existing approaches to the design of multi-agent systems, there is no clear way in which to relate organisational and normative structures to the model of the environment where they are to be situated and operate. Our work addresses this problem by putting together, in a practical approach to developing multi-agent systems (and social simulations in particular), a high-level environment modelling language that incorporates aspects of agents, organisations, and normative structures. The paper explains in some detail how the ideas of *normative objects* and *normative places*, put together as a *distributed normative infrastructure*, allow the definition of certain kinds of *situated multi-agent organisations*, in particular organisations for multi-agent systems that operate within concrete environments. Normative objects are environment objects used to convey explicitly normative content that regulates the behaviour of agents within the place where such objects can be perceived by such agents. The paper defines these concepts, shows how they were integrated into the MAS-SOC multi-agent systems platform for social simulation, gives examples that illustrate the approach, and hints on new problems of (situated) organisational and normative structures that were brought forward by the work presented here.

1 Introduction

Multi-agent systems (MAS) are typically composed of agents, an environment, organisational structures, and means of interaction among those components. Organisational structures for multi-agent systems have been usually defined in a non-situated way, by which we mean independently of the environment where the system is to operate. In face of this issue, when a MAS is to be situated in a environment, there appears to be a 'gap' between the environment and the organisational structures, since no connection can be made between elements of the organisational structure and the physical places where such elements operate.

Furthermore, most current approaches to normative multi-agent systems [1] address the various issues on how norms can be defined, enforced, and so forth but with no clear indication on how those approaches can be used in the practical development of MAS. The work reported in this paper aims to address both these issues of state-of-the-art normative multi-agent systems.

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 29–46, 2009.

At first sight, the connection between environment and organisation could appear to be unimportant for the modelling and understanding of the system. However, as one recognises that the physical environment may influence the operation of the organisation and of the agents that work in it, one also has to recognise that the explicit connection between organisational structures and environmental structures may be of some importance for the concrete realisation of such *situated organisations*.

In particular, the connection between organisational and environmental structures and processes is important when one is dealing with organisations whose normative structure does not operate in a homogeneous way throughout the physical environment. That is, it is important for organisations where the rules that regulate the behaviour of the agents vary according to the different places where the agents are located. For instance, in a certain factory, workers that are located in an excessively noisy room may be required to work no more than two consecutive hours with a break of at least twenty minutes, while workers that are located in a less noisy room may be allowed to work more consecutive hours without a break. Also, organisations often make use of norms that are spatially and temporally bounded — that is, norms that refer to some specific places and times, and not to others. Simple behavioural regulations are often of this kind, for example in signs such as "Please keep silence" and "Do not enter after 18:00h". Moreover, locations and physical objects may be used by organisations to the *empowering* of their agents. For example, porters are given the authority to check and control anyone entering or leaving a building, and one signal of that authority is the place where they conduct their work.

In other words, in many situations organisations regulate their operation by making use of physical resources (objects and places) as means for the propagation and instantiation of norms and agent powers. Lacking an explicit connection between organisational and environmental structures represents, thus, a conceptual gap between the modelling of organisations and their realisation in concrete multi-agent systems. Often the problem is not noticed because in most approaches norms are given at a very abstract level, which then has the effect of making it difficult for norms to be implemented in practice (i.e., to be formulated in a way that agents can interpret and follow them).

It is precisely the gap between environmental and organisational/normative structures that we intend to bridge in our current work and, importantly, in such a way that can be directly implemented through a combination of our previous work on environment modelling with an agent-oriented programming framework, as well as with existing organisational models and normative languages. In particular, we have extended the MAS-SOC [2] multi-agent based simulation platform to incorporate the approach presented in this paper. Based on [12], which presents the ELMS language for environment description, and on [14], where notions of a normative infrastructure is presented, we have developed an approach for relating environmental structures to normative organisational structures of multi-agent systems, as reported in this paper.

In brief, with the extensions proposed here, the ELMS language for environment description has been extended to support *situated organisations* through *situated norms* and *situated group structures*. This was done by two means: first, we developed a *distributed normative infrastructure*, which is the structure that allows the distribution of

normative information over the spatial environment; and second, we defined a *normative principle*, associated with the design of MAS, and conceived as a special form of a conditional deontic rule, where an explicit condition on an agent's perception of a norm is included:

> Agent $\mathcal{A}$, when playing the relevant role to a norm $\mathcal{N}$ and being physically situated within the confines referred to by a normative object $\mathcal{O}$ carrying the norm $\mathcal{N}$, is expected to reason about following $\mathcal{N}$, if the agent perceived $\mathcal{O}$; otherwise, agent $\mathcal{A}$ is exempted from reasoning about $\mathcal{N}$.

In other words, the normative infrastructure is meant to provide the required elements to allow the agent to take the decision if it will follow the norms. Since our main goal is to provide means to enable the development of social simulations with cognitive agents, it is mostly the case where the norm-breaking behaviour would be desirable (to the designer of the simulation) in order to observe how the other agents (playing organisational roles) would react in such situation.

In this paper, we present the results we have achieved so far, and hint on some of the new avenues that the current work opened for further research. In Section 2, we give the background to our work, summarising the main ideas of the MAS-SOC platform, and the AgentSpeak language as interpreted by the ***Jason*** interpreter, which we use to program the agents' reasoning. In Section 3, we quickly review the environment language that we defined for describing physical environments. In Section 4, we summarise the idea of a distributed normative infrastructure for situated multi-agent organisations, introducing the concepts of *normative object* and *normative place*. In Section 5, we discuss the way norms are integrated into the MAS-SOC platform, in particular the way norms can be represented, contextualised, interpreted, and checked. Section 6 discusses various issues brought forward by our work. Section 7 discusses some related work. The conclusions of this paper are given in Section 8.

2 Background

We are developing a simulation platform called MAS-SOC (**M**ulti-**A**gent **S**imulations for the **SOC**ial Sciences). Its purpose is to provide a framework for the creation of agent-based simulations which do not require too much experience in programming from users, yet allowing users to use state-of-the-art agent technologies. In particular, it should allow for the implementation of simulations with *cognitive agents*.

In our approach, the agents' reasoning is specified in an extended version of AgentSpeak (an abstract programming language for BDI agents introduced by A. Rao [16]), as interpreted by ***Jason***, an agent platform based on Java [3].

The environment where agents are to be situated are specified in ELMS (**E**nvironment Description **L**anguage for **M**ulti-Agent **S**imulation), a language specially designed for the description of multi-agent environments. Further references about the ELMS language can be found in [13].

Until recently, the description of shared environments for situated multi-agent systems [20] had not been thoroughly addressed in the literature on agent-oriented software engineering, as the environment where agents of multi-agent systems were to be situated

used to be considered as "given" rather than an essential part of the engineering of such systems.

In [14], extensions to ELMS were made in order to introduce a distributed normative infrastructure within a (simulated) physical environment. Through the use of such infrastructure, an organisation can be integrated into the environment. However, those extensions only provide the means to situate the organisations within an environment. The concepts of *normative objects*, *normative places*, and *norm supervisors* introduced to the language are meant to bridge the above mentioned gap between environment structures and organisation structures, helping to support the proper instantiation of an organisation in an environment.

3 The ELMS Language

According to [21], agents are computational systems situated in some environment, and are capable of autonomous action in this environment in order to meet their design objectives. Therefore, the environment has an important role in the specification, design and implementation of a MAS, whether it is the Internet, the real world, or some simulated environment. Even when the environment of the multi-agent system is the "real world" and the agent is embodied in a robot with sensors and effectors, the environment model should play a significant role in the design of the system. Any robot should have a set of sensors that give a predefined set of percepts that the robot will acquire when sensing the environment. Also, it should have a set of effectors that allow a restricted set of (parameterisable) actions. Thus, the possible sensor inputs and effectors output should be modelled first to facilitate the development of the software for the robot.

We understand by *environment modelling*, the modelling of all of the external world that an agent needs to know about for reasoning and deciding on courses of action. Agents themselves should also be considered components of the environment insofar as, from the point of view of an agent, any other agent is also part of the environment.

Thus, to define agents from this point of view, it is necessary to include in the description of the environment all properties that define the aspects of the "body" of each agent, which represent the properties of an agent that are perceptible to other agents. Furthermore, it is necessary to model explicitly the physical actions that agents are allowed to perform and the perception capabilities they have in their environment.

Through the definition of actions and perceptions, we define the *physical rules* of the environment (e.g., no one may see or pass through a wall) that must be satisfied in order to ensure the meaningfulness of the system model. This is in contrast to the norms of a organisational structure, which an agent may reason upon and decide to breach, with no implication to the meaningfulness of the system.

The objects that are part of an environment can be modelled as a set of properties and a set of reactions that characterise the behaviour of such objects in response to stimuli. They are treated as *reactive* components; only agents are pro-active, through their reasoning and deliberation processes.

From the point of view of the ELMS language, the deliberative activities of an agent are not relevant and are left out of the description of the environment, since they are internal to the agents (i.e., they are not physically perceived by the other agents). As

mentioned before, in the MAS-SOC platform, the mental aspects of agents are described in the AgentSpeak language.

Below, we briefly review how an environment is described using this language. In section 3.1 we describe the modelling of agent "bodies", and in Section 3.2 the physical environment.

3.1 Modelling Agent Bodies

In the environment description, agents are characterised through the definition of classes of *agent bodies*, agent sensorial capabilities, and agent effective capacities.

Agent Body: The agent's characteristics that are perceptible to other agents. In our approach, classes of agent *bodies* are defined by a set of properties that characterise them and are perceptible to other agents. Each *body* is associated with a set of actions that it is allowed to perform and a set of environment properties that it can perceive.

Agent Sensorial Capabilities (Percepts): Each sensorial capability is used to specify which environment properties will be perceptible to each agent that has a "body" with such capacity. It determines the environmental properties that will be sensed by the agent and the specific circumstances where they are possible. If the preconditions are all satisfied, then the values of those properties will be made available to the agent's reasoning procedure as the result of the agent's perception of the environment.

Agent Effective Capacities (Actions): These are the capabilities of performing actions in the environment that are made available to each class of agent body. Each definition of an action determines the environmental changes that such action can make when performed by an agent that has a body with such capacities. These changes are defined as assignments of values to the attributes of the environment. As the choice on (courses of) actions is meant to be part of the agent's "mind", something that is more naturally seen as a whole series of actions should not be implemented as one action available to agents at the environment level.

3.2 Environment Modelling

The environment is modelled by the definition of the objects available in the environment, the reactions that such objects might produce, and the spatial representation of the environment. Each of these form specific language constructs classified as follows.

Physical Environment Objects: The objects that are present in the environment. Agents interact with objects through the actions they perform in the environment, and by perceiving them. Object structures are defined by a set of properties that are relevant to the modelling and that could potentially be perceived by agents. The reactions that a class of objects can have is given by a list of identifying labels.

Object Reactions: The objects can "react", under specific circumstances, responding to actions performed by the agents in the environment.

Space Representation: The spatial representation of the environment can be done as a grid or a graph, according to the requirements of a particular simulation or design preferences. When a grid is used, the space is divided into cells forming a grid that

represents the spatial structure of the environment, either in 2 or 3 dimensions. As for resources, each grid cell can have reactions associated with them. When a graph is used, the space is represented by *nodes* connected by *links*. As the cells of the grid, each node of the graph represents a spatial location, where reactions may occur in response to some action performed on it. The links represent connections between places that agents can follow, and may have weights or values associated with them, according to the needs of each particular project. In both grid- and graph-base spaces, the granularity of the spatial representation should follow the requirements of the application.

4 Normative Infrastructure

Certain real environments have objects aimed at informing "agents" about norms, give some advice, or warn about potential dangers. For example, a poster fixed on a wall in a library asking for *silence* is an object in the environment, but also informs about a norm that should be respected within that space. The existence of such signs, which we call *normative objects*, implies the existence of a regulating code in such context, which we call *situated norm*.

Situated norms are only meant to be followed within certain boundaries of space or time, and lose their effect completely if those space and time restrictions are not met. Another important advantage of modelling some norms as situated norms is the fact that the spatial and temporal context where the norm is to be followed is immediately determinable. Thus, the norm can be "pre-compiled" to its situated form, making it easier for the agents to operationalise the norm, and also facilitating the verification of norm compliance.

The *normative infrastructure* is intended to provide a means to inform the agents about the norms in a specific spatial context, allowing the agents to reason about norms of which they may have no previous knowledge. Also, the normative infrastructure has no specific enforcing nature, as the agents might not perceive (or may pretend to have not perceived) some of the normative objects, when they should.

In this section we present the extensions to ELMS that are meant to provide an infrastructure allowing the distribution of normative information within an environment. We refer as *normative infrastructure*, the concepts introduced in ELMS in order to allow the distribution of norms over the environment, while we use *normative structure* to designate a structured set of norms, some of which may be expressed by instances of normative objects and places, thereby regulating some agent organisation.

4.1 Normative Objects

Normative objects are "readable" by agents under specific individual conditions; that is, an agent can read a specific rule if it has the ability to perceive that type of object, at the location where the object is placed in the environment model. In the most typical case, the condition is simply being physically close to the object.

Such objects can be defined before the simulation starts, or can be created dynamically during the simulation. Each normative object can be placed in a collection of cells/nodes of the spatial representation of the environment. For example, a cell or group of cells of an environment grid can be used to represent a *normative place*, determining

the first condition for the normative object being perceived: it is only within that normative place that the content of the normative object is relevant. The conditions under which the normative objects can be perceived are defined by the simulation designer using the constructs for defining perception conditions.

The normative information in a normative object is "read" by an agent through its usual sensing/perceptual abilities. It contains the norm itself and also meta-information, as follows:

Id: Identification string, for the management of the normative object within the system.

Norm: A string that represents the normative information; this can be in any format that the targeted agents are be able to understand — for instance, AgentSpeak terms in the case of ELMS environments in the MAS-SOC platform. However, to enforce a uniform norm specification format over all applications, a fixed format should be adopted. For practical reasons, we have chosen the policy language REI [11] for such purpose.

Type: The type of the normative information contained in the object; it determines the level of importance (e.g., a warning, an obligation, a direction);

Issued by: Agent or group that issued the norm;

Source: Where the power underlying the norm issuance comes from; this could be the role that was being performed by the agent when issuing the norm, and the organisation (or group) that endorses such role;

Addressees: The organisational components (agent groups and agent roles) to which the normative information applies.

Placement: The set of normative places where the normative information applies. If omitted, the object is assumed to be valid everywhere in the environment, but normally only under the specific conditions determined by the designer (see the next item).

Condition: Conditions under which the normative information can be perceived. The conditions can be associated with physical location, time, perception capabilities, spatial orientation of agents and objects, etc.

It is worth noting that norm-abiding behaviour is not related just to the existence of a normative object at some place. Beyond the existence of such object, it is necessary for the agent to physically perceive the normative object. Besides, autonomous agents will also reason about whether to follow or not the norm stated by the normative object, if and when they perceived it. This suggests that various specific problems should be tackled, concerning the study of agent reasoning about *situated normative objects*, a topic we discuss in Section 6.

4.2 Normative Places

Normative places are abstractions to define the boundaries of spatial locations where a set of related activities are done, or where groups of agents interact, and where some specific norms are valid and relevant. These places are also the physical spaces where the components of an organisational structure are located; that is, a *normative place* constitutes the spatial scope of an organisation, as well as the norms related to that

organisation. The relevant normative information for each place is usually stored there, through the use of *normative objects*.

A normative place is defined simply by an identification label (a name) and the specification of its spatial boundaries, which is defined by the set of cells of the grid that are part of it (or, the nodes of the graph, according to the spatial representation being used). For each normative place, a set of *local roles* is defined to be located at such place, so that the roles that are present in such spatial context are regulated through norms embedded in the *normative objects* that are placed in that space.

A normative place may have intersections with other normative places, or may even be contained by another. For example, a "school" may be seen as a normative place encompassing a large portion of the environment grid where some of those grid cells refer to a normative place "classroom" and others to a normative place "library".

The area covered by a normative place may increase or decrease during a simulation, since we are dealing with possibly dynamic environments, which may be associated with possibly dynamic organisations. Thus, the influence area of an organisation may expand or reduce dynamically, according to the requirements of the application, by changing the set of cells or nodes defined to belong to such normative place, which can be done by an agent empowered to effect such change. Such changes may occur under two circumstances: first, when the organisation deliberately rearranges the area where it needs to influence agent behaviour; and second, when the organisation acknowledges that the agent behaviour prescribed in a particular place has become more widely practiced by the agents themselves, so the organisation changes its area of operation (a normative structure) to reflect the actual (emergent) agent behaviour.

Similarly, different social behaviour might emerge if we rearrange the distribution of normative objects within a normative place where a particular organisation is situated, or if we create new normative objects. Clearly, these situations appear in many social situations, and having high-level abstractions available to model such situations can greatly facilitate the development of social simulations.

5 Using Norms

5.1 Norm Contextualisation

Normative objects are not supposed to be means of broadcasting general norms. The norms informed through normative objects should be *contextualised* (by the system designer or the agent that created the norm), incorporating specific information about the normative place where it is relevant.

As the spatial context of the norm is bounded and determined by its normative place, a generic abstract norm can be "pre-compiled" using such information, in order to make it less abstract. This process is meant to facilitate norm operationalisation, as such concrete norms are "ready to use" in the spatial scope where it is relevant. Other advantages of having less abstract norms are that the verification of norm compliance is facilitated and that they can reduce misinterpretations that could occur with abstract, non-contextualised norms.

For example, a norm that says "be kind to the elderly" can be quite hard to operationalise and verify, in general. However, in a fixed spatial context, such as a bus or

a train, with the norm contextualised as "give up your seat for the elderly", or in a street crossing with the norm contextualised as "help elderly people cross the street", the norm would be much easier for the agents to interpret, and easier to verify using any norm-compliance checking mechanism.

5.2 Policy Language

In the MAS-SOC platform, the norms contained in the normative objects can be expressed simply as valid AgentSpeak predicates. In order to represent the norms in a uniform way in all simulations, we have adopted the policy language REI [11]. REI is a policy language aimed at the definition of policies for pervasive computing, being well-suited to our platform, as the pervasive computing paradigm is very close to the notion of having different normative places in a physical environment. In the REI language, there are constructs to define rights, prohibition, obligations, and dispensations. Also, the language has many other constructs, which include specific ones to solve conflicts, action specification, and rights delegation. A detailed description of the language can be found at `http://www.cs.umbc.edu/~lkagal1/rei`. Below, we present some of the main constructs available in the REI language, summarised from [11].

Rights, prohibitions, obligations, and dispensations:

`has(agent_ag, right(action_act, Conds)):` means that agent *agent_ag* has the right of executing action *action_act* if expression *Conds* is satisfied. The notion of *right* or *permission* can be interpreted as the deontic expression $\sim$O$\sim$.

`has(agent_ag, prohibition(action_act, Conds)):` means that agent *agent_ag* is prohibited from executing action *action_act* if expression *Conds* is satisfied. The notion of prohibition can be interpreted as the deontic expression O$\sim$.

`has(agent_ag, obligation(action_act, Conds)):` means that agent *agent_ag* is obliged to execute action *action_act* if expression *Conds* is satisfied. The notion of obligation can be interpreted as the deontic operator O.

`has(agent_ag, dispensation(action_act, Conds)):` means that agent *agent_ag* is dispensed from executing action *action_act* if expression *Conds* is satisfied. The notion of dispensation can be interpreted as the deontic expression $\sim$O.

Definition of priorities:

`overrides(A1,B1):` means that rule `A1` overrides rule `B1`.

5.3 Library of Norm-Considering Plans

In order to facilitate the programming of normative agents, we developed plans to deal with the reasoning and deliberation about certain kinds of norms. Those plans are organised in files that can be imported from another file by the use of the `include` directive in ***Jason***. Since such plans are available as plain files, it is also possible to use them as templates to build customised plans according to the requirements of individual projects.

For example, in order to program an agent that never violates a prohibition to execute an action `a`, one should replace in its program, every occurrence of `a` by `!execute(a)`, and also include the following plans in the agent's plan library:

```
+!execute(Action)
      : not prohibition(Action,_)
      <- Action.

+!execute(Action)
      : prohibition(Action,Condition)
        & not Condition
      <- Action.
+!execute(Action)
      : prohibition(Action,Condition)
        & Condition
    <- .fail.
```

As another example, to program an agent that always accomplishes an obligation determined by an organisation it trusts, unless the agent turns out to be dispensed from it (or, of course, if it violates some prohibition), the following plans can be used:

```
+has(Self,obligation(Action,Condition))[sourceOrg(SO)]
    :  .my_name(Self) & trusted(SO) & Condition
    <- !checkDispensation(Action);

+!checkDispensation(Action)
    :  .my_name(Self)
       & has(Self,dispensation(Action,Condition))
       & not Condition
    <- !execute(Action).

+!checkDispensation(Action).
```

In the code above, to handle the event that occurred because a new obligation was perceived, the agent checks whether the organisation that endorses the norm is trusted; if so, it checks the conditions of the obligation, and then it checks if there is a dispensation for such obligation; if there is none, it will execute the action, after checking if there is no prohibition on such action, as usual. The AgentSpeak code above is used to give priority to prohibition: a prohibited action is never executed. However, priorities among norms can be easily changed. In the code above, replacing `!execute(Action)` by `Action` would cause the agent to give priority on the obligation over the prohibition.

A topic of research in normative multi-agent systems is precisely the definition of general reasoning procedures to cope with various sorts of normative situations, such as when an action is both desired and prohibited [1]. We have not yet addressed some of these issues, in particular those that are controversial and still being debated at the theoretical level.

6 Issues in Distributing Norms

6.1 Norm Monitoring

In our approach, we define a special class of agents, called *norm supervisors*, which monitor other agents' compliance to norms within an organisation. Since agents are free to reason about abiding or not to a norm stated in a normative object, there is also the need

to monitor the behaviour of those agents, at least in some applications. In order to be able to act as a norm supervisor, an agent may need extra information and perhaps extra capabilities. For this reason, it is possible to define, in ELMS, an agent as a *norm supervisor*, which will enable it to receive information about the relevant normative structure as well as about the actions being done by other agents in a given normative place.

Agents in charge of norm supervision could be agents that are not part of the actual simulation being conducted, designed specially to check agents' compliance to norms, or could be "normal" agents that belong to the simulation, and whose interests require that certain other agents follow certain norms. As the norm and the possible violations are confined to a specific normative place, it is potentially easier to identify the possible violations of those norms. The simulation designer may want to enable such capacity in a agent just to help it achieving its goal, to use such information to monitor/debug the simulation, or as an input to a reputation system, among other things.

For instance, according to [5], an agent may be motivated to verify the compliance to norms by other agents in order to reassure itself that the costs of norm adherence is being paid by the other agents too. A norm abiding agent will want that all the other addressees of the norms follow it too, otherwise the norm adhering behaviour may become some sort of competitive disadvantage. In [5], the authors refer to agents with such behaviour as "norm defenders".

6.2 Organisations and Environments

In most of the existing approaches for multi-agent organisations, such as MOISE$^+$ [10], the organisational structures are connected to the agent's reasoning by the implementation or through communication messages. Our work is not intended to replace such connection. In fact, we aim to intensify the interactions of agents and organisations, by having both direct an indirect interactions. The connection of an organisation to an environment, in our approach, can be done essentially in two ways:

Static: As shown in the left-hand side of Figure 1, from an (external) organisation description, the designer can model the normative structure to reflect a static image of an organisation, converting the organisational structures into roles and organisational

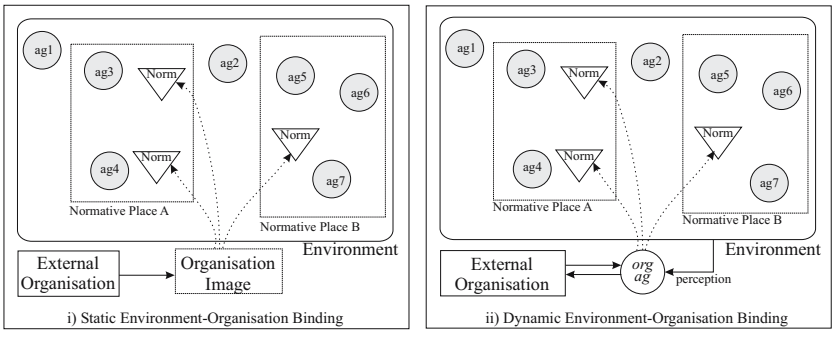

Fig. 1. Organisation and Environment Binding

links. The roles are attached to the normative places while the organisational behaviours and links are prescribed by norms included in normative objects.

Dynamic: As shown in the right-hand side of Figure 1, agent '*org ag*' obtains the information available in the organisation description and dynamically changes the normative structure in the environment. An agent, when receiving the percepts from the environment, may use this information as feedback to the organisational engine, which may change the organisation. Agent *'org ag'* may take part in the simulation or not, according to the requirements of the application.

It seems that, by using either static or dynamic binding as described above, it is possible to integrate an environment definition with most of the existing approaches to multi-agent systems organisations such as [18, 10, 8, 6]. However, simplifications may be required, and certain features of some approaches to agent organisations may not be captured by the use of such integration.

6.3 Implicit Role Adoption

For each normative place, a set of *local roles* that are regulated by the normative objects present in such place can be defined. In each spatially and temporally bounded normative place, an agent may adopt such temporary roles according to the activity it is doing in that place. The adoption of such roles may happen in an explicit or implicit manner.

An agent may have to explicitly identify itself to the other agents or institutions in order to adopt a specific role. For example, in a website that offers banking transactions, an agent may browse anonymously the public sections, but if it wants access to private information about its account, it must authenticate itself (e.g., with a username and password), explicitly adopting the role of an "identified user", acquiring access to the specific rights of that role.

Using special elements (spatial positioning, orientation, possession of certain objects, agents' roles in organisational structures, etc.) an implicit role adoption may happen, which can be defined for each normative place with the use of simple rules. Below, we give some examples (using pseudocode) of how this can be done:

Default role: a default role may be defined for each place; for example, the *user* role, in a library:

```
default -> agent.role = user
```

Possession of an object: an agent may hold an object that associates it to a role; for example an agent carrying a badge may be assigned the role of *staff* in a library:

```
agent.hasBadge -> agent.role in library = staff
```

Positioning: according to its position, an agent may have a specific role; for example an agent in a moving car, seated at the driver's seat, will be assigned the role of *driver*:

```
agent in car AND agent.position = driver-position
 -> agent.role in car = driver
```

Relative role: a role in an organisation can be associated to the role performed in another organisation:

```
agent.role in university1 = researcher
AND agent at university2
 -> agent.role in university2 = visiting-researcher
```

6.4 Perception-Bounded Norm Reasoning

Given that a norm published through a normative object is only accessible as far as the normative object itself is accessible (i.e., perceptible), the normative reasoning concerning that norm is bounded by the boundaries of perception of the normative object. That is, when an agent did not follow a norm that was supposed to be followed at a given normative place, at least three different types of reason could explain that fact: (i) the agent really did not perceive the normative object; (ii) the agent perceived the object but not carefully enough to be able to grasp its normative content; and finally (iii) the agent correctly perceived the object and its normative content, but decided not to follow the norm.

The problem of norm abiding in normative situations, based on normative objects, then, has to take into account not only the possibility that agents autonomously decide to follow or not to follow the norms, but also the possibility that agents are not able to correctly perceive the normative objects. Issues such as responsibility, and others related to norm abiding, incorporate thus not only the usual aspects of rationality and affectivity, but also issues related to physical perception in concrete environments. Our approach, by bridging the gap between environment, organisations, and normative systems, has highlighted such issues.

6.5 Distribution of Normative Objects

Issues related to the adequacy of the distribution of normative objects in a normative environment also arise when dealing with the kind of distributed normative infrastructure that we are proposing. That is, guaranteeing that the set of normative objects is well distributed, and distributed in a satisfactory way, is an issue that should concern the agents responsible for issuing normative objects or the system designer.

Moreover, regulation of normative exceptions is also often made with normative objects (e.g., “It is forbidden to smoke in this building, except in the areas marked with a ‘Smoking allowed’ sign”). Thus, guaranteeing that the hierarchical structure of norms operating at a given normative place is well accessible and understandable to the agents is also an issue for the regulating agents.

6.6 Modular Design of Normative Places

Using the abstractions of *normative place* and *local roles*, the roles referred to in each normative place are strictly relative to the functions being performed in such place, which may have no direct relation to existing roles in other parts of the organisation. For example, consider a city as a multi-agent system. In a normative place “street”, an agent driving a car will be playing the “driver” role, and another one not driving a car

will be playing the "pedestrian" role — this is regardless of whether they are a school teacher or a hospital nurse in other places.

In such case, from the point of view of the design of such environment portion and normative structure, the agents are simply moving from one place to another, irrespective of the various other roles that they may be playing for other organisations within the city. The partitioning the environment into *normative places*, which can be done in a modular way, reducing the interdependence of each part, facilitates the modelling of large-scale simulations.

7 Related Work and Discussion

The notion of artifacts [19] and coordination artifacts [15] resembles, in some aspects, our notion of *normative objects*. However, they are clearly a different concept. Coordination artifacts are defined as runtime abstractions that encapsulate and provide a coordination service to the agents, but they express normative rules only implicitly, through their automatic effects on the actions of the agents. So their impositive impact on agent behaviour dismisses any need for normative reasoning on the part of the agents. In fact, coordination artifacts preempt the agents' freedom to overlook social norms and to decide not to follow them. In our work, rather than having a notion of objects that by their (physical) properties facilitate coordination, normative objects are used to store *symbolic* information that can be interpreted by agents, so that they can become aware of norms that should be followed within a well-defined location. Even if the general notion of artifact is similar to ELMS objects and could also contain symbolic normative information, one advantage of our approach is that it allows for a declarative language in which to represent the environment, which is executed by the interpreter to simulate the environment, whereas the existing implementations of artifacts provide a Java API with which to program the environment model.

Our choice in regards to normative objects has the advantage of keeping open the possibility of agent autonomy, as suggested in [4]. Agents are, in principle, able to decide whether to follow the norms or not when pursuing goals. Another important aspect is that normative objects are spatially distributed over a physical environment, with a spatial scope where they apply, and closely tied to the part of the organisation that is physically located in that space. Our work simplifies the way designers can guide the behaviour of each individual agent as they move around an environment where organisations are spatially located; this allows agents to adapt the way they behave in different social contexts.

The AGRE model, presented in [6], allows the definition of structures that represent the physical space, as "specialisations" of a generic space. However, we find that the social structures are not contextualised as they are in our work, leaving the social and physical structures rather unrelated.

Another important series of related work is that on Electronic Institutions [8]. The internal workings of an electronic institutions is given (in rough terms) as a state-machine where each state is called a "scene". Each scene specifies the set of roles that agents can perform in it, and a "conversation protocol" that the agents should follow when interacting within a scene. To traverse the series of scenes of the electronic institution,

agents must do a sequence of actions in each scene, and also commit to certain actions in certain scenes, as the result of having performed certain other actions in certain other scenes. Our notion of normative place was inspired by such notion of scene, in giving it a physical, spatial reference where norms apply.

Similar to the electronic institutions approach, Computational Institutions [17] are defined as virtual organisations ruled by constitutive norms and regulative norms. In such institutions, organisational modelling uses coordination artifacts as building blocks, in a way that is very similar to our use of normative objects in spatially distributed organisations, but still keeping implicit in coordination artifacts the normative content imposed on the agents.

In [9], the notion of *governor* agents was introduced. Such agents aim to ensure that external agents fulfil all their social duties during the enactment of an electronic institution. The external agents interact with the environment through the governor agents, which also inform them about norms and possible actions. Governor agents differ from the *norm supervisors* presented here, as norm supervisors do not have as primary objective to avoid norm infringement, while governors aim to prevent external agents from performing actions that are not permitted in an institution.

In the research on normative multi-agent systems, various aspects of normative systems are discussed which are not directly dealt with in our model as yet, such as an explicit model of sanctions and norm enforcement, and a formal basis for norm representation and reasoning [1, 22, 7]. The advantage of our approach, however, is that is can be directly used in the development of practical multi-agent systems. We plan to incorporate increasingly sophisticated aspects of normative systems into our framework in the future.

8 Conclusions

We have presented an approach to integrate the modelling of environments and organisations, using a normative infrastructure that provides the means to distribute normative information over an environment. Such infrastructure, composed of *normative objects* and *normative places*, allows the spatial contextualisation of norms. The contextualisation of norms in a bounded spatial/temporal scope facilitates the operationalisation of the norms and the verification of compliance, and helps avoiding the misinterpretation of norms. Also, in our approach, a normative structure is a connection point relating environments and organisations, being a reflection of the organisation on its environment.

The distribution of norms over the environment, using normative objects, allows the environment to be partitioned in a modular way. Such partitioning facilitates an independent modelling of each part of the system, reducing the interdependence among the various parts, thus facilitating the modular modelling of the environment and organisations, taking advantage of the natural distribution of certain environments, with norms being associated only with the places where they should be followed, instead of requiring a central repository of norms. Therefore, the proposed normative infrastructure facilitates the design, development, and maintenance of large-scale multi-agent systems or simulations. Besides, this should pose less of a burden on agents that dynamically access normative information compared to approaches where all norms are made centrally available.

We believe that an explicit environment description is an important part of a multi-agent system, as thoroughly discussed in the literature on approaches to modelling multi-agent environments. Also, environment modelling facilitates the engineering of multi-agent systems as it is a stable point from where the agent reasoning and the organisational structures can be tuned to facilitate the development of agents and organisations. The notion of spatially distributed normative objects that we have introduced seems to serve well the integration of organisations and environments.

As future work, we plan not only to develop simulations using our platform, so as to further evaluate it and improve our approach, but also to study interesting issues related to the use of this approach. One issue to be investigated in future work is that having the norms spread over many independent spatial scopes may result in different reputations of a single agent over the environment, leading to a notion of *locality of reputation*. Another interesting aspect is that, being conditioned on the possibility of perceiving the existence of a normative object, the reasoning of agents that deal with normative objects is necessarily of a non-monotonic nature.

In summary, the normative reasoning required by the possibility of having normative places within the environment, each one with with its own organisational purposes and sets of norms, leads to many issues to be addressed in the future.

Acknowledgements

This work was partially supported by CNPq and FAPERGS.

References

1. Boella, G., van der Torre, L., Verhagen, H.: Introduction to normative multiagent systems. In: Boella, G., van der Torre, L., Verhagen, H. (eds.) Normative Multi-agent Systems, number 07122 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2007), `http://drops.dagstuhl.de/opus/volltexte/2007/918` [date of citation: 2007-01-01]
2. Bordini, R.H., Costa, A.C.d.R., Hübner, J.F., Moreira, A.F., Okuyama, F.Y., Vieira, R.: MASSOC: a social simulation platform based on agent-oriented programming. Journal of Artificial Societies and Social Simulation 8(3) (2005)
3. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using Jason. Wiley Series in Agent Technology. John Wiley & Sons, Chichester (2007)
4. 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)
5. Conte, R., Castelfranchi, C.: Cognitive and Social Action. UCL Press, London (1995)
6. Ferber, J., Michel, F., Baez, J.: AGRE: Integrating Environments with Organizations. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2004. LNCS, vol. 3374, pp. 48–56. Springer, Heidelberg (2005)

7. Fornara, N., Colombetti, M.: Specifying and enforcing norms in artificial institutions. In: Boella, G., van der Torre, L., Verhagen, H. (eds.) Normative Multi-agent Systems, number 07122 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2007), http://drops.dagstuhl.de/opus/volltexte/2007/909 [date of citation: 2007-01-01]
8. García-Camino, A., Noriega, P., Rodríguez-Aguilar, J.A.: Implementing norms in electronic institutions. In: Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M.P., Wooldridge, M. (eds.) AAMAS 2005: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 667–673. ACM Press, New York (2005)
9. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.W.: A distributed architecture for norm-aware agent societies. In: Baldoni, M., Endriss, U., Omicini, A., Torroni, P. (eds.) DALT 2005. LNCS, vol. 3904, pp. 89–105. Springer, Heidelberg (2006)
10. Hübner, J.F., Sichman, J.S., Boissier, O.: $\mathcal{M}$OISE$^+$: Towards a structural, functional, and deontic model for MAS organization. In: AAMAS 2002: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems. ACM Press, New York (2002)
11. Kagal, L., Finin, T.W., Joshi, A.: A policy language for a pervasive computing environment. In: 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), Lake Como, Italy, June 4-6, 2003, pp. 63–74. IEEE Computer Society, Los Alamitos (2003)
12. Okuyama, F.Y., Bordini, R.H., da Rocha Costa, A.C.: ELMS: An Environment Description Language For Multi-Agent Simulation. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2004. LNCS, vol. 3374, pp. 91–108. Springer, Heidelberg (2005)
13. Okuyama, F.Y., Bordini, R.H., da Rocha Costa, A.C.: Spatially Distributed Normative Infrastructure. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2006. LNCS, vol. 4389, pp. 203–220. Springer, Heidelberg (2007)
14. Okuyama, F.Y., Bordini, R.H., da Rocha Costa, A.C.: Spatially Distributed Normative Objects. In: Noriega, P., Vázquez-Salceda, J., Boella, G., Boissier, O., Dignum, V., Fornara, N., Matson, E. (eds.) COIN 2006. LNCS, vol. 4386, pp. 133–146. Springer, Heidelberg (2007)
15. Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., Tummolini, L.: Coordination artifacts: Environment-based coordination for intelligent agents. In: AAMAS 2004: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, Washington, DC, USA, pp. 286–293. IEEE Computer Society, Los Alamitos (2004)
16. Rao, A.S.: Agentspeak(l): Bdi agents speak out in a logical computable language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
17. Rubino, R., Omicini, A., Denti, E.: Computational institutions for modelling norm-regulated MAS: An approach based on coordination artifacts. In: Lindemann, G., Ossowski, S., Padget, J., Vazquez-Salceda, J. (eds.) 1st International Workshop Agents, Norms and Institutions for Regulated Multi-Agent Systems (ANI@REM 2005), AAMAS 2005, Utrecht, The Netherlands, July 25 (2005)
18. Vázquez-Salceda, J., Dignum, V., Dignum, F.: Organizing multiagent systems. Autonomous Agents and Multi-Agent Systems 11(3), 307–360 (2005)
19. Viroli, M., Omicini, A., Ricci, A.: Engineering MAS environment with artifacts. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2005. LNCS, vol. 3830, pp. 62–77. Springer, Heidelberg (2006)
20. Weyns, D., Van Dyke Parunak, H., Michel, F., Holvoet, T., Ferber, J.: Environments For Multiagent Systems State-Of-The-Art And Research Challenges. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2004. LNCS, vol. 3374, pp. 1–47. Springer, Heidelberg (2005)

21. Wooldridge, M.: Intelligent agents. In: Weiß, G. (ed.) Multiagent Systems—A Modern Approach to Distributed Artificial Intelligence, ch. 1, pp. 27–77. MIT Press, Cambridge (1999)
22. Lopez, F.L.y., Luck, M., d'Inverno, M.: A normative framework for agent-based systems. In: Boella, G., van der Torre, L., Verhagen, H. (eds.) Normative Multi-agent Systems, number 07122 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany (2007), http://drops.dagstuhl.de/opus/volltexte/2007/933 [date of citation: 2007-01-01]

A Complete STIT Logic for Knowledge and Action, and Some of Its Applications

Jan Broersen

Department of Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
broersen@cs.uu.nl

Abstract. This paper presents a complete temporal STIT logic for reasoning about multi-agency. I discuss its application for reasoning about norms, knowledge, autonomy, and other multi-agent concepts. Also I give some arguments in favor of taking STIT formalisms instead of dynamic logics as the basis for logics for representing multi-agent system notions.

1 Introduction

The acronym 'STIT' stands for 'Seeing To It That', and STIT logics are philosophical logics of agency. Recently these logics have attracted the attention of computer scientist who aim at using STIT formalisms to model and reason about multi-agent systems [11,20,12,13]. The present paper takes several new steps in this line of research. First of all a new semantics is presented. Key features of the semantics are its two-dimensional structure and the circumstance that STIT actions only take effect in successor states. Second, the present logic encompasses reasonable axioms for the interaction of the next-time operator and the STIT-operators. This solves one of the weaknesses of the logic in [10] where there is no interaction between the time and the action dimensions. Third, new principles for the interaction between knowledge and action are proposed. I show how the combination of knowledge and STIT operators can be used to represent a notion of 'knowingly doing'. Several new and fascinating questions arise by introducing this notion. For instance, the concept presupposes that things can also be done 'unknowingly' or 'unaware'. Finally, the paper not only presents the formal semantics and a complete axiomatization, but also discusses the applicability of the proposed epistemic STIT logic to the modeling of several key multi-agent system concepts.

2 A Temporal Epistemic STIT Logic

In this section 1 define a complete STIT logic with operators for knowledge. Knowledge operators were first introduced in the STIT framework in [20] the ideas of which were further developed and generalized in [13] and [10]. The distinguishing feature of the present STIT logic is that actions only take effect in

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 47–59, 2009.

'next' states, where 'next' refers to immediate successors of the present state. This distinguishes the present STIT logic not only from the STIT variants in the above mentioned papers, but also from any STIT-logic in the (philosophical) literature. However, there are very good reasons for taking this approach. The first reason is that it can be shown (see [4]) that the logics of the multi-agent versions of, what we might call, the standard 'instantaneous' STIT, are undecidable. The second reason is that the view that actions only take effect in some immediate next state, is the standard view in formal models of computation in computer science. And finally, also from an ontological perspective, the choice is defendable. Given that an action can always be thought of as a 'process' connected to some effort of the agent involved, and given that processes can suitably thought of as occurring 'in' time, we may conclude that also actions take place 'in' time.

The present paper also further adapts and develops ideas from [20,13,10] by suggesting new properties and corresponding axioms for the interaction between time and action, and between action and knowledge, and by giving a two-dimensional modal semantics. The semantics has two strong advantages. As compared to the semantics in [10] it is much closer in spirit to the branching time STIT semantics known from the philosophical literature, while at the same time it is completely 'standard' from a modal logic perspective.

Besides the usual propositional connectives, the syntax of the logic comprises an operator $K_a\varphi$ for knowledge of individual agents a, an operator $\Box\varphi$ for historical necessity, which plays the same role as the well-known path quantifiers in logics such as CTL and CTL^* [16], and finally, an operator $[A\ \mathsf{xstit}]\varphi$ for 'agents A jointly see to it that φ in the (immediate) next state'.

Definition 1. *Given a countable set of propositions P and $p \in P$, and given a finite set Ags of agent names, and $a \in Ags$ and $A \subseteq Ags$, formally the language can be described as:*

$$\varphi, \psi, \ldots := p \mid \neg\varphi \mid \varphi \wedge \psi \mid K_a\varphi \mid \Box\varphi \mid [A\ \mathit{xstit}]\varphi$$

I define operators for 'next' $X\varphi$, and several operators for obligation as abbreviations in the language. In this section 1 only give the definition for the 'next' because the explanation of the definition of the obligation operators can better be done after the formal semantics of the base operators is given. I define the 'next' operator as the current action performed by the complete set of agents Ags:

Definition 2

$$X\varphi \equiv_{def} [Ags\ \mathit{xstit}]\varphi$$

The view that the complete set of agents uniquely determines the next state is a common one. Not only it can be found in the multi-agent STIT logics of Horty [21], but also in related computer science formalisms such as ATL [1,2]. For the relation between STIT formalisms and computer science formalisms such as ATL and Coalition Logic [23], see [11,12].

Before I give the formal definitions for the frames and the models, I briefly elaborate on what these structures represent. The frames are two-dimensional, with a dimension of 'histories' which are thought of as linear time-lines coming from the past and extending into the future, and a dimension of 'states' which are possible states the system of agents can be in. Given any particular history, the next time relation relates states along that history. Given any particular state, the historical necessity relation relates all histories associated with that state. Effectivity relations[1] relate history/state pairs to sets of possible next history/state pairs: the pairs the actual situation is ensured to be among next, if the choice is taken. Behaviors of the system of agents can be seen as trajectories through the two dimensional space going from the past to the future along the dimension of states, and jumping from sets of histories to sub-sets of histories (the choices) along the dimension of histories.

Definition 3. *A frame is a tuple* $\mathcal{F} = \langle H, S, R_\Box, \{R_A \mid A \subseteq Ags\}, \{\sim_a \mid a \in Ags\}\rangle$ *such that:*

- H *is a non-empty set of histories. Elements of* H *are denoted* h, h', *etc.*
- S *is a non-empty set of states. Elements of* S *are denoted* s, s', *etc.*
- $R_\Box$ *is a 'historical necessity' relation over the elements of* $H \times S$ *such that* $\langle h, s\rangle R_\Box \langle h', s'\rangle$ *if and only if* $s = s'$
- *The* R_A *are 'effectivity' relations over the elements of* $H \times S$ *such that:*
  - R_{Ags} *is a 'next time' relation such that if* $\langle h, s\rangle R_{Ags}\langle h', s'\rangle$ *then* $h = h'$, *and* R_{Ags} *is serial and deterministic (the next state is completely determined by the choice made by the complete set of agents). So, histories 'contain' linearly ordered sets of states.*
  - $R_\Box \circ R_{Ags} \subseteq R_\emptyset$ *(the empty set of agents is ineffective)*
  - $R_A \subseteq R_\Box \circ R_{Ags}$ *for any* A *(an action undertaken by* A *in the present state ensures the next state is element of a specific subset of all possible next states)*
  - $R_{Ags} \circ R_\Box \subseteq R_A$ *for any* A *(no actions constitute a choice between histories that are undivided in next states)*
  - $R_A \subseteq R_B$ *for* $B \subset A$ *(super-groups are at least as effective)*
  - *if* $\langle h, s\rangle R_\Box \langle h', s\rangle$ *and* $\langle h, s\rangle R_\Box \langle h'', s\rangle$ *then there is a* $\langle h, s\rangle R_\Box \langle h''', s\rangle$ *such that for* $A \cap B = \emptyset$, *if* $\langle h''', s\rangle R_A \langle h'''', s'\rangle$ *then* $\langle h', s\rangle R_A \langle h'''', s'\rangle$ *and if* $\langle h''', s\rangle R_B \langle h''''', s''\rangle$ *then* $\langle h'', s\rangle R_B \langle h''''', s''\rangle$ *(independence of agency)*
- *The* $\sim_a$ *are epistemic equivalence relations over the elements of* $H \times S$ *such that:*
  - $\sim_a \circ R_a \subseteq \sim_a \circ R_{Ags}$ *(agents cannot know what choices other agents perform concurrently)*
  - $R_{Ags} \circ \sim_a \subseteq \sim_a \circ R_a$ *(agents recall the effects of the actions they knowingly perform themselves)*

[1] This terminology is inspired by Coalition Logic, where in the semantics the actions (or 'choices') are represented by effectivity functions.

Definition 4. *A frame* $\mathcal{F} = \langle H, S, R_\Box, \{R_A \mid A \subseteq Ags\}, \{\sim_a \mid a \in Ags\}\rangle$ *is extended to a model* $\mathcal{M} = \langle H, S, R_\Box, \{R_A \mid A \subseteq Ags\}, \{\sim_a \mid a \in Ags\}, \pi\rangle$ *by adding a valuation* π *of atomic propositions:*

- π *is a valuation function* $\pi : P \longrightarrow 2^{H\times S}$ *assigning to each atomic proposition the set of history/state pairs in which they are true.*

The truth conditions for the semantics of the operators on these models is standard for a two-dimensional modal logic [17].

Definition 5. *Validity* $\mathcal{M}, \langle h, s\rangle \models \varphi$*, of a formula* φ *in a history/state pair* $\langle h, s\rangle$ *of a model* $\mathcal{M} = \langle H, S, R_\Box, \{R_A \mid A \subseteq Ags\}, \{\sim_a \mid a \in Ags\}, \pi\rangle$ *is defined as:*

$$
\begin{array}{ll}
\mathcal{M}, \langle h, s\rangle \models p & \Leftrightarrow \langle h, s\rangle \in \pi(p) \\
\mathcal{M}, \langle h, s\rangle \models \neg\varphi & \Leftrightarrow \textit{not } \mathcal{M}, \langle h, s\rangle \models \varphi \\
\mathcal{M}, \langle h, s\rangle \models \varphi \wedge \psi & \Leftrightarrow \mathcal{M}, \langle h, s\rangle \models \varphi \textit{ and } \mathcal{M}, \langle h, s\rangle \models \psi \\
\mathcal{M}, \langle h, s\rangle \models K_a\varphi & \Leftrightarrow \langle h, s\rangle \sim_a \langle h', s'\rangle \textit{ implies that } \mathcal{M}, \langle h', s'\rangle \models \varphi \\
\mathcal{M}, \langle h, s\rangle \models \Box\varphi & \Leftrightarrow \langle h, s\rangle R_\Box \langle h', s'\rangle \textit{ implies that } \mathcal{M}, \langle h', s'\rangle \models \varphi \\
\mathcal{M}, \langle h, s\rangle \models [A \textit{ xstit}]\varphi & \Leftrightarrow \langle h, s\rangle R_A \langle h', s'\rangle \textit{ implies that } \mathcal{M}, \langle h', s'\rangle \models \varphi
\end{array}
$$

Satisfiability, validity on a frame and general validity are defined as usual.

While the semantics is very standard from a (two-dimensional) modal logic perspective, the relation with standard STIT semantics deserves some explanation. In the conditions on the frames we recognize standard STIT properties like 'no choice between undivided histories' and properties that are specific for the present STIT version, like 'actions take effect in successor states'. Actually, the frames can easily be pictured as trees where histories branch in states, like in standard STIT theory. The main difference is that *states* are not partitioned into choice sets. The choice sets appear here (implicitly) as sets of possible *next* states (like in Coalition Logic). From a given 'actual' history/state pair, we reach these choice sets by first jumping (along $R_\Box$) to another history through the same state, and then looking (along R_A) what next states are reachable through the choice made by agents on that history.

In particular one aspect of the present semantics needs extra clarification. Like in standard STIT semantics, all the history/state pairs belonging to one state can have different valuations of atomic propositions. In standard STIT formalisms this is actually needed to give semantics to the instantaneous effects of actions. But here, as said, the effects are not instantaneous. Therefore, in the present logic, the fact that different histories through the same state can have different valuations of non-temporal propositions, does not carry much meaning. Of course, in the logic we can talk about atomic propositions being true or not in other histories through the same state. For instance, the formula "$\Box p$" expresses that all the histories through the present state have in common that the atomic proposition p holds on them. But the point is that one might think that actually we should *impose* on the models that all histories through a state come with identical valuations of atomic propositions. That would induce the

property $\varphi \rightarrow \Box\varphi$ for φ any 'STIT-operator-free' formula (in [10] we give a system encompassing this axiom). However, this would complicate establishing a completeness result, and does not strengthen the logic in any essential or interesting way. I think there is no need at all to impose such a condition. Since actions only take effect in next states, alternative valuations for *atomic* propositions on other histories through the same state are 'harmless'.

Now I go on to the axiomatization of the logic. Actually, axiomatization is really easy. The approach I have taken for constructing this logic is to build up the semantic conditions on frames and the corresponding axiom schemes simultaneously, while staying within the Sahlqvist class. This ensures that the semantics cannot give rise to more logical principles than can be proven from the axiomatization.

Definition 6. *The following axiom schemas, in combination with a standard axiomatization for propositional logic, and the standard rules (like necessitation) for the normal modal operators, define a Hilbert system:*

	S5 for $\Box$
	KD for each $[A\ \mathit{xstit}]$
(C-Mon)	$[A\ \mathit{xstit}]\varphi \rightarrow [A \cup B\ \mathit{xstit}]\varphi$
(Indep)	$\Diamond[A\ \mathit{xstit}]\varphi \wedge \Diamond[B\mathit{xstit}]\psi \rightarrow \Diamond([A\ \mathit{xstit}]\varphi \wedge [B\ \mathit{xstit}]\psi)$ *for* $A \cap B = \emptyset$
(Det)	$\neg X\neg\varphi \rightarrow X\varphi$
(Ineff-$\emptyset$)	$[\emptyset\ \mathit{xstit}]\varphi \rightarrow \Box X\varphi$
(X-Eff)	$\Box X\varphi \rightarrow [A\ \mathit{xstit}]\varphi$
(N-C-U-H)	$[A\ \mathit{xstit}]\varphi \rightarrow X\Box\varphi$
	S5 for each K_a
(Know-X)	$K_aX\varphi \rightarrow K_a[a\ \mathit{xstit}]\varphi$
(Rec-Eff)	$K_a[a\ \mathit{xstit}]\varphi \rightarrow XK_a\varphi$

Theorem 1. *The Hilbert system of definition 6 is complete with respect to the semantics of definition 5.*

Proof. The axioms are all within the Sahlqvist class. This means that the axioms are all expressible as first-order conditions on frames and that they are complete with respect to the corresponding frame classes, cf. [7, Th. 2.42]. So the only thing we need to check is whether the axioms correspond one-to-one to the semantic conditions defined on the frames. This can be done automatically, using for instance SQEMA [14].

As part of the above axiomatization, we recognize Ming Xu's axiomatization for multi-agent STIT logics (see the article in [6]). Xu's axiomatization is for the standard, instantaneous STIT variant. But, it should not come as a surprise that the same axioms apply to the present logic. The central property in Xu's axiomatization is the 'independence of agency' property. But the issue of independence of choices of different agents does not depend on the condition that effects are instantaneous or occur in next states.

As a proposition I list some theorems. Derivation of these is just a little exercise in normal modal logic. The last theorem in the list below is the well

known 'perfect recall' or 'no forgetting' axiom, known from the literature on the interaction between epistemic and temporal modalities.

Proposition 1. *The following are derivable:*

$$[A\ \mathit{xstit}]\varphi \wedge [B\ \mathit{xstit}]\psi \rightarrow [A \cup B\ \mathit{xstit}](\varphi \wedge \psi)$$
$$\Box X\varphi \rightarrow X\Box\varphi$$
$$[A\ \mathit{xstit}]\varphi \rightarrow X\varphi$$
$$X\neg\varphi \rightarrow \neg X\varphi$$
$$\Box X\varphi \leftrightarrow [\emptyset\ \mathit{xstit}]\varphi$$
$$K_a X\varphi \leftrightarrow K_a[a\ \mathit{xstit}]\varphi$$
$$K_a X\varphi \rightarrow X K_a\varphi$$

Pauly's Coalition logic [23] is a logic of ability that is very closely related to STIT formalisms, as was shown in [11]. Since in Coalition Logic actions also take effect in next states, restricting the STIT formalism by only allowing effects in next state, as in the logic of this paper, does not inhibit definability of Coalition Logic.

Theorem 2. *Coalition logic, whose central operator is* $[A]\varphi$ *for 'agents* A *are able to do* φ*', is embedded into the present logic by the definition* $[A]\varphi := \Diamond[A\ \mathit{xstit}]\varphi$ *(plus the obvious isomorphic translations for other connectives).*

Proof. The same strategies as in [11] and [10] can be applied. First we make sure that the axioms of coalition logic, after applying the above translation, are valid for the present logic. Here I will not verify this explicitly, and I only list the translated CL axioms:

$$(\bot) \quad \neg\Diamond[A\ \mathsf{xstit}]\bot$$
$$(\top) \quad \Diamond[A\ \mathsf{xstit}]\top$$
$$(\mathrm{N}) \quad \Diamond[\emptyset\ \mathsf{xstit}]\varphi \vee \Diamond[Ags\ \mathsf{xstit}]\neg\varphi$$
$$(\mathrm{MON}) \quad \Diamond[A\ \mathsf{xstit}](\varphi \wedge \psi) \rightarrow \Diamond[A\ \mathsf{xstit}]\varphi$$
$$(\mathrm{S}) \quad \Diamond[A\ \mathsf{xstit}]\varphi \wedge \Diamond[B\ \mathsf{xstit}]\psi \rightarrow \Diamond[A \cup B\ \mathsf{xstit}](\varphi \wedge \psi) \text{ for } A \cap B = \emptyset$$

It is quite straightforward to verify these properties for the present logic, either semantically, or as theorems in the Hilbert system. To complete the proof, we also have to show that the translation preserves validity in the other direction. Or, equivalently, we check that it preserves satisfiability in the same direction. That is, given that a CL formula is satisfiable on a CL-model, we have to show that its translation is satisfiable on the models I defined in this paper. This is not difficult to show given the structural similarities between CL-models and the models in this paper.

3 More on 'Knowingly Doing'

Since it is a rather new, in this section 1 elaborate on the notion of 'knowingly doing'. I explain what it means to do something *(un)knowingly*. I gave semantics in terms of models where epistemic equivalence sets (information sets) contain

history/state pairs. An agent knowingly does something if his action holds for all the history/state pairs in the epistemic equivalence set that contains the *actual* history/state pair.

Several closure conditions apply. The first one says that epistemic equivalence sets are closed under choices[2]. The corresponding axiom, is $K_aX\varphi \rightarrow K_a[a\ \mathsf{xstit}]\varphi$ (this property does not hold if the STIT operator is replaced by a *deliberative* STIT oparator). This property ensures that an agent cannot know that two histories through the same choice are different, which reflects that agents cannot knowingly do *more* then what is affected by the choices they have. In particular, the property $K_aX\varphi \rightarrow K_a[a\ \mathsf{xstit}]\varphi$ says that agents can only know things about the (immediate) future if they are the result of an action they themselves knowingly perform. Then, an agent *unknowingly* does everything that is (1) true for all the history/state pairs belonging to the actual *choice* it makes in the actual *state*, but (2) not true for all the history/state pairs it considers possible. In general the things an agent does unknowingly vastly outnumber the things an agent *knows* he does. For instance, by sending an email, I may enforce many, many things I am not aware of, which are nevertheless the result of me sending the email. All these things I do *unknowingly* by knowingly sending the email.

Another, equivalent way of interpreting the property $K_aX\varphi \rightarrow K_a[a\ \mathsf{xstit}]\varphi$ is to say that it expresses that agents cannot know what actions other agents perform concurrently. This is because choices of other agents always refine the choice of the agent whose choice we consider. Then, knowing the choice of the other would mean that the agent would be able to know more about the future state of affairs then is guaranteed by his own action. Yet another way of explaining this is to say that for any agent, the histories within its choices are indistinguishable.

The second constraint on the interaction between knowledge and action is the one expressed by the axiom $K_a[a\ \mathsf{xstit}]\varphi \rightarrow XK_a\varphi$. The issue here is that if agents knowingly see to it that a condition holds in the next state, in that same next state they will recall that the condition holds.

The epistemic equivalence sets of history/state pairs represent the actions an agent knows it can do. The historical possibility operator ranges over the histories in these classes. So we can say things like: there is a history for which the agent knows it can take the bus. It might be considered puzzling that the agent can knowingly take the bus while at the same time it knows it is not taking the bus. A similar issue arises in standard STIT logic, that is, without the knowledge operator. In STIT it is consistent to assert that an agent actually does p, while he can do $\neg p$. This may seem inherently contradictory, since, if the agent *actually* does p, how can it be that at the same he is *able* to do $\neg p$? Is it not the case that the fact that he actually does p prevents him from being able to do $\neg p$ at the same state? A possible answer to such questions is that truth of the operator $[a\ \mathsf{xstit}]p$ should better be associated to the agent having

[2] An extreme case is where the information sets are exactly the choices in each state. In that case an agent knows all the consequences of his actions.

'decided' on p, and not to the agent 'doing' p. This interpretation leaves room for still being able to 'decide' $\neg p$ at the same time, because decisions can be reconsidered.

The above discussion also shows that we have to adopt a slightly more agile stance towards the concept of 'actual world/history'. This should not come as a surprise. We talk about agents choosing between actions thereby determining themselves what will be the actual worlds (histories). So it no longer makes sense to talk of an actual world independent of what agents do/choose. In standard S5 epistemic logic, one usually pays no attention to the meaning of the epistemic equivalence classes outside the one containing the actual world. One actually never asks what the meaning of these classes is. And indeed in S5 epistemic logic we may neglect their meaning, since we evaluate only with respect to an actual world that is independent of what agents choose. But in our present setting, a historical possibility operator ranges over the histories contained by the different epistemic equivalence classes. So in this setting, the equivalence classes outside the one containing the actual world, *do* have meaning. In our setting they mean that the agent can knowingly do the associated actions thereby forcing the actual world to be among a different set of histories.

4 A Discussion on Applications and Further Extensions

4.1 Deliberate Action

The kind of STIT operator I defined above has often been criticized for properties like $[A\ \mathsf{xstit}]\top$. The idea is that agents should not be able to bring about things that are true inevitably, but only things that without their intervention might not become true. If we want an operator that takes this into account we can easily define a deliberative version of the STIT operator, as follows:

$$[A\ \mathsf{d1xstit}]\varphi \equiv_{def} [A\ \mathsf{xstit}]\varphi \wedge \neg\Box X\varphi$$

This is the standard way in the literature for defining the deliberative STIT in a so called 'Chellas' STIT. However, the addition of a knowledge operator and the introduction of the notion of 'knowingly doing' enables us to give a more fine-grained analysis of 'deliberateness'. It seems strange to accept that agents can deliberately do something without knowing that they do it. So the action part of the above definition should better be replaced by a 'conformant' STIT.

$$[a\ \mathsf{d2xstit}]\varphi \equiv_{def} K_a[a\ \mathsf{xstit}]\varphi \wedge \neg\Box X\varphi$$

But now what about the side condition? Should the agent be aware of the fact that there is a side condition $\neg\Box X\varphi$ saying that the outcome *could* have been different if it was not for a's action, or not? If this question is answered affirmatively (which I think is the better option), we can define:

$$[a\ \mathsf{d3xstit}]\varphi \equiv_{def} K_a[a\ \mathsf{xstit}]\varphi \wedge K_a\neg\Box X\varphi$$

4.2 Autonomy and (In)Dependency

Note that we can express that agents A see to it that agents B see to it that φ as $[A\ \mathsf{xstit}][B\ \mathsf{xstit}]\varphi$. So, since in our STIT version effects only occur in next states, one agent being able to influence the behavior of some other agent is expressed as a simple nesting of modalities. In the standard, instantaneous STIT formalisms, we actually have that a nesting of the STIT operators is equivalent with the *ineffective* action, that is, we have the axiom $[A\ \mathsf{stit}][B\ \mathsf{stit}]\varphi \leftrightarrow [\emptyset\ \mathsf{stit}]\varphi$ (in [10] we used this to show that in standard STIT formalisms the historical necessity operator is definable). In standard STIT formalisms the above axiom actually replaces the 'independence of agency' axiom I give in the present paper. For the present STIT version, I had to formulate the independency property explicitly by using Xu's axiom.

If one group of agents A influencing another group B can be expressed as $[A\ \mathsf{xstit}][B\ \mathsf{xstit}]\varphi$, then the concept of 'autonomy' for a group of agents A *relative* to some other group B and a certain property φ might be associated to something like $\neg[B\ \mathsf{xstit}][A\ \mathsf{xstit}]\varphi$. Of course this is a very specific expression of autonomy, because it is relative to another group of agents and a property φ. If we are interested in autonomy of a group of agents 'as such' we have to quantify the other agents and properties out. This is clearly a topic of future research, since it is beyond the scope of the present language.

There is a direct link between autonomy and deliberateness of actions. The relation is that if other agents can see to it that you are no longer able to *deliberately* see to something, you are not an autonomous agent.

4.3 Deontic Modalities

For the extension of this framework with an operator for 'ought-to-do', I adapt the approach taken by Bartha [5] who introduces Anderson style ([3]) violation constants in STIT theory. The approach with violation constants is very well suited for theories of ought-to-do, witnessing the many logics based on adding violation constants to dynamic logic [22,8]. However, I believe that the STIT setting is even more amenable to this approach. Some evidence for this is found in Bartha's article ([5]), who shows that many deontic logic puzzles (paradoxes) are representable in an intuitive way. And a clear advantage in the present approach is that since our base STIT logic is complete, defining obligation as a reduction using violation constants guarantees that completeness is preserved under addition of the obligation operator

To define an operator for 'obligation to do', I adapt the approach of Bartha [5] to the present situation where actions only take effect in next states. The intuition behind the definition is straightforward: an agent is obliged to do something if and only if by not doing it, it performs a violation. As said, the difference with Bartha's definition is that the effect of the obliged action can only be felt in next states, which is why also violations have to be properties of next states. Formally, our definition is given by:

$$O[a\ \mathsf{xstit}]\varphi \equiv_{def} \Box(\neg[a\ \mathsf{xstit}]\varphi \rightarrow [a\ \mathsf{xstit}]V)$$

First note that I slightly abuse notation by denoting $[\{a\}\ \mathsf{xstit}]\varphi$ as $[a\ \mathsf{xstit}]\varphi$. Also note that $\neg[a\ \mathsf{xstit}]\varphi$ expresses that A do not see to it that φ, which is the same as saying that A 'allow' a choice where $\neg\varphi$ is a possible outcome. The definition then says that all such choices *do* guarantee that a violation occurs.

The $\Box$ operator in the definition ensures that obligations are 'moment determinate'. This means that their validity only depends on the state, and not on the history (see [21] for a further explanation of this concept). I think that this is correct. But see [26] for an opposite opinion.

The above defined obligation is a 'personal' one. If, by 'coincidence', φ occurs, apparently due the action of other agents, while the agent bearing the obligation did not make a choice that *ensured* that φ would occur, a violation is guaranteed. So agents do not escape an obligation by letting other agents do the work for them. Using the notion of 'knowingly doing' we can define other variants of obligation, but we reserve that for another paper.

A seemingly bad feature of the above definition of obligations is that it results in the formulas $\Box[a\ \mathsf{xstit}]\varphi \rightarrow O[a\ \mathsf{xstit}]\varphi$ and $\Box[a\ \mathsf{xstit}]V \rightarrow O[a\ \mathsf{xstit}]\varphi$ being theorems. It is sometimes argued that these counter-intuitive properties might be avoided by concentrating on *conditional* obligations. But the solution is much simpler than that. We only have to replace the STIT operator in the definition of obligation by a deliberative STIT operator. Again, the exact details of these deontic issues will be discussed in a future paper.

4.4 STIT Versus Dynamic Logic

The final theme in this paper will be the distinction between our present formalism and dynamic logic formalisms [25,19]. I want to contribute to the discussion about which formalism is better suited as a basis for logics for multi-agent systems. I know there are many arguments in favor of dynamic logic. But here I give a few arguments in favor of STIT formalisms.

Action Negation. In PDL there are no really satisfactory solutions for defining action negation. The natural 'stance' on this issue is to look for a semantics for dynamic logic object level formulas of the form $[\sim\alpha]\varphi$, where '$\sim$' constructs an action that is the negation of the action α. The problem is, of course, that if we have no idea about what the action α is more than just a name for something unspecified, we have no idea what it is we have to negate. Do we negate the effect, and refer to the opposites effect? What is the opposite effect if the effect is not even specified (see [9] for a possible answer)? Or do we negate the possibility for execution of the action? That is, does negation of an action mean that we cannot execute it? Or do we assume that α is a complex action, that can thus be seen as a non-deterministic program, and do we refer to all other programs? To which other program? To the non-deterministic choice between all of them? These are many questions. Several authors have tried to give an answer to some of these question, resulting in several concrete proposals [9,27,22]. However, a definite answer seems hard to obtain.

In STIT the issue is of negating actions is completely clear. Since an action is identified with its own effect, it is clear what we have to negate: the effect. By

doing so, we obtain the well known STIT definitions for 'refraining'. Refraining is defined as not seeing to it that a certain effect is ensured. In other words: allowing that the opposite effect might occur.

Concurrent Action. More or less the same story can be told for modeling concurrency of action. In STIT things are clear: concurrency of acts of an individual agent is modeled by logical conjunction, and concurrency of acts by different agents is modeled by group STIT. As shown in section 2, coalition logic is definable in our STIT logic. Coalition logics super-additivity axiom, which in our logic comes in the form $\Diamond[A\ \mathsf{xstit}]\varphi \wedge \Diamond[B\ \mathsf{xstit}]\psi \rightarrow \Diamond[A \cup B\ \mathsf{xstit}](\varphi \wedge \psi)$ for $A \cap B = \emptyset$, can be read directly as an axiom on concurrent action: if A and B each can do something, together they can do it concurrently.

In PDL, again, things are less clear. Several proposals are around [24,22,15]. Actually, all semantics that have been developed in concurrency theory [18], are amenable for being imported in a dynamic logic formalism.

Agency. STIT is a logic for *agency*. In STIT we can express abilities of agents, talk about autonomy and dependency, and formulate logical properties concerning how abilities of groups relate to abilities of group-members. All of these things are much harder to express in dynamic logic formalisms. Actually, I do not know of any dynamic logic formalisms with a satisfactory solution for defining agency. Agents are usually simply introduced as indexes to dynamic logic modalities.

5 Conclusions

In this paper I presented an epistemic STIT logic that improves on earlier work in several ways. First of all there is a new semantics that is close to the STIT semantics in the philosophical literature. At the same time, the semantics is completely standard which enables me to ensure completeness by referring to the Sahlqvist result. Second, new interactions between time and action, and between knowledge and action are proposed and incorporated in the logic. Finally, I discuss the application of the logic to reasoning about norms, knowledge, autonomy, and several other multi-agent concepts. I conclude with a brief discussion on the pros and cons of the STIT formalism as compared to systems based on dynamic logic. I gave some arguments in favor of taking STIT formalisms instead of dynamic logics as the basis for logics for representing multi-agent concepts.

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. In: Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, Florida (October 1997)
2. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM 49(5), 672–713 (2002)

3. Anderson, A.R.: A reduction of deontic logic to alethic modal logic. Mind 67, 100–103 (1958)
4. Balbiani, P., Gasquet, O., Herzig, A., Schwarzentruber, F., Troquard, N.: Coalition games over Kripke semantics: expressiveness and complexity. In: Dègremont, C., Keiff, L., Rückert, H. (eds.) Festschrift in Honour of Shahid Rahman, College Publications (to appear, 2008)
5. Bartha, P.: Conditional obligation, deontic paradoxes, and the logic of agency. Annals of Mathematics and Artificial Intelligence 9(1-2), 1–23 (1993)
6. Belnap, N., Perloff, M., Xu, M.: Facing the future: agents and choices in our indeterminist world, Oxford (2001)
7. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press, Cambridge (2001)
8. Broersen, J.M.: Modal Action Logics for Reasoning about Reactive Systems. PhD thesis, Faculteit der Exacte Wetenschappen, Vrije Universiteit Amsterdam, februari (2003)
9. Broersen, J.M.: Relativized action negation for dynamic logics. In: Balbiani, P., Suzuki, N.-Y., Wolter, F., Zakharyaschev, M. (eds.) Advances in Modal Logic, vol. 4, pp. 51–70 (2003)
10. Broersen, J.M., Herzig, A., Troquard, N.: A normal simulation of coalition logic and an epistemic extension. In: Proceedings Theoretical Aspects Rationality and Knowledge (TARK XI), Brussels (2007)
11. Broersen, J.M., Herzig, A., Troquard, N.: From coalition logic to STIT. In: Proceedings LCMAS 2005. Electronic Notes in Theoretical Computer Science, vol. 157, pp. 23–35. Elsevier, Amsterdam (2005)
12. Broersen, J.M., Herzig, A., Troquard, N.: Embedding Alternating-time Temporal Logic in strategic STIT logic of agency. Journal of Logic and Computation 16(5), 559–578 (2006)
13. Broersen, J., Herzig, A., Troquard, N.: A STIT-extension of ATL. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS, vol. 4160, pp. 69–81. Springer, Heidelberg (2006)
14. Conradie, W., Goranko, V., Vakarelov, D.: Algorithmic correspondence and completeness in modal logic I: The core algorithm SQEMA. Logical Methods in Computer Science 2(1), 1–26 (2006)
15. Danecki, R.: Nondeterministic propositional dynamic logic with intersection is decidable. In: Skowron, A. (ed.) SCT 1984. LNCS, vol. 208, pp. 34–53. Springer, Heidelberg (1985)
16. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics, ch. 14, pp. 996–1072. Elsevier, Amsterdam (1990)
17. Gabbay, D.M., Kurucz, A., Wolter, F., Zakharyachev, M.: Many-Dimensional Modal Logics: Theory and Applications. Elsevier, Amsterdam (2003)
18. van Glabbeek, R.J.: Comparative Concurrency Semantics and Refinement of Actions, 2nd edn. CWI Tract, vol. 109. CWI, Amsterdam (1996)
19. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. The MIT Press, Cambridge (2000)
20. Herzig, A., Troquard, N.: Knowing How to Play: Uniform Choices in Logics of Agency. In: Weiss, G., Stone, P. (eds.) 5th International Joint Conference on Autonomous Agents & Multi Agent Systems (AAMAS 2006), Hakodate, Japan, pp. 209–216. ACM Press, New York (2006)
21. Horty, J.F.: Agency and Deontic Logic. Oxford University Press, Oxford (2001)
22. Meyer, J.-J.C.: A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. Notre Dame Journal of Formal Logic 29, 109–136 (1988)

23. Pauly, M.: A modal logic for coalitional power in games. Journal of Logic and Computation 12(1), 149–166 (2002)
24. Peleg, D.: Communication in concurrent dynamic logic. Journal of Computer and System Sciences 35, 23–58 (1987)
25. Pratt, V.R.: Semantical considerations on Floyd-Hoare logic. In: Proceedings 17th IEEE Symposium on the Foundations of Computer Science, pp. 109–121. IEEE Computer Society Press, Los Alamitos (1976)
26. Wansing, H.: Obligations, authorities, and history dependence. In: Wansing, H. (ed.) Essays on Non-classical Logic, pp. 247–258. World Scientific, Singapore (2001)
27. Wansing, H.: On the negation of action types: Constructive concurrent PDL. In: Valdes-Villanueva, L., Hájek, P., Westerstahl, D. (eds.) Proceedings of the Twelfth International Congress of Logic Methodology and Philosophy of Science (LMPS 2003), pp. 207–225. King's College Publications (2005) (invited lecture)

Combining Multiple Knowledge Representation Technologies into Agent Programming Languages

Mehdi M. Dastani[1], Koen V. Hindriks[2], Peter Novák[3], and Nick A.M. Tinnemeier[1]

[1] Utrecht University, Utrecht, The Netherlands
{mehdi,nick}@cs.uu.nl
[2] Delft University of Technology, Delft, The Netherlands
k.v.hindriks@tudelft.nl
[3] Clausthal University of Technology, Clausthal-Zellerfeld, Germany
peter.novak@tu-clausthal.de

Abstract. In most agent programming languages in practice a programmer is committed to the use of a single knowledge representation technology. In this paper we argue this is not necessarily so. It is shown that rational agent programming languages allow for the combination of various such technologies. Specific issues that have to be addressed to realize such integration for rational agents that derive their choice of action from their beliefs and goals are discussed. Two techniques to deal with these issues which enable the integration of multiple knowledge representation techniques are presented: a meaning-preserving translation approach that maps one representation to another, and an approach based on so-called bridge rules which add additional inference power to a system combining multiple knowledge representation technologies.

1 Introduction

Rational agent programming has been motivated on several grounds. One of its motivations has been to provide for a high-level specification framework for agent programs based on common sense concepts such as beliefs, goals, actions, and plans. Such a programming framework comes with several benefits, among others that, though the programming framework is abstract, it can be realized computationally, and, that the programming framework is based on common sense intuitive concepts which nevertheless have a well-defined semantics.

In our view, rational agent programming is abstract in one sense in that it does not commit to a particular knowledge representation language. Though it is common in several concrete programming languages for rational agents to use Prolog like expressions to represent an agent's mental states (cf. Jason[1], 2APL[4], and GOAL[6]), this is more a de facto standard and is not implied by the concept of rational agent programming itself. Some agent frameworks such as Jadex[16] and JACK[18] have taken a much more pragmatic road and use object oriented technology to implement the beliefs and goals of an agent.

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 60–74, 2009.

Even though in implemented agent programming systems there are ways to add a limited support for external KR technologies, such as accessing a database engine, this is achieved on the technological level and the solution is bound to be proprietary w.r.t. the implemented application. Therefore as far as we know, it can be said that all of the existing programming frameworks for rational agents in principle commit to a particular kind of knowledge representation (in the sense of Definition 1; see Section 2). In our view, this is not so much a limitation implied by rational agent programming per se. Moreover, we believe that rational agent programming has the integrative potential to facilitate and support the design and construction of agents that use multiple and various knowledge representation languages. In this sense, we are inspired by similar ideas of using various knowledge representation languages that motivated the knowledge interchange format project (KIF)[15].

As a motivating example to allow multiple knowledge representations for building rational agents consider Ape (Airport Passenger E-assistant), a mobile robot with the task of helping passengers by providing them with flight schedules and getting them at the right gate on time. Ape needs to make decisions about, for example, who to serve first, and when to call for the assistance of airport personnel in case she cannot handle all requests in time. In practice, to support such decision-making, it may be best to use various representation technologies. For example, Ape could use Prolog to reason about various decision options, and may have access to a Geographic Information System (GIS) containing the topology of the airport and an Oracle database with the flight schedules.

The aim of this paper is to explore the use of multiple knowledge representations, not only between agents, but also *within a single agent*, both from a pragmatic, practical perspective as well as a theoretical perspective. One of the main motivations to do this stems from the fact that various knowledge representations come with various strengths and weaknesses which would be inherited by an agent program if such a program would be restricted to the use of only a single knowledge representation language. This point has also been particularly argued for by Marvin Minsky[13]. In this work, he argues that to organize intelligence multiple representations are required to do the job. We believe that rational agent programming may provide part of a solution for integrating such a multitude of representations in a clean and well-organized manner with applications in mind.

From a pragmatic point of view, we would also not like rational agent programming languages to commit a programmer to learn a new and specific knowledge representation language that comes with the agent programming language. Learning to program rational agents should not necessarily mean also learning a new and unfamiliar language for representing knowledge. The programmer should have (at least some) freedom to choose his or her favorite language for representing facts about the application domain. The motivation thus is practical, but at the same time it has been and still is quite hard to combine various knowledge representations in a useful manner. In practice, moreover, it should be possible to develop agent applications that incorporate legacy databases, such

as, for example, an Oracle flight schedule database. To facilitate such integration, the use of a particular agent language should not imply the need to redesign the original database but instead an agent language ideally would support and provide an Oracle interface.

In Section 2, we first define knowledge representation in a formally precise way. In order to illustrate the use of multiple knowledge representations within a single agent, we present and discuss in Section 3 the syntax and semantics of the GOAL [6] agent programming language. In Section 4, we introduce a technique to integrate different knowledge representations for the case that the beliefs and goals of a single agent are represented by means of different knowledge representations. In Section 5, we discuss another technique for the case that the belief base of a single agent is represented using different knowledge representations. Finally, in Section 6, we discuss related work and conclude the paper.

2 KR Technology

Our interest in this paper is in basic representation and reasoning tools such as logic, Bayesian networks, or others more than in upper ontologies or domain-specific ontologies. In [5] such representation and reasoning tools are referred to as *knowledge technologies*, a convention we will adopt here as well. A knowledge representation is characterized by means of five roles. Here we are particularly interested in two roles associated with knowledge technologies:

1. Knowledge technologies provide a *fragmentary theory of intelligent reasoning*, i.e. a knowledge technology defines a notion of *inference* that enables drawing conclusions from other available information *represented by means of the same technology*, and
2. Knowledge technologies provide a *medium for pragmatically efficient computation*, i.e. a knowledge technology provides tools and techniques to *compute* with (or to use) the representations supported by the technology.

Other roles of knowledge representations discussed in [5] as being a set of ontological commitments or as providing a medium of human expression are less important in this context. The choice of ontology presumably is application driven, whereas the integration of various representation tools into agent programming languages poses more general challenges.

For our purposes, it is useful to more formally define what a knowledge technology is. Our aim is to relate the semantics of agent programming languages with a *very generic* concept of a knowledge technology. Informally, a knowledge technology is defined here as a language with a *well-defined semantics* that comes with an *inference relation* and *procedures to update* information stored in a knowledge base. Though update procedures may not commonly be regarded as part of a knowledge technology, in our view providing some support and concept of updating a knowledge base to maintain a correspondence with the entities being represented is essential. Moreover, in the context of agent programming update operators are essential, which provides our main motivation to include them in our definition.

Definition 1. (Knowledge Representation Technology)
A knowledge representation technology *is defined as a tuple:*

$$\langle \mathcal{L}, \models, \oplus \rangle$$

where $\mathcal{L}$ *is a representation language which can be used to express declarative sentences, with a given set* $L_q \subseteq L$ *of* query expressions, $\models : 2^L \times L_q \rightarrow \{\top, \bot\}$ *is an* inference relation, *and* $\oplus : 2^L \times L \rightarrow 2^L$ *is an* update operator.

Definition 1 is intentionally kept very abstract. Our concern here is with the roles that a knowledge representation can fulfill in the context of agent languages. The main roles in this context are that it can be used to *represent* an agent's mental states (e.g., beliefs and goals) and to *query* the agent's mental state, as well as that it allows for performing an *update* on stored representations to maintain a correspondence of the agent's mental state and its environment. These operations can be conceptualized as providing a TELL and ASK interface on a database of stored representations as discussed in [10]. We assume (though it is in our quite general setting not strictly necessary to do so) that provided consistency of a database Σ and a sentence ϕ the update operator $\oplus$ preserves consistency, i.e. $\Sigma \oplus \phi$ is consistent as well and the database remains unchanged if the sentence to update with is inconsistent ($\Sigma \oplus \bot = \Sigma$). $\bot$ denotes here falsity. Information is expressed by means of *sentences* from the language that can be *true or false*, but we also allow for semantics that incorporate more truth values as is typical in multi-valued logic. A *knowledge base*, or, alternatively, simply a *database*, is then defined as a set of sentences from the knowledge representation language.

The use of the inference relation and update operations in defining an agent language is clarified below where we show how the semantics of an agent language can be defined in terms of these operators. In concrete agent languages the update operator typically is not a single operator $\oplus$ but one or more operations to *add* and *remove* stored representations from a database (though it may be a single operator, as e.g. postconditions of actions can be interpreted in a STRIPS-like fashion as add/delete lists). The definition provided suits, however, for the purposes of this paper.

Some typical examples of knowledge representation technologies that fit the definition are logical languages such as first-order logic and description logics such as the Ontology Web Language (OWL), Frame languages [12], Prolog, Answer Set Programming, Constraint Programming, relational databases, and others, such as Bayesian Network also fit though the notion of an update in Bayesian Networks is rather limited and it is not common to view a Bayesian network as a database consisting of a set of sentences.

To illustrate the scope of our definition, we discuss the example of relational databases - one of the most common technologies for storing information in practice - in slightly more detail. In this case, the representational language $\mathcal{L}$ can be identified with languages such as Datalog [3] or SQL. Datalog is a declarative database query language whereas SQL is a declarative language for both querying and updating relations stored in a database. SQL query formulas provide the query language $\mathcal{L}_q$ whereas SQL update formula can be used to

specify the insertion or removal of relations from a database. Finally, the SQL interpreter (the relational database engine) implements the inference relation $\models$ and update operator $\oplus$. The correspondence between Datalog or SQL and the abstract KR language scheme of Definition 1 thus is rather straightforward.

3 Integrating Multiple KRTs into Rational Agents

The question of how to usefully integrate multiple knowledge representations into a software application in general poses several complex issues [15]. One particularly interesting question is how to facilitate the derivation of a conclusion from knowledge stored in different representations in various databases controlled or accessible by the agent. The combination of multiple knowledge technologies into a rational agent programming language, however, raises some specific issues of its own. To illustrate these, we first provide a very brief overview of the essential ingredients of the agent programming language GOAL [6], which are introduced here mainly for illustrative purposes. Only those parts of the GOAL language related to the subject of this paper are introduced. The interested reader is referred to [6] for a more extensive presentation of the language. GOAL agents - as agents programmed in related agent programming languages such as 2APL, Jason, and Jadex - derive their choice of action from their *beliefs* and *goals*. Beliefs and goals are represented by means of some knowledge technology (as mentioned earlier, typically Prolog is used), and, for the purpose of this paper, it will be particularly relevant to look into the relation between these two notions.

3.1 GOAL: Syntax and Semantics

The main defining features of a rational agent programming language are constructs for defining the agent's mental state, including its *beliefs and goals*, its *action selection mechanism* used by the agent to derive a choice of action from its beliefs and goals, and a *commitment strategy* which determines when an agent will revise its goals given its beliefs. An example of a commitment strategy is to only drop a goal when the agent believes it has been achieved, a so-called *blind* commitment strategy (cf. [17]). The semantic interdependence of an agent's goals and beliefs differentiate rational agent programming languages from other high-level languages such as database languages. At the same time, however, this interdependency raises some special issues for integrating multiple knowledge representation languages into such a language. In order to clarify these issues some of the key semantic rules of GOAL that formalize these interdependencies are introduced.

In the following we use $\langle \Sigma, \Gamma \rangle$ to denote an arbitrary mental state of a rational agent where Σ is the *belief base* and Γ is the *goal base* of the agent. Although it is usual to assume both the belief base as well as the goal base consist of sentences from a single knowledge technology (e.g. Prolog), for the purposes of this paper, we are interested in relaxing this assumption in two ways. First, the belief base and goal base do not need to be based on one and the same knowledge technology. Second, the belief base does not need to be monolithic and might

instead consist of various databases based on various knowledge technologies. Similarly, a goal base might be based on multiple technologies, but we do not discuss this possibility explicitly in this paper since it seems less useful to us. However, the the same techniques we propose for handling belief base multiple belief bases could be applied also in this situation.

The issues that are introduced by relaxing this assumption can be illustrated after introducing some basic definitions.

First, we introduce a belief operator $\mathbf{bel}(\phi)$ and goal operator $\mathbf{goal}(\phi)$ and associated semantics which express that an agent has a belief or goal ϕ in a mental state $M = \langle \Sigma, \Gamma \rangle$. These operators enable the expression of conditions on mental states of an agent. Formally, a *mental state condition* is a boolean combination of belief and goal conditions, i.e.

$$m ::= \mathbf{bel}(\phi) \mid \mathbf{goal}(\phi) \mid \neg m \mid m \wedge m$$

The semantics of simple belief and goal conditions is defined next. In the standard way these can be extended to handle boolean combinations.

Definition 2. (Semantics of Mental State Conditions)
Let $M = \langle \Sigma, \Gamma \rangle$ be a mental state. Then the semantic clauses for **bel** *and* **goal** *are provided by:*

- $M \models \mathbf{bel}(\phi)$ *iff* $\Sigma \models \phi$
- $M \models \mathbf{goal}(\phi)$ *iff* $\exists \gamma \in \Gamma$ *s.t.* $\gamma \models \phi$ *and* $\Sigma \not\models \phi$.

According to this definition, an agent has a belief ϕ if ϕ is entailed by the agent's belief base and a goal ϕ if and only if ϕ is entailed "locally" from its goal base (i.e. from one of the agent's goals in its goal base) but does not follow from its belief base. What is important in this context to note is that this semantic clause requires us to both verify whether ϕ is entailed by the goal base as well as the belief base.

Second, we introduce a transition rule which defines the operational ("execution step") semantics for GOAL agents. This transition rule defines when the agent can perform an action. The execution of an action involves updating the agent's mental state and to formally define it we need a *transition function* $\mathcal{T}(\mathtt{a}, \Sigma)$. Presumably, this transition function can be defined in terms of the update operators associated with the knowledge representation language used to specify the belief base, e.g. we could define the function by $\mathcal{T}(\mathtt{a}, \Sigma) = \Sigma \oplus \psi$ where ψ is the postcondition of action $\mathtt{a}$.

Definition 3. (Action Execution Rule)
Let $\langle \Sigma, \Gamma \rangle$ be a mental state, c be a conditional action of the form **if** φ **then** $\mathtt{a}$ *where φ is a mental state condition. Then the execution of the conditional action c is defined by:*

$$\frac{\langle \Sigma, \Gamma \rangle \models \varphi}{\langle \Sigma, \Gamma \rangle \longrightarrow \langle \Sigma', \Gamma' \rangle}$$

where: $\Sigma' = \mathcal{T}(\mathtt{a}, \Sigma)$, and $\Gamma' = \Gamma \setminus \{\gamma \in \Gamma \mid \Sigma' \models \gamma\}$.

This rule defining the semantics of action execution also "implements" the blind commitment strategy discussed above. In the absence of other facilities to modify goals, an agent will drop a goal γ only if it is believed to be achieved (i.e. $\Sigma \models \gamma$). This automatic update of the goal base requires that each goal in the goal base is checked against the belief base to verify if it has been achieved (though in practice more efficient implementations are possible).

The semantics introduced above enables us to introduce the issues raised by introducing multiple knowledge technologies into a rational agent more precisely. First, by allowing the belief base and the goal base to be based on different knowledge technologies (but still assuming each uses a *single* technology) we need a means to relate these technologies. The reason is that Definition 3 requires an agent to verify whether its goals γ have been achieved by verifying whether they are entailed by its belief base Σ. Definition 3 thus requires that a query γ specified using one knowledge technology can be resolved using a database Σ based on another knowledge technology.

Second, by allowing a belief base to consist of multiple databases (i.e. $\Sigma = D_1 \times \ldots \times D_n$) using different knowledge technologies the question arises what it means to query such a belief base. It may be useful to decompose an agent's belief base into several databases in practice, e.g. to integrate legacy databases, but how does the agent derive a conclusion that requires *combining information* from such a distributed set of databases?

Finally, the semantics of both Definition 2 and 3 pose certain requirements for the knowledge representation technologies used for goal bases. The point is that both definitions require that a goal base can be viewed as a *set*, either to check whether an element of the goal base entails some formula, or to remove achieved goals from the goal base (Df. 2). Though most logic-based knowledge technologies do support such a view not all do so naturally. For example, it is not clear how a Bayesian network could be viewed as a set, and, in practice, even Prolog systems allow for multiple occurrences of facts. Although the latter issue is easily circumvented, the use of Bayesian networks to represent goals in a goal base is practically excluded. In the remainder, we will assume that goal bases are always implemented with a KRT that allows for set-theoretic view of its associated databases.

There are several options to deal with the issues discussed above. It is unlikely that there is one and only one unique best solution for handling these issues. Our strategy here therefore will be to discuss some of the more promising options. We do not claim to discuss an exhaustive list of options. Our main interest is in the specific issues that are raised when dealing with the problem of combining different knowledge representations in the context of agent programming and our objective is to provide some generic solutions to be able to usefully combine knowledge representations into a rational agent.

A natural first suggestion is that if it would be possible to somehow translate one knowledge representation into another one. In that case, below we show that some of the issues specific to rational agents can be solved using translation operators. In the next section some variations on this topic are explored.

A second suggestion explored in this paper is to use so-called *bridge rules* to connect knowledge stored in various databases (or contexts) and derive a new conclusion from those knowledge sources much in the spirit of multi-context logic [9]. This technique is discussed in Section 5.

4 A Translation Approach to Combine KRT's

In this section, we assume that the belief and goal bases of an individual agent are represented using different knowledge representation technologies. This translation approach will be applied to define the semantics of the GOAL language to fit multiple KRT's. This approach is based on the assumption that the expressions of one knowledge representation language can be translated to expressions of the second language by means of a translation operator.

Definition 4. (translation operator) *Let L_1 and L_2 be two knowledge representation languages. A translation operator τ from L_1 to L_2 is a function from L_1 to L_2. The translation operator can be defined on sets of formula as follows: $\tau(\{\phi_1, \ldots, \phi_n\}) = \{\tau(\phi_1), \ldots, \tau(\phi_n)\}$.*

A translation operator can be used to connect knowledge representation technologies with each other if their entailment relations and update operators impose the same structures on the set of language expressions.

Definition 5. (KRT translation operator) *Let $\mathcal{K}_1 = \langle L_1, \models_1, \oplus_1 \rangle$ and $\mathcal{K}_2 = \langle L_2, \models_2, \oplus_2 \rangle$ be two knowledge representation technologies and $\tau^{\mathcal{K}_1 \to \mathcal{K}_2} : L_1 \to L_2$ be a translation operator. We write τ instead of $\tau^{\mathcal{K}_1 \to \mathcal{K}_2}$ if it is clear that $\tau : L_1 \to L_2$.*

τ is a KRT translation operator from $\mathcal{K}_1$ to $\mathcal{K}_2$ iff

- $\forall \Lambda \subseteq L_1, \forall \phi \in L_1 \ : \ \Lambda \models_1 \phi \ \rightarrow \ \tau(\Lambda) \models_2 \tau(\phi)$
- $\forall \Lambda \subseteq L_1, \forall \phi \in L_1 \ : \ \tau(\Lambda \oplus_1 \phi) \ = \ \tau(\Lambda) \oplus_2 \tau(\phi)$

In this paper, we will use a particular knowledge representation technology which is based on a propositional language, its corresponding well-known entailment relation and an update operator. Using this specific knowledge representation technology, we can study some logical properties of other knowledge representation technologies and investigate their behaviors when they are used in agent programming languages.

Definition 6. (Logically founded KRT) *Let $\mathcal{K}_p = \langle L_p, \models_p, \oplus_p \rangle$ be the propositional knowledge representation technology, where L_p is the language of propositional logic, $\models_p$ is its corresponding entailment relation, and $\oplus_p$ is an update function that satisfies some reasonable belief update postulates, amongst which the consistency preservation property.*

Let $\mathcal{K} = \langle L, \models, \oplus \rangle$ be an arbitrary knowledge representation technology. $\mathcal{K}$ is called logically founded if and only if there exists a KRT translation operator τ from $\mathcal{K}_p$ to $\mathcal{K}$. Moreover, we say $\Lambda \subseteq L$ is τ-consistent only if $\tau(\Lambda)$ is consistent.

The choice of propositional language in the above definition is not strict. In fact, we may use an expressive but computational subset of predicate logic or KIF [15]. We use the propositional language to simplify the presentation of the relevant part of our approach. In the rest of this section, we use the translation operator and present two ways to adapt the semantics of the GOAL programming language.

4.1 Intermediate KRT Translation Approach

One approach is to assume that an agent programming language comes with a propositional knowledge representation technology without assuming how the belief and goal bases are represented. The proposition knowledge representation technology is used to express the query and update expressions. For example, in the GOAL programming language, the propositional formulae are used to implement the pre- and post-conditions of actions without making any assumption on the belief and goal languages. We call such a programming language *generic*.

In particular, one and the same language for query expressions (e.g., a propositional language L_p) is assumed, whereas the representation languages used to represent the belief and goal bases may differ from each other and from the generic language L_p embedded in the agent language. The entailment relation for the propositional language L_p is well-known. Moreover, various update operators are studied for propositional language and some postulates are proposed that should be valid for such operators. For example, if we consider only propositional atoms and their negations, then an update operator can be defined in terms of addition and deletion of atoms.

Additionally we assume that the knowledge representation technologies used for beliefs and goals are logically founded. This implies that there exists a KRT translation operator that maps propositional expressions to the expressions of the languages used in the knowledge representation technologies. In order to illustrate this approach, we apply it to the GOAL programming language. The semantic clauses of the GOAL programming language as defined above can be modified to allow for the integration of multiple knowledge representation technologies.

Definition 7. (Semantics for Generic GOAL) *Let* $\mathcal{K}_b = \langle L_b, \models_b, \oplus_b \rangle$ *and* $\mathcal{K}_g = \langle L_g, \models_g, \oplus_g \rangle$ *be logically founded KRT, based on KRT translation operator* τ_b *and* τ_g*, respectively. Let also* $M = \langle \Sigma, \Gamma \rangle$ *be a mental state with* $\Sigma \subseteq L_b$*,* $\Gamma \subseteq L_g$*,* $\phi \in L_p$ *be a proposition, and* $c =$ **if** φ **then** $\mathtt{a}$ *be a conditional action. Let* $\psi \in L_p$ *be a proposition representing the postcondition for action* $\mathtt{a}$*. The semantics of the generic GOAL language can be defined as follows:*

- $M \models \mathbf{bel}(\phi)$ *iff* $\Sigma \models_b \tau_b(\phi)$
- $M \models \mathbf{goal}(\phi)$ *iff* $\exists \gamma \in \Gamma : \gamma \models_g \tau_b(\phi)$ *and* $\Sigma \not\models_b \tau_b(\phi)$
- *Action execution:*

$$\frac{\langle \Sigma, \Gamma \rangle \models \varphi}{\langle \Sigma, \Gamma \rangle \longrightarrow \langle \Sigma', \Gamma' \rangle}$$

where: $\Sigma' = \Sigma \oplus_b \tau_b(\psi)$

$\Gamma' = \Gamma \setminus \{\tau_g(\psi) \in \Gamma \mid \Sigma' \models_b \tau_b(\psi)\}$

The semantics of the GOAL programming language as defined above has some interesting properties. In particular, despite using different knowledge representation technologies for belief and goal bases, it can be shown that when executed the agent's belief base remains consistent if the initial belief base of the agent is consistent. Moreover, it can be shown that the agent will never have goals that are already achieved.

Proposition 1. *Let* $\mathcal{K}_b = \langle L_b, \models_b, \oplus_b \rangle$ *and* $\mathcal{K}_g = \langle L_g, \models_g, \oplus_g \rangle$ *be logically founded KRT based on* τ_b *and* τ_g*, respectively. Let* $\langle \Sigma_0 \subseteq L_b, \Gamma_0 \subseteq L_g \rangle$ *be an agent's initial state, and* $\langle \Sigma_i, \Gamma_i \rangle$ *(for* $i > 0$*) be a state generated by executing the agent according to the semantics as defined above. Then,*

- *if* Σ_0 *is* τ_b*-consistent then* Σ_i *is* τ_b*-consistent for* $i > 0$
- *if* $\Sigma_i \models_b \tau_b(\phi)$ *then* $\Gamma_i \not\models_g \tau_g(\phi)$ *for* $\phi \in L_p$ *and* $i > 0$.

An advantage of this approach is that agent programs, which are implemented in the generic version of the GOAL programming language, are independent from the employed knowledge representation technologies. Consequently, changing the employed knowledge representation technologies requires only a modification of the translation operators such that nothing needs to be changed in the agent programs. Furthermore, an agent program can be designed before a final choice for a specific knowledge representation technology is made. A disadvantage is that we should specify the translation operator in terms of the set of belief queries which can be a large set.

4.2 Direct KRT Translation Approach

In this subsection, we assume that the adapted agent programming language is defined in terms of two distinct knowledge representation technologies, one to implement the belief base and its corresponding query expressions and one to implement the goal base and its corresponding query expressions. The idea is thus to represent each mental attitude (goals and beliefs) and its corresponding queries with one and the same knowledge representation technology. In this approach, the knowledge representation technologies form integral constituents of the definition of the agent programming language. We illustrate this approach by applying it to the GOAL programming language. As the query languages depend on the knowledge representation technologies, we first redefine the syntax of the GOAL programming language by allowing expressions of different knowledge representation technologies to be used as query expressions.

Definition 8. (Syntax for Multiple KRTs) *Let* L_b *and* L_g *be representation languages for belief and goal expressions, respectively. Let* $\Sigma \subseteq L_b$ *and* $\Gamma \subseteq L_g$*. The GOAL programming language based on Multiple KRT's can be defined as follows:*

- *'if φ then* `a`*', where*
  - *if* $\mathbf{bel}(\phi)$ *occurs in* φ*, then* $\phi \in L_b$
  - *if* $\mathbf{goal}(\phi)$ *occurs in* φ*, then* $\phi \in L_g$
  - $PostCondition(\mathtt{a}) \in L_b$

Here we assume a translation operator that translates L_g into L_b. Given such translation operator, the semantics for the GOAL programming language can be redefined as follows.

Definition 9. (Semantics for Multiple KRTs) *Let* $\tau : L_g \rightarrow L_b$*,* $\mathcal{K}_b = \langle L_b, \models_b, \oplus_b \rangle$ *and* $\mathcal{K}_g = \langle L_g, \models_g, \oplus_g \rangle$*. Let also* $M = \langle \Sigma, \Gamma \rangle$ *be a mental state with* $\Sigma \subseteq L_b$*,* $\Gamma \subseteq L_g$*,* $\phi_b \in L_b$*,* $\phi_g \in L_g$*, and* $c = \mathbf{if}\ \varphi\ \mathbf{then}\ \mathtt{a}$ *be a conditional action. Let* $\psi \in L_b$ *be the postcondition for action* `a`*. The semantics of the GOAL programming language based on Multiple KRT's can be defined as follows:*

- $M \models \mathbf{bel}(\phi_b)$ *iff* $\Sigma \models_b \phi_b$
- $M \models \mathbf{goal}(\phi_g)$ *iff* $\exists \gamma \in \Gamma : \gamma \models_g \phi_g$ *and* $\Sigma \not\models_g \tau(\phi_g)$
- *Action execution:*

$$\frac{\langle \Sigma, \Gamma \rangle \models \varphi}{\langle \Sigma, \Gamma \rangle \longrightarrow \langle \Sigma', \Gamma' \rangle}$$

where: $\Sigma' = \Sigma \oplus_b \psi$

$\Gamma' = \Gamma \setminus \{\psi \in \Gamma \mid \Sigma' \models_b \tau(\psi)\}$

Like the previous approach, it is shown that the consistency of the belief base can be preserved and its relation with the goal base can be maintained.

Proposition 2. *Let* $\mathcal{K}_b = \langle L_b, \models_b, \oplus_b \rangle$ *be a KRT,* $\mathcal{K}_g = \langle L_g, \models_g, \oplus_g \rangle$ *be logically founded based on* τ_p *and let* τ *be a KRT translation operator from* L_g *to* L_b*. Let also* $\langle \Sigma_0 \subseteq L_b, \Gamma_0 \subseteq L_g \rangle$ *be an agent's initial state, and* $\langle \Sigma_i, \Gamma_i \rangle$ *(for* $i > 0$*) be a state generated by executing the agent according to the agent semantics as defined above. Then,*

- *if* Σ_0 *is* τ_p*-consistent then* Σ_i *is* τ_p*-consistent for* $i > 0$*.*
- *if* $\Sigma_i \models_b \tau(\phi)$ *then* $\Gamma_i \not\models_g \phi$ *for* $\phi \in L_g$ *and* $i > 0$*.*
- $\mathcal{K}_b = \langle L_b, \models_b, \oplus_b \rangle$ *is logically founded.*

An advantage of this approach is that only a translation has to be made for the goals of the agents, which are known at design time. A disadvantage is that if the employed knowledge representation technologies for the goal or belief bases are changed, not only a new translation function has to be defined, but also the code of the agent program should be updated.

5 Integrating Multiple KRTs into a Belief Base

Although in principle the techniques of translating sentences specified using different KRTs also can be applied to handle inferencing on a composed belief base that consist of multiple belief bases (cf. previous section and e.g. [15]), we propose

another technique to deal with such inferencing. One reason is that translation may work well only for certain application types that use relatively small knowledge bases. Another reason is that we believe that the technique to handle multiple KRTs should facilitate drawing conclusions that *combine* information from several of the databases a belief base may be composed of. As a simple example, consider the airport service robot again which this time needs to give lost luggage back to a passenger. The robot will need to combine information from several databases to derive the quickest way to do this. A Prolog-like query to obtain this information might look like `loc(luggage,L,passenger),loc(passenger ,P),route(L,P,R)`. A translation approach that has to deal with a query like this would give rise to redundant processing and search for the right source of information that can answer (part) of the query. It is a priori not clear from the query itself to identify the right database to pose (part of) the query to. A technique is needed that allows an agent programmer to "guide" the reasoning of an agent.

The approach suggested in this section proposes to connect various knowledge bases by means of so-called *bridge rules*. Instead of translating languages, the main idea of bridge rules is to add additional *inference power* on top of the two or more knowledge technologies that are to be integrated into the agent application. The mechanism to do so should also provide a means to connect pieces of knowledge represented by different knowledge technologies. The relation suggested by calling these rules bridge rules with *multi-context logic* is intentional [9]. Multi-context logic provides a framework that can be used to achieve our objective to integrate various knowledge technologies in the sense of Definition 1 (cf. also [8] for a similar proposal).

Bridge rules are particular kind of inference rules. They sanction an additional inference to a conclusion represented using one knowledge technology given available inferences and associated conclusions using other knowledge technologies. More formally, a bridge rule can be defined as a rule of the following form:

$$\varphi_1, \ldots, \varphi_n \;\Rightarrow\; \psi$$

where each φ_i and ψ are representations from a particular knowledge representation language $\mathcal{L}$. The intended semantics is that a bridge rule allows the inference of ψ if all φ_i can be derived somehow given the inference relations $\models_{\mathcal{K}_i}$ associated with each φ_i. A bridge rule thus sanctions the inference of ψ given these other inferences, and allows ψ to be used in other inferences to draw certain conclusions again. It does not require such inferences to be made, nor does it require any updates on knowledge bases or the like; these rules only provide additional inference power.

Continuing the example of the service robot, suppose information about passengers is stored "ad hoc" in the robots' belief bases implemented in Prolog, lost luggage information is stored in a SQL database, and routing information may be requested from a GIS system implemented using OO database technology. In that case, a bridge rule could be used to compute a route by directing queries to these various information sources by a rule that such as the following:

$loc(passenger, P)$,
`SELECT` L `FROM`
$LostLuggage$ `WHERE` $Pgnr = passenger$,
$mapGIS.get\_route(L, P, R)$
$\Rightarrow route(R)$

It will be clear that the syntax of the bridge rules provides clues how to resolve a particular (part of a) query.

The idea thus is to allow a programmer to add specific bridge rules to an agent program to facilitate inferences using multiple knowledge technologies. The programmer is supposed to be able to design such rules given his knowledge about the application and the use that the various knowledge technologies have been put to. Bridge rules only add additional inference power and give rise to a new inference relation $\models^*$. The inference relation $\models^*$ defines when a query $\phi \in \mathcal{L}$ from some knowledge representation language $\mathcal{L}$ is entailed by multiple knowledge bases using various knowledge technologies which are possibly related by a set of bridge rules $\mathcal{B}$.

Definition 10. (Induced Inference Relation)
Let a set of knowledge bases $KB_1, \ldots, KB_n$ with associated knowledge technologies $\mathcal{K}_i = \langle \mathcal{L}_i, \models_i, \oplus_i \rangle$ for $i = 1, \ldots, n$ be given. Furthermore, assume a set of bridge rules $\mathcal{B}$ consisting of formulas of the form $\varphi_1, \ldots, \varphi_m \Rightarrow \psi$ with φ_i, ψ each taken from one of the knowledge representation languages $\mathcal{L}_i$. Then the induced inference relation $\models^*$ is defined by:

$$KB_1, \ldots, KB_n, \mathcal{B} \models^* \phi \text{ iff } \exists i : 1 \leq i \leq n \wedge KB_i^* \models_i \phi$$

where the KB_i^* are defined by simultaneous induction as the smallest set such that:

- *$KB_i \subseteq KB_i^*$, and*
- *whenever $\varphi_1, \ldots, \varphi_m \Rightarrow \psi \in \mathcal{B}$ with $\psi \in \mathcal{L}_i$ and for all $j = 1, \ldots, m$ there is a k such that $KB_k^* \models_k \varphi_j$, then $\psi \in KB_i^*$.*

The semantics indicates that each knowledge base with an additional set of bridge rules can be computed incrementally, and that bridge rules can be viewed as a kind of completion operator. An implementation using backward chaining would make this approach a practical option for integration into agent languages.

It should be clear that a translation approach and an approach using bridge rules do not exclude each other. In fact, both can be used to address the issue discussed in the previous section - to facilitate inference when a belief base and goal base use different KRTs - as well as the issue discussed in this section - handling inference in a composed belief base. Bridge rules thus can be viewed as kind of a translation operators but provide a programmer with more flexibility whereas the approach using translation operators is more generic.

6 Conclusion and Related Work

The paradigm of rational agents and multi-agent systems provides an integrative view on a multitude of topics in AI. Agents can usefully exploit the entities to strengths of various technologies, especially w.r.t. knowledge representation and control. To our knowledge, the problem of integrating heterogeneous knowledge bases in a single agent system arose in the agent-oriented programming community only recently. Most state-of-the-art agent oriented programming frameworks do prescribe employment of a single knowledge base in a fixed KR language. Most of the time it is either a logical language (*Prolog*), or a programming language in which the particular framework is developed (*Java*). Homogeneous KBs in such systems do not pose a problem, as formulas of different KBs come from the same language, hence the same entailment/update operators can be used with them.

We are aware of only two efforts in the context of agent oriented programming which aimed at mixing heterogeneous knowledge representations in a single agent system. Project *IMPACT* [7] aimed at integration of heterogeneous legacy knowledge bases accessible to an agent. *IMPACT* treats each underlying KB as an opaque *body of software code*, modeled as a set of predefined functions providing access to the underlying data objects capturing a part of the current agent's (mental) state. The agent logic program consists of a set of *if-then-else* rules regarded as a logic program. However, as *IMPACT* did aim for integration of heterogeneous information sources in the first place, *IMPACT* agents, by default, do not maintain any stronger semantic conditions on their knowledge bases (such as e.g. blind commitment strategy). That implies no special need for translation of formulas from different KBs. In terms of approaches introduced in this paper, *IMPACT* can be seen as an instance of a system implementing a mechanism similar to bridge-rules discussed in Section 5. *Modular BDI architecture* [14] is another recent attempt to approach combining heterogeneous KR technologies in an BDI-inspired agent system. Even though the agent system dynamics and semantics differs from that of *IMPACT*, the approach to integration of heterogeneous KBs in a single agent is very similar.

In this paper we explored several approaches to integrate various knowledge representation technologies so that these can be exploited in a single agent system in a consistent way. This is as an initial attempt to study the problem. We believe that an implemented proof of concept for the presented integration approaches is necessary. Moreover, in our future research we want to compare our approach with the results regarding translating database schemes, such as [11].

We would like to emphasize that the use of propositional logic as an intermediary knowledge representation technology was for simplicity reasons and in order to focus on the problem of integration of knowledge representation technologies. We believe that for developing practical agent systems the propositional knowledge representation technology can easily be extended with first-order elements (such as variables) or even with representation technologies as developed in KIF [15].

References

1. Bordini, R., Hübner, J., Vieira, R.: Jason and the Golden Fleece of agent-oriented programming. In: Multi-Agent Programming - Languages, Platforms and Applications. Springer, Heidelberg (2005)
2. Bordini, R.H., Dastani, M., Dix, J., Seghrouchni, A.E.F.: Multi-Agent Programming Languages, Platforms and Applications. Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 15. Kluwer Academic Publishers, Dordrecht (2005)
3. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about datalog (and never dared to ask). IEEE Trans. of KDE 1(1) (1989)
4. Dastani, M., Meyer, J.-J.C.: A Practical Agent Programming Language. In: Dastani, M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS, vol. 4908, pp. 107–123. Springer, Heidelberg (2008)
5. Davis, R., Shrobe, H.E., Szolovits, P.: What is a knowledge representation? AI 14(1), 17–33 (1993)
6. de Boer, F., Hindriks, K., van der Hoek, W., Meyer, J.-J.: A Verification Framework for Agent Programming with Declarative Goals. Journal of Applied Logic (2007)
7. Dix, J., Zhang, Y.: IMPACT: A Multi-Agent Framework with Declarative Semantics. In: Multi-Agent Programming - Languages, Platforms and Applications. Springer, Heidelberg (2005)
8. Farquhar, A., Dappert, A., Fikes, R., Pratt, W.: Integrating Information Sources Using Context Logic. In: Knoblock, C., Levy, A. (eds.) Information Gathering from Heterogeneous, Distributed Environments (1995)
9. Giunchiglia, F., Serafini, L.: Multilanguage hierarchical logics or: How we can do without modal logics. AI 65(1), 29–70 (1994)
10. Levesque, H.: Foundations of a functional approach to knowledge representation. AI 23, 155–212 (1984)
11. Makowsky, J.A., Ravve, E.V.: Translation schemes and the fundamental problem of database design. In: Thalheim, B. (ed.) ER 1996. LNCS, vol. 1157, pp. 5–26. Springer, Heidelberg (1996)
12. Minsky, M.: A framework for representing knowledge. In: Haughland, J. (ed.) Mind Design, pp. 95–128. MIT Press, Cambridge (1981)
13. Minsky, M.: The society of mind. Simon & Schuster, Inc., New York (1986)
14. Novák, P., Dix, J.: Modular BDI architecture. In: Nakashima, H., Wellman, M.P., Weiss, G., Stone, P. (eds.) Proc. AAMAS 2006, pp. 1009–1015. ACM, New York (2006)
15. Patil, R.S., Fikes, R.E., Patel-Schneider, P.F., McKay, D., Finin, T., Gruber, T.R., Neches, R.: The DARPA knowledge sharing effort: Progress report. In: Rich, C., Nebel, B., Swartout, W. (eds.) Princ. of KR and Reasoning: Proc. of the Third Int. Conf. Morgan Kaufmann, San Francisco (1992)
16. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A BDI Reasoning Engine. Multiagent Systems, Artificial Societies, and Simulated Organizations. In: [2], ch. 6, vol. 15, pp. 149–174 (2005)
17. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a BDI-architecture. In: Proc. of the 2nd Int. Conf. on Princ. of KR and Reasoning, pp. 473–484 (1991)
18. Winikoff, M.: JACK(TM) Intelligent Agents: An Industrial Strength Platform. Multiagent Systems, Artificial Societies, and Simulated Organizations. In: [2], ch. 7, vol. 15, pp. 175–193 (2005)

Model-Checking Strategic Ability and Knowledge of the Past of Communicating Coalitions

Dimitar P. Guelev[1] and Catalin Dima[2]

[1] Institute of Mathematics and Informatics, Bulgarian Academy of Sciences
gelevdp@math.bas.bg
[2] Laboratory of Algorithms, Complexity and Logic, University XII of Paris, France
dima@univ-paris12.fr

Abstract. We propose a variant of alternating time temporal logic (*ATL*) with imperfect information, perfect recall, epistemic modalities for the past and strategies which are required to be uniform with respect to distributed knowledge. The model-checking problem about *ATL* with perfect recall and imperfect information is believed to be unsolvable, whereas in our setting it is solvable because of the uniformity of strategies. We propose a model-checking algorithm for that system, which exploits the interaction between the cooperation modalities and the epistemic modality for the past. This interaction allows every expressible goal φ to be treated as the epistemic goal of (eventually) establishing that φ holds and thus enables the handling of the cooperation modalities in a streamlined way.

1 Introduction

Alternating time temporal logic (*ATL*, [AHK97, AHK02]) was introduced as a reasoning tool for the analysis of strategic abilities of coalitions in infinite multiplayer games with temporal winning conditions. Several variants of *ATL* have been proposed in the literature. The main differences arise from various restrictions on the considered games such as the players' information on the game state, which may be either *complete* or *incomplete* (*imperfect*), and their ability to keep complete record of the past, which is known as *perfect recall* [JvdH04, Sch04]. The awareness of coalitions of the existence of winning strategies is another source of differences, which is specific to the case of incomplete information. The completeness of a proof system for *ATL* with complete information and the decidability of validity in it was demonstrated in [GvD06]. Notably, model checking is believed to be undecidable for *ATL* with imperfect information and perfect recall. The result is attributed to a personal communication of Mihalis Yannakakis to the authors of [AHK02]; this reference has been borrowed in [Sch04] too. This undecidability recall has stimulated the introduction of several systems [Sch04, vOJ05, JvdH06] with restrictions leading to more feasible model-checking. An extensive study of the complexity of the model checking problem

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 75–90, 2009.

for the variants of *ATL* which arise from allowing imperfect information and/or perfect recall was done in [DJ08].

The formal analysis of multi-agent systems has generated substantial interest in the study of combinations of *ATL* with modal logics of knowledge [vdHW03, JvdH04]. Such combinations can be viewed as related to temporal logics of knowledge (cf. e.g [HFMV95]) in the way *ATL* is related to computational tree logic *CTL*. Epistemic goals make it essential to study strategic ability with incomplete information. Variants of the cooperation modalities which correspond to different forms of coordination within coalitions were proposed in [JvdH04]. The recent work [JÅ07] proposes a combination of *ATL* with the epistemic modalities for collective knowledge. In that system formulas are interpreted at *sets* of states and the existence of strategies which are winning for all the epistemically indiscernible states can be expressed by combining epistemic and cooperation modalities. Such strategies are called *uniform* with respect to the corresponding form of collective knowledge.

Along with the alternating transition systems proposed in [AHK02], *ATL* has been given semantics on *interpreted systems*, which are known from the study of knowledge-based programs [HFMV95], and other structures, some of which have been shown to be equivalent [GJ04]. Most of the proposed extensions of *ATL* and other temporal logics by epistemic modalities include only the future temporal operators and the indiscernibility relations which are needed for the semantics of the $S5$ epistemic modalities are either defined as the equality of current local states of the corresponding agents or assumed to be given explicitly in the respective structures and required to respect equality of local state [LR06a, LR06b]. The axiomatisation of knowledge in the presence of past temporal operators has been studied in [FvdMR05], where indiscernibility is defined as equality of local state again. *ATL* with complete information can be regarded as an extension of *computation tree logic CTL*. CTL^* with past modalities was given a complete axiomatisation in [Rey05]. Model-checking an extension of *LTL* by epistemic modalities, including common knowledge, with perfect recall semantics, but no past temporal operators in the language, has been studied in [vdMS99]. We also note the paper [SG02], which gives a model-checking algorithm for a variant of CTL with knowledge (no common knowledge), by a reduction to the chain logic with equal level predicate [Tho92]. Extensions of *CTL* by modalities to reason about indiscernibility with respect to path observations in the past have been proposed in [AČC07]. The model-checking problem for a corresponding more expressive system of μ-calculus has been found to be undecidable. A system of *CTL* with past whose set of temporal modalities is closest to that of the system which we propose in this paper was introduced and studied in [LS95, LS00].

In this paper we propose a variant of *ATL* with epistemic modalities which can be applied to past formulas to allow the formulation of epistemic goals. We assume incomplete information and perfect recall and choose a variant of the meaning of the cooperation modalities of *ATL* in a way which renders the model-checking problem decidable. We achieve this by requiring strategies to be uniform with respect to the distributed knowledge of the coalition and assuming

that strategies are functions on the combined local state of all the members of the coalition. This corresponds to the unrestricted sharing of information within the coalition while implementing the coalition strategy. Requiring uniform strategies can spoil the *determinacy* of games: it is possible that neither side knows how to win without taking chances. However, the impossibility to prevent one's opponent from achieving something using a uniform strategy still means that the opponent has a possibly non-uniform strategy. Notably, some games with imperfect information for just two players are solvable, and our technique for model-checking uniform strategies bears similarities with that from [CDHR07]. That is why the reason for the undecidability under the standard (non-uniform) interpretation of *ATL* with imperfect information and perfect recall appears to be the fact that, even though the members of a coalition are assumed to be working towards a common goal, each agent's strategy is supposed to use only its own observations on the evolution of the system. With strategies that are uniform with respect to distributed knowledge and allow the agents to act using their combined knowledge a coalition can be viewed as a single player whose abilities and information are a combination of those of all the members. We allow past formulas which are interpreted on finite histories in the scope of epistemic modalities. The corresponding indiscernibility relations are defined as equality of local state *throughout the past* and not just of current local state, which is the most marked difference from the majority of the systems known from the literature. Our model-checking algorithm exploits the interaction between uniform strategies and knowledge, which can be formulated in the logic thanks to the presence of both epistemic and strategic modalities: in our setting any strategic goal φ can be formulated as the goal of *(eventually) establishing that* φ holds, which is an epistemic goal. The respective strategies, of course, incorporate the effort of *making* the original goal φ hold.

Structure of the paper. After brief preliminaries on *ATL* and its semantics on interpreted systems we introduce our extension of *ATL* by a modality for distributed knowledge of the past and our variant of the cooperation modalities. Then we propose a transformation of interpreted systems which enables the elimination of any given finite set of formulas built using the epistemic modality, and establish some properties of the new variants of the cooperation modalities which, in turn, allow the formulation of all strategic goals as corresponding epistemic goals. Finally we combine these technical results into a model checking algorithm for the proposed system of *ATL*.

2 Preliminaries

2.1 Interpreted Systems

Definition 1 (Interpreted systems). *Given a set of agents* $\Sigma = \{1, \ldots, n\}$ *and a set of atomic propositions AP, an interpreted system is a tuple of the form* $\langle \langle L_i, Act_i, P_i, t_i \rangle : i \in \Sigma \cup \{e\}, I, h \rangle$ *where*

- L_i *is the set of the* local states *of agent* $i \in \Sigma \cup \{e\}$*;* $e \notin \Sigma$ *stands for the* environment*;*
- Act_i *is the set of* actions *available to agent* i*;*
- $P_i : L_i \to 2^{Act_i}$ *is the* protocol *of agent* i*;*
- $t_i : L_i \times L_e \times Act_1 \times \ldots \times Act_n \times Act_e \to L_i$ *is the* local transition function *of agent* i*;*
- $I \subseteq L$ *is the set of* initial *global states, where* $L = L_1 \times \ldots \times L_n \times L_e$ *is the set of all* global states*;*
- $h : AP \to 2^L$ *is the valuation function.*

Informally, $P_i(l)$ is the set of actions which are available to agent i when its local state is l and $t_i(l, l_e, a_1, \ldots, a_n, a_e)$ is the local state of i after a transition in which the actions chosen by the agents and the environment are $a_1, \ldots, a_n, a_e$, respectively.

A *run* of an interpreted system $IS = \langle\langle L_i, Act_i, P_i, t_i\rangle : i \in \Sigma \cup \{e\}, I, h\rangle$ is an infinite sequence $r = r^0 r^1 \ldots r^k \ldots \in L^\omega$ such that $r^0 \in I$ and for every $k < \omega$, if $r^k = \langle l_1, \ldots, l_n, l_e\rangle$ and $r^{k+1} = \langle l'_1, \ldots, l'_n, l'_e\rangle$, then there exist some $a_1 \in P_1(l_1), \ldots, a_n \in P_n(l_n), a_e \in P_e(l_e)$ such that $l'_i = t_i(l_i, l_e, a_1, \ldots, a_n, a_e)$, for $i \in \Sigma \cup \{1, \ldots, n, e\}$. In words, an interpreted system evolves by each agent choosing an available action at every step and the successor local states of the agents being determined by their respective current local states and all the actions which were simultaneously chosen by the agents and the environment. The environment behaves as an agent, except that, along with its actions, its local state too has direct influence on the evolution of the local states of all the agents.

The local state of agent i within global state $l \in L$ is denoted by l_i. Furthermore, $l' = t(l, a_1, \ldots, a_n, a_e)$ denotes $l'_i = t_i(l_i, l_e, a_1, \ldots, a_n, a_e)$ for all $i \in \Sigma \cup \{e\}$.

2.2 *ATL* on Interpreted Systems

Syntax. *ATL* formulas are built from propositional variables p from some given *vocabulary* AP and sets of agents Γ within some given set of agents Σ using the following syntax:

$$\varphi ::= \bot \mid p \mid \varphi \Rightarrow \varphi \mid \langle\!\langle \Gamma \rangle\!\rangle \circ \varphi \mid \langle\!\langle \Gamma \rangle\!\rangle (\varphi \mathsf{U} \varphi) \mid [\![\Gamma]\!] (\varphi \mathsf{U} \varphi)$$

Semantics. The semantics of the $\langle\!\langle \Gamma \rangle\!\rangle$-modalities of *ATL* involves strategies for sets of agents. A *strategy* for agent i is a function f of type $L_i^+ \to Act_i$ such that $f(l_0 \ldots l_k) \in P_i(l_k)$ for all $l \in L_i^{k+1}$. Given a set of agents Γ, a global state $l \in L$, and a system of strategies $F = \langle f_i : i \in \Gamma\rangle$, $\mathrm{out}(l, F)$ denotes the set of infinite sequences $r \in L^\omega$ such that $r^0 = l$ and r^{k+1} is reached from r^k by a transition in which the action of each agent $i \in \Gamma$ is $f_i(r_i^0 \ldots r_i^k)$ for all $k < \omega$. Informally, $\mathrm{out}(l, F)$ consists of the behaviours starting at l in which the agents from Γ stick to their respective strategies in F.

Given an interpreted system $IS = \langle\langle L_i, Act_i, P_i, t_i\rangle : i \in \Sigma \cup \{e\}, I, h\rangle$, the modeling relation $\models$ is defined between global states $l \in L$ and *ATL* formulas as follows:

$l \not\models \bot$
$l \models p$ iff $l \in h(p)$
$l \models \varphi \Rightarrow \psi$ iff $l \models \psi$ or $l \not\models \varphi$
$l \models \langle\!\langle \Gamma \rangle\!\rangle \circ \varphi$ iff there exists a system of strategies F for Γ such that $r^1 \models \varphi$ for all $r \in \mathrm{out}(l, F)$
$l \models \langle\!\langle \Gamma \rangle\!\rangle (\varphi \mathsf{U} \psi)$ iff there exists a system of strategies F for Γ such that for every $r \in \mathrm{out}(l, F)$ there exists an $m < \omega$ such that $r^k \models \varphi$ for all $k < m$ and $r^m \models \psi$
$l \models [\![\Gamma]\!] (\varphi \mathsf{U} \psi)$ iff for every system of strategies F for Γ there exists an $r \in \mathrm{out}(l, F)$ and an $m < \omega$ such that $r^k \models \varphi$ for all $k < m$ and $r^m \models \psi$

The other combinations between $\langle\!\langle \Gamma \rangle\!\rangle$ and $[\![\Gamma]\!]$ and the *LTL* modalities $\circ$, $\Diamond$ and $\Box$ are introduced as abbreviations:

$$\langle\!\langle \Gamma \rangle\!\rangle \Diamond \varphi \rightleftharpoons \langle\!\langle \Gamma \rangle\!\rangle (\top \mathsf{U} \varphi),\ [\![\Gamma]\!] \Diamond \varphi \rightleftharpoons [\![\Gamma]\!] (\top \mathsf{U} \varphi),\ [\![\Gamma]\!] \circ \varphi \rightleftharpoons \neg \langle\!\langle \Gamma \rangle\!\rangle \circ \neg \varphi,$$
$$\langle\!\langle \Gamma \rangle\!\rangle \Box \varphi \rightleftharpoons \neg [\![\Gamma]\!] \Diamond \neg \varphi,\quad [\![\Gamma]\!] \Box \varphi \rightleftharpoons \neg \langle\!\langle \Gamma \rangle\!\rangle \Diamond \neg \varphi.$$

3 *ATL* with Knowledge of the Past and Communicating Coalitions

According to the definition of $\models$ for formulas built using a cooperation modality, despite pursuing a common goal, the members of a coalition are supposed to follow strategies which are based on each individual member's observation of the behaviour of the system as represented by its sequence of local states. This appears to render the model-checking problem for *ATL* with incomplete information undecidable. A proof of the undecidability is attributed to a private communication of Mihalis Yannakakis to the authors of [AHK02]; the reference to that communication has been borrowed also in [Sch04]. In this section we introduce a semantics for the cooperation modalities which allows to treat a coalition as an individual player with its abilities being an appropriate combination of the abilities of the coalition members by assuming that they exchange information on their local state while acting as a coalition, in order to coordinate each others' actions. We believe that both this assumption and the assumption of each coalition member having to cope using just its own local state are realistic. We also include knowledge modalities, which can be applied to past *LTL* formulas. Making reference to the past requires the satisfaction relation $\models$ to be defined with respect to global histories r instead of just a current state. Unlike runs, histories include the actions of the agents because the knowledge of the coalition's actions contributes restrictions on the global behaviours which correspond to the coalition's observation.

The proposed logic involves two types of formulas denoted by φ and ψ in the BNFs for their syntax:

$$\varphi ::= \bot \mid p \mid \varphi \Rightarrow \varphi \mid \mathsf{D}_\Gamma \psi \mid \langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \circ \varphi \mid \langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} (\varphi \mathsf{U} \varphi) \mid [\![\Gamma]\!]^{\mathsf{D}} (\varphi \mathsf{U} \varphi)$$
$$\psi ::= \bot \mid \varphi \mid \psi \Rightarrow \psi \mid \overline{\circ} \psi \mid (\psi \mathsf{S} \psi) \mid \mathsf{D}_\Gamma \psi$$

As expected, Γ denotes a subset of Σ in $\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}}$, $[\![\Gamma]\!]^{\mathsf{D}}$ and D_Γ. We use D to denote the epistemic modality of *distributed knowledge*, which is usually introduced in systems with K_i as the basic modalities about the knowledge of individual agents i. Since K_i is equivalent to $\mathsf{D}_{\{i\}}$, and we have little technical need to treat the case of singleton coalitions separately, we use only D in our syntax. The superscript $^{\mathsf{D}}$ to the cooperation modalities is meant to emphasize the uniformity with respect to distributed knowledge within coalitions in their semantics. The "main" type of formulas are those defined by the BNF for φ. The past modalities $\overline{\mathsf{o}}$ and $(.\mathsf{S}.)$ can appear only in the scope of D_Γ, without intermediate occurrences of $\langle\!\langle.\rangle\!\rangle^{\mathsf{D}}$ or $[\![.]\!]^{\mathsf{D}}$.

3.1 Cooperation Modalities for Communicating Coalitions

Let $IS = \langle\langle L_i, Act_i, P_i, t_i\rangle : i \in \Sigma \cup \{e\}, I, h\rangle$ be an interpreted system IS with set of global states L. Let $Act = \prod\limits_{i\in\Sigma\cup\{e\}} Act_i$. Then $r = l^0a^0l^1\ldots l^{m-1}a^{m-1}l^m \in L(Act\,L)^*$ is a *global history* of IS, if $l^0 \in I$ and $a_i^{k+1} \in P_i(l_i^k)$ for all $i \in \Sigma\cup\{e\}$ and $l^{k+1} = t(l^k, a_1^k, \ldots, a_n^k, a_e^k)$ for $k = 0, \ldots, m$. The *length* $|r|$ of r is m. We denote the set of the global histories of IS of length m by $R(IS, m)$ and write $R(IS)$ for $\bigcup\limits_{m=0}^{\infty} R(IS, m)$. Given $r = l^0a^0l^1\ldots l^{m-1}a^{m-1}l^m \in R(IS, m)$ and a coalition Γ, the corresponding *local* history r_Γ of Γ is the sequence $l_\Gamma^0a_\Gamma^0l_\Gamma^1\ldots l_\Gamma^{m-1}a_\Gamma^{m-1}l_\Gamma^m \in L_\Gamma(Act_\Gamma L_\Gamma)^m$ where l_Γ stands for $\langle l_i : i \in \Gamma\rangle$, a_Γ stands for $\langle a_i : i \in \Gamma\rangle$, and L_Γ and Act_Γ are $\prod\limits_{i\in\Gamma} L_i$ and $\prod\limits_{i\in\Gamma} Act_i$, respectively. In case $\Gamma = \emptyset$, l_Γ and a_Γ are the empty tuple $\langle\rangle$. The local histories of the empty coalition are sequences of $\langle\rangle$s. Two histories r, r' are *indistinguishable* to coalition Γ, written $r \sim_\Gamma r'$, if $r_\Gamma = r'_\Gamma$. The definition of $\models$ for formulas built using the cooperation modalities $\langle\!\langle.\rangle\!\rangle^{\mathsf{D}}$ and $[\![.]\!]^{\mathsf{D}}$involves a notion of joint strategies for coalitions which can have internal communication.

Definition 2. *Let IS be as above and $\Gamma \subseteq \Sigma$. A (communicating) strategy for a coalition Γ in IS is a mapping $s : L_\Gamma(Act_\Gamma L_\Gamma)^* \to Act_\Gamma$ such that if the last member of r is l, then $s(r) \in \prod\limits_{i\in\Gamma} P_i(l_i)$.*

The set of outcomes $\mathrm{out}(r, s)$ of a given strategy s for Γ starting from a given global history $r \in R(IS)$ consists of the infinite extensions of r which can be obtained if Γ sticks to the strategy s from the end of r on. The clauses for $\models$ on formulas built using cooperation modalities are as follows:

$r \models \langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}} \circ \varphi$ iff there exists a strategy s for Γ such that for all $r' \in [r]_{\sim_\Gamma}$ and all $l^0a^0l^1\ldots l^ka^k\ldots \in \mathrm{out}(r', s)$ $l^0a^0l^1\ldots l^{|r|}a^{|r|}l^{|r|+1} \models \varphi$

$r \models \langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}}(\varphi\mathsf{U}\psi)$ iff there exists a strategy s for Γ such that for all $r' \in [r]_{\sim_\Gamma}$ and all $l^0a^0l^1\ldots l^ka^k\ldots \in \mathrm{out}(r', s)$ there exists an $m \geq |r|$ such that $l^0a^0l^1\ldots a^{k-1}l^k \models \varphi$ for all $k \in \{|r|, \ldots, m-1\}$ and $l^0a^0l^1\ldots a^{m-1}l^m \models \psi$

$$\begin{aligned} r \models [\![\Gamma]\!]^{\mathsf{D}}(\varphi \mathsf{U} \psi) \text{ iff for all strategies } s \text{ for } \Gamma \text{ there exists an } r' \in [r]_{\sim_\Gamma}, \\ \text{an } l^0a^0l^1 \ldots l^k a^k \ldots \in \mathrm{out}(r', s) \text{ and an } m \geq |r| \\ \text{such that } l^0a^0l^1 \ldots a^{k-1}l^k \models \varphi \text{ for all } k \in \{|r|, \ldots, m-1\} \\ \text{and } l^0a^0l^1 \ldots a^{m-1}l^m \models \psi \end{aligned}$$

Formulas of the forms $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \Diamond \varphi$, $[\![\Gamma]\!]^{\mathsf{D}} \Diamond \varphi$, $[\![\Gamma]\!]^{\mathsf{D}} \circ \varphi$, $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \Box \varphi$ and $[\![\Gamma]\!]^{\mathsf{D}} \Box \varphi$ are introduced as abbreviations just like in *ATL* with standard semantics.

Since the same strategy is supposed to work for all the histories which the considered coalition cannot distinguish from the actual one, a coalition can achieve something in the sense of $\langle\!\langle . \rangle\!\rangle^{\mathsf{D}}$ iff it knows that it can achieve it. As it becomes clear below, it is also true that a coalition can achieve something iff it can eventually establish that it has achieved it, or that it keeps achieving it, in the case of $\Box$-goals.

3.2 Knowledge of the Past

Past *LTL* modalities $\overline{\circ}$ and $(.\mathsf{S}.)$ in the scope of D are used to express properties of the history of behaviour of the considered interpreted system. We call formulas built using just these modalities *past LTL formulas*. The semantics of D is defined in terms of the indistinguishability of global histories to coalitions. The clause for $\models$ for knowledge formulas is as follows:

$$r \models \mathsf{D}_\Gamma \psi \text{ iff } r' \models \psi \text{ for all } r' \in [r]_{\sim_\Gamma}$$

Formulas built using past temporal modalities and propositional connectives have their usual meaning:

$$\begin{aligned} &l^0a^0l^1 \ldots a^{k-1}l^k \not\models \bot \\ &l^0a^0l^1 \ldots a^{k-1}l^k \models p \quad \text{iff } l^k \in h(p) \\ &l^0a^0l^1 \ldots a^{k-1}l^k \models \varphi \Rightarrow \psi \text{ iff either } l^0a^0l^1 \ldots a^{k-1}l^k \not\models \varphi \\ &\qquad\qquad \text{or } l^0a^0l^1 \ldots a^{k-1}l^k \models \psi \\ &l^0a^0l^1 \ldots a^{k-1}l^k \models \overline{\circ}\varphi \quad \text{iff } k \geq 1 \text{ and } l^0a^0l^1 \ldots a^{k-2}l^{k-1} \models \varphi \\ &l^0a^0l^1 \ldots a^{k-1}l^k \models (\varphi \mathsf{S} \psi) \text{ iff there exists an } m \leq k \\ &\qquad\qquad \text{such that } l^0a^0l^1 \ldots a^{m-1}l^m \models \psi \\ &\qquad\qquad \text{and } l^0a^0l^1 \ldots a^{j-1}l^j \models \varphi \text{ for } j = m+1, \ldots, k \end{aligned}$$

We denote the dual of D by P and use $\overline{\Diamond}$, $\overline{\Box}$ and I as abbreviations in the usual way:

$$\mathsf{P}_\Gamma \psi \rightleftharpoons \neg \mathsf{D}_\Gamma \neg \psi, \qquad \overline{\Diamond}\psi \rightleftharpoons (\top \mathsf{S} \psi), \qquad \overline{\Box}\psi \rightleftharpoons \neg\overline{\Diamond}\neg\psi, \qquad \mathsf{I} \rightleftharpoons \neg\overline{\circ}\top.$$

This completes the definition of our variant of *ATL*, which we denote by ATL^P_{D}.

The definition of D_Γ affects the way local state contributes to the agents' knowledge. Instead of an explicit encoding of *the entire memory* of agent i as originally proposed [HFMV95], l_i becomes just the projection of the global state which is *visible* to i. Now agents' understanding of the overall structure of the given interpreted system, including their knowledge of its set of initial

states I and the effect of actions as described by the functions t_i are involved in calculating the global histories r' which are indistinguishable from the actual one, and $\mathsf{D}_\Gamma\psi$ holds if ψ holds at all these runs. This means that, e.g., $\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}} \circ (\mathsf{D}_{\{i\}}\psi \vee \mathsf{D}_{\{i\}}\neg\psi)$ is an expression for *Γ can enforce a transition after which i's local history is sufficient to determine whether the global history satisfies ψ or not*. Furthermore, it is possible that, e.g., $\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}} \circ \mathsf{D}_\Gamma\psi$ $[\![\Gamma]\!]^{\mathsf{D}} \circ \mathsf{P}_\Delta\neg\psi$ are simultaneously true for some ψ and a proper sub-coalition $\Delta \subset \Gamma$. The exact meaning of $\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}} \circ \mathsf{D}_\Delta\psi$ is that Γ is capable of causing the system's next state to be one in which Δ can conclude that ψ holds about the past *from its own observations*, regardless of the fact that reaching this state might have involved unrestricted sharing information between Δ and the rest of the members of the greater coalition Γ. Therefore $\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}} \circ \mathsf{D}_\Delta\psi$ can be read as *in one step Γ can achieve ψ in a way which lets Δ realize that ψ holds*, that is, disregarding the information exchanged in order to coordinate the transition designated by $\circ$.

Facts about the past can include (possibly missed) opportunities to enforce certain properties of behaviours; such facts are expressible by writing $\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}}$-formulas in the scope of past modalities. Our model-checking algorithm below involves restoring to local state the role of explicitly storing all the information *which is relevant to a fixed set of epistemic goals about the past.*

4 Encoding Knowledge of the Past in the Local State

In this section we show how, given a finite interpreted system

$$IS = \langle\langle L_i, Act_i, P_i, t_i\rangle : i \in \Sigma \cup \{e\}, I, h\rangle$$

and a finite set Φ^Γ of past *LTL* formulas for each coalition Γ, one can construct a corresponding (bigger) finite interpreted system $IS_{\langle\Phi^\Gamma:\Gamma\subseteq\Sigma\rangle}$ with its states encoding whatever knowledge Γ can extract on the satisfaction of the formulas from Φ_Γ by observing the evolution of its local state in IS. The transitions of $IS_{\langle\Phi^\Gamma:\Gamma\subseteq\Sigma\rangle}$ correspond to the transitions of IS, but connect states of $IS_{\langle\Phi^i:i\in\Sigma\rangle}$ with appropriately related accounts on the satisfaction of the formulas from Φ^Γ for each coalition Γ in them.

To encode knowledge of the past in the local state we use a guarded normal form for the past formulas from the sets Φ_Γ. A finite set of formulas A is said to be a *full system*, if the formulas from A are pairwise inconsistent and their disjunction $\bigvee A$ is valid.

Lemma 1. *Let π be a past LTL formula. Then there exists a finite set of formulas Φ_π of the form*

$$\theta \wedge \mathsf{I} \vee \bigvee_i \alpha_i \wedge \overline{\circ}\beta_i$$

where θ and the α_is are purely propositional, and the α_is form a full system, such that

- *π has an equivalent in Φ_π;*
- *if $\theta \wedge \mathsf{I} \vee \bigvee_i \alpha_i \wedge \overline{\circ}\beta_i \in \Phi_\pi$, then all the β_is have equivalents in Φ_π.*

Proof. An induction on the construction of π shows that it has an equivalent of the above form with the *modal height* of the β_is in it being no greater than that of π itself. The latter implies that a closure of $\{\pi\}$ under taking the β_is from guarded forms would contain only finitely many pairwise non-equivalent formulas.

Note that no syntactical restriction is imposed on the β_is in the normal form above. For example, one normal form for $(p\mathsf{S}q)$ is

$$q \wedge \mathsf{I} \vee q \wedge \overline{\circ}\top \vee (p \wedge \neg q) \wedge \overline{\circ}(p\mathsf{S}q) \vee (\neg p \wedge \neg q) \wedge \overline{\circ}\bot$$

and $\Phi_{(p\mathsf{S}q)}$ can be chosen to consist of the latter formula and the formulas $\bot \wedge \mathsf{I} \vee \top \wedge \overline{\circ}\bot$ and $\top \wedge \mathsf{I} \vee \top \wedge \overline{\circ}\top$, which are normal forms for $\bot$ and $\top$, respectively.

A more accurate estimate of the size of Φ_π can be obtained by taking a deterministic finite state machine $\langle Q, q_0, \delta, F\rangle$ which recognizes the language $\{r^k \dots r^0 \in L^+ : r^0 \dots r^k \models \pi\}$ and taking Φ_π to consist of the formulas π_q which define in the same way the languages accepted by the finite state machines $\langle Q, q, \delta, F\rangle$ for each $q \in Q$.

In the sequel we assume a fixed Φ_π for every given formula π and, without loss of generality, we assume that $\Phi^\Gamma = \bigcup_{\pi \in \Phi^\Gamma} \Phi_\pi$ for each $\Gamma \subseteq \Sigma$.

Next we show that at each step of the evolution of IS the knowledge of coalition Γ on the satisfaction of the formulas from $\Phi = \Phi^\Gamma$ can be represented as a collection of facts of the form:

> *The current global state of IS is one of the states in X and, for each $l \in X$, if the global state of IS is actually l, then the past satisfies the formulas from Φ which are in Ψ_l.*

That is, the knowledge of Γ can be encoded as the tuple $\langle X, \Psi_l : l \in X\rangle$, where $X \subseteq L$ and $\Psi_l \subseteq \Phi$ for every $l \in X$.

Consider a local history $v^0 b^0 v^1 \dots v^{m-1} b^{m-1} v_m$. The possible corresponding global histories $l^0 a^0 l^1 \dots l^{m-1} a^{m-1} l^m$ satisfy the conditions

$$l^k_\Gamma = v^k \text{ for } k = 0, \dots, m, \text{ and } a^k_\Gamma = b^k \text{ for } k = 0, \dots, m-1.$$

The initial state of the global history can be any of the states from $X^0 = \{l \in I : l_\Gamma = v^0\}$, and if the actual initial state is l, then Ψ^0_l consists of those $\pi \in \Phi$ which have a disjunctive member $\theta \wedge \mathsf{I}$ such that $l \models \theta$. Given $v = v^0$, in the sequel we denote the corresponding $H_0 = \langle X^0, \Psi^0_l : l \in X\rangle$ defined above by $I_{\Gamma,\Phi}(v)$. Let $k < m$ and the knowledge of Γ on Φ corresponding to $v^0 b^0 v^1 \dots v^{k-1} b^{k-1} v^k$ be $H_k = \langle X^k, \Psi^k_l : l \in X^k\rangle$. Then Γ's knowledge $H_{k+1} = \langle X^{k+1}, \Psi^{k+1}_{l'} : l' \in X^{k+1}\rangle$ at $v^0 b^0 v^1 \dots v^k b^k v^{k+1}$ can be derived as follows. The set X^{k+1} of the possible global states is

$$\{l' \in L : l'_\Gamma = v^{k+1}, (\exists l \in X^k)(\exists a \in \prod_{i \in \Sigma \cup \{e\}} P_i(l_i))(a_\Gamma = b^k \text{ and } l' = t(l, a_1, \dots, a_n, a_e)\}$$

To determine $\Psi^{k+1}_{l'}$, $l' \in X^{k+1}$, observe that $\theta \wedge \mathsf{I} \vee \bigvee_s \alpha_s \wedge \overline{\circ}\beta_s \in \Psi^{k+1}_{l'}$ for $l' \in X^{k+1}$ iff $\beta_s \in \Psi_l$ for the only s such that $l' \models \alpha_s$ and all $l \in X^k$ such that $l' = t(l, a_1, \ldots, a_n, a_e)$ for some $a \in \prod_{i \in \Sigma \cup \{e\}} P_i(l_i)$ such that $a_\Gamma = b^k$.

In the sequel, given $H = H_k$, $v = v_k$ and $b = b_k$, we denote H_{k+1} by $T_{\Gamma,\Phi}(H, b, v)$. Given that the current knowledge of Γ on the satisfaction of the formulas from Φ is encoded by H, $T_{\Gamma,\Phi}(H, b, v)$ encodes Γ's knowledge on the satisfaction of the formulas from Φ after a transition by action b which leads to local state v for Γ. Since $X \subseteq \{l \in L : l_\Gamma = v\}$, the local state v can always be determined from $H = \langle X, \Psi_l : l \in X\rangle$. Given H, we denote the corresponding local state by $v_\Gamma(H)$. Now we are ready to define

$$IS_{\langle \Phi^\Gamma : \Gamma \subseteq \Sigma\rangle} = \langle \tilde{L}_\Gamma, \tilde{Act}_\Gamma, \tilde{P}_\Gamma, \tilde{t}_\Gamma : \Gamma \in 2^\Sigma \cup \{e\}, \tilde{I}, \tilde{h}\rangle,$$

in which each coalition from $\Gamma \subseteq \Sigma$ is represented as an agent. We put:

$\tilde{L}_e = L_e$, $\tilde{L}_\Gamma = \{\langle X, \Psi_l : l \in X\rangle : X \subseteq \{l \in L : l_\Gamma = v\}$ for some $v \in L_\Gamma, \Psi_l \subseteq \Phi^\Gamma$ for each $l \in X\}$;

$\tilde{Act}_{\{i\}} = Act_i$, $\tilde{P}_{\{i\}}(H) = P_i(v(H))$ for singleton coalitions $\{i\}$; $\tilde{Act}_\Gamma = \{*\}$, $\tilde{P}_\Gamma(H) = \{*\}$ for non-singleton coalitions Γ;

$\tilde{t}_\Gamma(H, l_e, a_1, \ldots, a_n, a_e) = T_{\Phi^\Gamma,\Gamma}(H, a_\Gamma, \langle t_i(v_i(H), l_e, a_1, \ldots, a_n, a_e) : i \in \Gamma\rangle)$;

$\tilde{I} = \{\langle I_{\Gamma,\Phi^\Gamma}(l_\Gamma) : \Gamma \subseteq \Sigma, l_e\rangle : l \in I\}$;

The set of atomic propositions $\tilde{AP}$ for $IS_{\langle \Phi^i : \Gamma \subseteq \Sigma\rangle}$ extends AP by the fresh propositions $p_{\mathsf{D}_\Gamma \pi}$, $\pi \in \Phi^\Gamma$, $\Gamma \subseteq \Sigma$. For $p \in AP$, $\tilde{h}$ is defined by the equality

$$\tilde{h}(p) = \{\langle H_\Gamma : \Gamma \subseteq \Sigma, l_e\rangle : \langle v_{\{1\}}(H_{\{1\}}), \ldots, v_{\{n\}}(H_{\{n\}}), l_e\rangle \in h(p)\}.$$

For the new propositions we put

$$\tilde{h}(p_{\mathsf{D}_\Gamma \varphi}) = \{\langle\langle X^\Gamma, \Psi^\Gamma_l : l \in X^\Gamma\rangle : \Gamma \subseteq \Sigma, l_e\rangle : \varphi \in \Psi^\Gamma_l \text{ for all } l \in X^\Gamma\}.$$

According to this definition, only agents in $IS_{\langle \Phi^\Gamma : \Gamma \subseteq \Sigma\rangle}$ who correspond to singleton coalitions in IS have proper choice of actions, which is the same as that of the respective individual agents in IS; all other coalitions have just singleton action sets, but have the combined ability of observation of their member agents.

Proposition 1. *Let $r = l^0 a^0 l^1 \ldots l^{k-1} a^{k-1} l^k \ldots$ be a run of IS. Let*

$$\tilde{l}^0 \tilde{a}^0 \ldots \tilde{l}^{k-1} \tilde{a}^{k-1} \tilde{l}^k \ldots$$

be a run of $IS_{\langle \Phi^\Gamma : \Gamma \subseteq \Sigma\rangle}$ with the actions of the singleton coalitions $\{i\}$, $i \in \Sigma$ in $\tilde{a}^k$ being those from a^k and let

$$\tilde{l}^0 = \langle I_{\Gamma,\Phi^\Gamma}(l^0_\Gamma) : \Gamma \in \Sigma, l^0_e\rangle.$$

Then for all $k < \omega$ we have

$$l^k = \langle v_{\{i\},\Phi^{\{i\}}}(\tilde{l}^k) : i \in \Sigma, \tilde{l}^k_e\rangle$$

and

$$l^0 a^0 l^1 \ldots l^{k-1} a^{k-1} l^k \models \mathsf{D}_\Gamma \pi \text{ iff } \tilde{l}^0 \tilde{a}^0 \ldots \tilde{l}^{k-1} \tilde{a}^{k-1} \tilde{l}^k \models p_{\mathsf{D}_\Gamma \pi}.$$

This can be established by a straightforward argument using induction on j. The proposition holds with $\tilde{L}_{\{i\}}$ being just L_i and $\tilde{t}_i$ being defined as t_i on the appropriate arguments in case $\Phi^{\{i\}}$ is empty; agents Γ can be omitted altogether for non-singleton Γ with empty Φ^Γ.

Corollary 1. *Let φ be an ATL^P_{D} formula written in the vocabulary $\tilde{AP}$, then IS satisfies the result $[\mathsf{D}_\Gamma\pi/p_{\mathsf{D}_\Gamma\pi} : \pi \in \Phi^\Gamma, \Gamma \subseteq \Sigma]\varphi$ of substituting the propositional variables $p_{\mathsf{D}_\Gamma\pi}$ by the respective formulas $\mathsf{D}_\Gamma\pi$ in φ iff $IS_{\langle \Phi^\Gamma : \Gamma\subseteq\Sigma\rangle}$ satisfies φ itself.*

Remark 1. The crucial property of *LTL* which enables the technique from this section is Lemma 1. Similar statements apply to quantified propositional *LTL*, the (linear time) modal μ-calculus [Koz83], regular expressions, propositional interval-temporal logic [Mos85], etc. All these systems have the expressive power of (weak) monadic second order logic on the natural numbers, which is greater than that of just *LTL*'s past modalities, and can be used in place of the past subset of *LTL* to define the formulas allowed in the scope of D without any substantial change to the model-checking algorithm described in this paper.

5 Some Properties of the ATL^P_{D} Cooperation Modalities

The previous section describes a method for the elimination of D-formulas by replacing them with dedicated propositional variables in appropriately extended interpreted systems. In this section we describe a method which does the same for formulas built using the cooperation modalities. As it becomes clear in the next section, this allows our model-checking algorithm to work bottom-up by replacing modal formulas with dedicated propositional variables in corresponding extensions of the given interpreted system. Only formulas with cooperation modality-free and D-free arguments need to be considered at each step. In this section we establish the properties of the ATL^P_{D} cooperation modalities which we need for the handling of modal formulas built with them.

Let us fix an interpreted system IS with its components named as previously.

Proposition 2. *Let φ and ψ be boolean combinations of propositional variables. Let $r \in R(IS)$ and $X = \{l \in L : (\exists r' \in R(IS))r' \sim_\Gamma r$ and $r'_{|r|} = l\}$. Then*

$$r \models \langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}}(\varphi \mathsf{U} \psi) \text{ is equivalent to } l \models \langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}} \Diamond \mathsf{D}_\Gamma \overline{\Diamond}(\psi \wedge \neg\overline{\circ}\neg\overline{\Box}\varphi)$$

for all the 0-length histories l consisting of an initial state $l \in X$ of the interpreted system $IS_X = \langle\langle L_i, Act_i, P_i, t_i\rangle : i \in \Sigma \cup \{e\}, X, h\rangle$. Similarly

$$r \models [\![\Gamma]\!]^{\mathsf{D}}(\varphi \mathsf{U} \psi) \text{ is equivalent to } l \models [\![\Gamma]\!]^{\mathsf{D}} \Diamond \mathsf{P}_\Gamma \overline{\Diamond}(\psi \wedge \neg\overline{\circ}\neg\overline{\Box}\varphi)$$

for all $l \in X$ in IS_X.

Informally, this means that a strategy which enforces $(\varphi \mathsf{U} \psi)$ also enables the coalition to eventually learn that $(\varphi \mathsf{U} \psi)$ was enforced. Learning this can as well

be postponed by some steps from the time point at which ψ first happens to be satisfied, and it is indeed possible that the coalition "overlook" several events of satisfying ψ, before being able to deduce that such an event took place.

Proof. The backward implication is obvious. For the forward implication, let s be a strategy for Γ which enforces $(\varphi \mathsf{U} \psi)$ starting from all $r' \in [r]_{\sim_\Gamma}$ in IS. Let r_Γ be the local history for Γ which corresponds to r. Let s' be a strategy for Γ which is defined by the equality $s'(v^0 b^0 v^1 \dots v^k) = s(r_\Gamma \cdot b^0 v^1 \dots v^k)$ for all local histories $v^0 b^0 v^1 \dots v^k$ in IS_X. Note that, by the definition of X, all the local histories in IS_X start at the last local state of r_Γ. Assume that $l \not\models \langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \Diamond \mathsf{D}_\Gamma \overline{\Diamond}(\psi \wedge \neg\overline{\circ}\neg\overline{\Box}\varphi)$ for some $l \in X$ for the sake of contradiction. Then this formula is satisfied for no $l \in X$, since, by the definition of X, the local states l_Γ are the same for all $l \in X$ and, consequently, the 0-length histories $l \in R(IS_X, 0)$ are all indistinguishable to Γ. This means that there are arbitrarily big numbers m and $r = l^0 a^0 \dots l^k a^k \dots \in \mathrm{out}(l, s')$ such that $l^0 a^0 \dots a^{m-1} l^m \not\models \mathsf{D}_\Gamma \overline{\Diamond}(\psi \wedge \neg\overline{\circ}\neg\overline{\Box}\varphi)$. By the definition of D_Γ, for such m and r there exist histories $r' \in IR(IS_X, m)$ such that $r' \sim_\Gamma l^0 a^0 \dots a^{m-1} l^m$ and $r' \not\models \overline{\Diamond}(\psi \wedge \neg\overline{\circ}\neg\overline{\Box}\varphi)$. Since s' is observation-based, the r's with the above property are in prefixes of infinite runs in $\mathrm{out}(l', s')$ for some $l' \in X$ as well. Since the finite prefixes of the behaviours from $\mathrm{out}(l', s')$ form trees of finite width, König's Lemma entails that for some $l' \in X$ there is an infinite run $l^{\infty,0} a^{\infty,0} \dots l^{\infty,k} a^{\infty,k} \dots \in \mathrm{out}(l', s')$ such that $l^{\infty,0} a^{\infty,0} \dots a^{\infty,m-1} l^{\infty,m} \not\models \overline{\Diamond}(\psi \wedge \neg\overline{\circ}\neg\overline{\Box}\varphi)$ for all m. By the definition of s', $r' \cdot a^{\infty,0} l^{\infty,1} \dots l^{\infty,k} a^{\infty,k} \dots \in out(r', s)$ for some $r' \in [r]_{\sim_\Gamma r}$ such that r' has l' as its last state. Hence s does not enforce $(\varphi \mathsf{U} \psi)$ starting from r', which is a contradiction. The proof about $[\![\Gamma]\!]^{\mathsf{D}}(.\mathsf{U}.)$-formulas is similar.

Similar statements apply to formulas built using $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \circ$ and $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \Box$:

Proposition 3. *The following equivalences are valid in* ATL^P_{D}*:*

$$\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \circ \varphi \Leftrightarrow \langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \circ \mathsf{D}_\Gamma \varphi \tag{1}$$

$$\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \Box \varphi \Leftrightarrow \langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \Box \mathsf{D}_\Gamma \varphi \tag{2}$$

The proofs are similar, though simpler, because $\Box$ and $\circ$ express safety properties and therefore it is not necessary to use König's Lemma. Cutting of the proper past part of the satisfying behaviour, which is done by moving from IS to IS_X, and is required in the case of $(.\mathsf{U}.)$ because of the need to satisfy it from the end of that behaviour on, is not needed either. This makes it possible to express the connection between the satisfaction of $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \circ \varphi$, $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \Box \varphi$ and the corresponding formulas with D_Γ applied to their designated arguments as a straightforward equivalence.

6 Model-Checking ATL^P_{D}

Now we are ready to describe a model-checking algorithm for ATL^P_{D}. The algorithm works by a series of transformations of the given interpreted system. The

transformations have the forms described in Section 4 and Propositions 2 and 3. The number of transformations is equal to the $\langle\!\langle . \rangle\!\rangle^{\mathsf{D}}$-*depth* $d(\varphi)$ of the given formula φ, which is defined by the clauses

$$
\begin{array}{l}
d(\bot) = d(p) = 0;\\
d(\varphi \Rightarrow \psi) = d((\varphi \mathsf{S} \psi)) = \max\{d(\varphi), d(\psi)\};\\
d(\overline{\circ}\varphi) = d(\varphi);\\
d(\mathsf{D}_\Gamma \varphi) = d(\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \circ \varphi) = d(\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \Box \varphi) = d(\varphi) + 1;\\
d(\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} (\varphi \mathsf{U} \psi)) = d([\![\Gamma]\!]^{\mathsf{D}} (\varphi \mathsf{U} \psi)) = \max\{d(\varphi), d(\psi)\} + 1.
\end{array}
$$

Note that the past modalities $\overline{\circ}$ and $(\mathsf{S}.)$ have no effect on d; the reasons for this become clear below.

Unless the given formula φ is modality-free, which renders the model-checking problem trivial, φ has either

(i) a subformula $\mathsf{D}_\Gamma \psi$ with no occurrences of the cooperation modalities,
or

(ii) a subformula of one of the forms $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \circ \varphi$, $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} (\varphi \mathsf{U} \psi)$, $[\![\Gamma]\!]^{\mathsf{D}} (\varphi \mathsf{U} \psi)$ and $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \Box \varphi$ in which φ and ψ are modality-free.
Despite that $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \Box \varphi$ is just an abbreviation for $\neg [\![\Gamma]\!]^{\mathsf{D}} \Diamond \neg \varphi$, we consider it separately in this case distinction because, as it becomes clear below, it can be handled more efficiently. The better efficiency justifies treating $[\![\Gamma]\!]^{\mathsf{D}} \Diamond \varphi$ as its equivalent $\neg \langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \Box \neg \varphi$ as well.

Assume that (i) holds. Let

$$\Phi^\Gamma = \{\psi : \mathsf{D}_\Gamma \psi \in \mathrm{Subf}(\varphi) \text{ and } \psi \text{ is cooperation modality-free}\}$$

for every $\Gamma \subseteq \Sigma$. Then φ can be written as $[\mathsf{D}_\Gamma \psi / p_{\mathsf{D}_\Gamma \psi} : \psi \in \Phi^\Gamma, \Gamma \subseteq \Sigma]\varphi'$ for some appropriate φ' which has no D-subformulas without a cooperation modality in their scope. Then by Corollary 1 IS satisfies φ iff $IS_{\langle \Phi^\Gamma : \Gamma \subseteq \Sigma \rangle}$, which is defined as in Section 4, satisfies φ'. Clearly $d(\varphi') = d(\varphi) - 1$.

Now assume that (ii) holds. For the sake of simplicity, we assume that there is just one subformula of the considered form. If there are more, then the transformations below can be done for all of them simultaneously.

Let the subformula in question be $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} (\varphi \mathsf{U} \psi)$. Note that, since φ and ψ make no reference to the past, the satisfaction of $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} (\varphi \mathsf{U} \psi)$ depends just on the set of states which are the ends of histories r' that Γ cannot distinguish from the actual reference history r. Consider $IS' = IS_{\langle \Phi^\Delta : \Delta \subseteq \Sigma \rangle}$ where $\Phi^\Delta = \emptyset$ for all $\Delta \subseteq \Sigma$. Moving from IS to IS' with Φ^Δs is equivalent to a subset construction for IS. The states of IS' are the sets of indistinguishable states for all the possible coalitions as the states of the new system. The satisfiability of $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} (\varphi \mathsf{U} \psi)$ is preserved in IS' according to Corollary 1. Moreover, we can define $l \models \langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} (\varphi \mathsf{U} \psi)$ for every individual global state l of IS', and, if $l = \langle \langle X_\Delta \rangle : \Delta \in 2^\Sigma \cup \{e\} \rangle$, that would be equivalent to the existence of a strategy for Γ to enforce $(\varphi \mathsf{U} \psi)$ for all the states of IS in $X = X_\Gamma$. According to Proposition 2, this is equivalent to the satisfaction of $\langle\!\langle \Gamma \rangle\!\rangle^{\mathsf{D}} \Diamond \mathsf{D}_\Gamma \overline{\Diamond} (\psi \wedge \neg \overline{\circ} \neg \overline{\Box} \varphi)$ at the system IS'_Y which is obtained by replacing the set of the initial states of IS' with the set $Y = \{\langle \langle X_\Delta \rangle : \Delta \in 2^\Sigma \cup \{e\} \rangle : X_\Gamma = X\}$, which consists of the states

of IS' which are the ends of 0-length histories in IS' that Γ cannot tell apart from the 0-length history l. To model-check $\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}}\Diamond\mathsf{D}_\Gamma\overline{\Diamond}(\psi\wedge\neg\overline{\circ}\neg\overline{\Box}\varphi)$ in IS'_Y, we construct $IS'' = (IS'_Y)_{\langle\Psi^\Delta:\Delta\subseteq\Sigma\rangle}$ where $\Psi^\Delta = \emptyset$ for $\Delta \neq \Gamma$ again, and $\Psi^\Gamma = \{\overline{\Diamond}(\psi\wedge\neg\overline{\circ}\neg\overline{\Box}\varphi)\}$. According to Corollary 1, what remains to be done is to model-check $\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}}\Diamond p_{\mathsf{D}_\Gamma\overline{\Diamond}(\psi\wedge\neg\overline{\circ}\neg\overline{\Box}\varphi)}$ in IS'', which can be done by calculating the appropriate fixpoint just like for $\langle\!\langle\Gamma\rangle\!\rangle\Diamond p_{\mathsf{D}_\Gamma\overline{\Diamond}(\psi\wedge\neg\overline{\circ}\neg\overline{\Box}\varphi)}$ with respect to the standard semantics of $\langle\!\langle\Gamma\rangle\!\rangle(\mathsf{.U.})$ in ATL.

The steps for formulas of the forms $\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}}\circ\varphi$ and $\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}}\Box\varphi$ are similar, with the role of Proposition 2 being played by the equivalences (1) and (2), respectively. Since there is no need to replace the set of the initial states as done upon moving from IS' to IS'_Y in the case of $\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}}(\mathsf{.U.})$, a single extension of the form from Proposition 1 with Φ^Γ being $\{\varphi\}$ is sufficient.

If the formula in question is $[\![\Gamma]\!]^{\mathsf{D}}(\varphi\mathsf{U}\psi)$, then we use the equivalence between $[\![\Gamma]\!]^{\mathsf{D}}\Diamond\mathsf{P}_\Gamma\overline{\Diamond}(\psi\wedge\neg\overline{\circ}\neg\overline{\Box}\varphi)$ and $\neg\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}}\Box\mathsf{D}_\Gamma\overline{\Box}\neg(\psi\wedge\neg\overline{\circ}\neg\overline{\Box}\varphi)$ and solve the case by model-checking the latter formula at IS'_Y, which is defined as in the case of $\langle\!\langle\Gamma\rangle\!\rangle^{\mathsf{D}}(\varphi\mathsf{U}\psi)$.

This concludes the description of our model-checking procedure for ATL^P_{D}.

7 Conclusion

We have proposed a system of ATL with imperfect information, perfect recall, an epistemic modality for past LTL formulas and cooperation modalities which are interpreted over strategies that are uniform with respect to the distributed knowledge on the past of the respective coalition. The model-checking problem for the proposed system is decidable. The system can be used to specify goals which combine enforcing conditions on the future behaviour of the given system with the acquisition of knowledge or the prevention of acquisition of knowlegde on its past behaviour. Our model-checking algorithm exploits the interaction between the epistemic modality and the cooperation modalities in order to encode all strategy goals as epistemic goals and works by transforming the model-checked system in a way which allows the relevant knowledge of the past to be encoded in the local states of the respective agents and coalitions and thus eliminate the explicit occurrences of the epistemic modality from the model-checked formula.

Acknowledgements

Work on this paper was done during the D. Guelev's visit to LACL at University XII of Paris in October-November 2007, and was partially supported by the ANR SPREADS (Safe P2P Reliable Architecture for Data Storage) project.

The authors are grateful to Pierre Yves Schobbens and Valentin Goranko for some discussions and helpful comments on the topic of the paper and to an anonymous referee for pointing to the relevance of [LS00].

References

[AČC07] Alur, R., Černý, P., Chaudhuri, S.: Model checking on trees with path equivalences. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 664–678. Springer, Heidelberg (2007)

[AHK97] Alur, R., Henzinger, T., Kupferman, O.: Alternating-time Temporal Logic. In: Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, pp. 100–109 (1997)

[AHK02] Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM 49(5), 1–42 (2002)

[CDHR07] Chatterjee, K., Doyen, L., Henzinger, T.A., Raskin, J.F.: Algorithms for Omega-regular Games with Imperfect Information. Logical Methods in Computer Science 3(4), 1–23 (2007)

[DJ08] Dix, J., Jamroga, W.: Model Checking Abilities of Agents: A Closer Look. Theory of Computing Systems 42(3), 366–410 (2008)

[FvdMR05] French, T., van der Meyden, R., Reynolds, M.: Axioms for Logics of Knowledge and Past Time: Synchrony and Unique Initial States. In: Advances in Modal Logic, vol. 5, pp. 53–72. King's College Publications (2005)

[GJ04] Goranko, V., Jamroga, W.: Comparing Semantics for Logics of Multi-agent Systems. Synthese 139(2), 241–280 (2004)

[GvD06] Goranko, V., van Drimmelen, G.: Decidability and Complete Axiomatization of the Alternating-time Temporal Logic. Theoretical Computer Science 353(1-3), 93–117 (2006)

[HFMV95] Halpern, J., Fagin, R., Moses, Y., Vardi, M.: Reasoning about Knowledge. MIT Press, Cambridge (1995)

[JÅ07] Jamroga, W., Ågotnes, T.: Constructive knowledge: what agents can achieve under imperfect information. Journal of Applied Non-Classical Logics 17(4), 423–475 (2007)

[JvdH04] Jamroga, W., van der Hoek, W.: Agents That Know How to Play. Fundamenta Informaticae 63(2-3), 185–219 (2004)

[JvdH06] Jamroga, W., van der Hoek, W.: Strategic Ability under Uncertainty. Technical Report 6, Institute of Computer Science, Clausthal University of Technology (2006)

[Koz83] Kozen, D.: Results on the propositional μ-calculus. Theoretical Computer Science 27, 333–354 (1983)

[LR06a] Lomuscio, A., Raimondi, F.: Model checking knowledge, strategies, and games in multi-agent systems. In: Proceedings of the 5th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS 2006), pp. 161–168. ACM Press, New York (2006)

[LR06b] Lomuscio, A., Raimondi, F.: The Complexity of Model Checking Concurrent Programs Against CTLK Specifications. In: Baldoni, M., Endriss, U. (eds.) DALT 2006. LNCS, vol. 4327, pp. 29–42. Springer, Heidelberg (2006)

[LS95] Laroussinie, F., Schnoebelen, P.: A Hierarchy of Temporal Logics with Past. Information and Computation 148(2), 303–324 (1995)

[LS00] Laroussinie, F., Schnoebelen, P.: Specification in CTL+Past for Verification in CTL. Information and Computation 156(1-2), 236–263 (2000)

[Mos85] Moszkowski, B.: Temporal Logic For Multilevel Reasoning About Hardware. IEEE Computer 18(2), 10–19 (1985)

[Rey05] Reynolds, M.: An Axiomatization of PCTL*. Information and Computation 201(1), 72–119 (2005)

[Sch04] Schobbens, P.Y.: Alternating-time logic with imperfect recall. Electronic Notes in Theoretical Computer Science 85(2), 82–93 (2003); Proceedings of LCMAS 2003

[SG02] Shilov, N.V., Garanina, N.O.: Model-checking Knowledge and Fixpoints. In: Preliminary Proceedings of the 3rd Workshop on Fixpoints in Computer Science (FICS 2002), number NS-02-2 in BRICS Notes Series, pp. 25–39. BRICS (July 2002) (Also available as Preprint 1998, Ershov, A.P., Institute of Informatics Systems, Russian Academy of Sciences (Siberian Division))

[Tho92] Thomas, W.: Infinite Trees and Automation-Definable Relations over ω-Words. Theoretical Computer Science 103(1), 143–159 (1992)

[vdHW03] van der Hoek, W., Wooldridge, M.: Cooperation, Knowledge and Time: Alternating-time Temporal Epistemic Logic and Its Applications. Studia Logica 75, 125–157 (2003)

[vdMS99] van der Meyden, R., Shilov, N.V.: Model Checking Knowledge And Time In Systems With Perfect Recall. In: Pandu Rangan, C., Raman, V., Ramanujam, R. (eds.) FST TCS 1999. LNCS, vol. 1738, pp. 432–445. Springer, Heidelberg (1999)

[vOJ05] van Otterloo, S., Jonker, G.: On Epistemic Temporal Strategic Logic. In: Proceedings of the 2nd International Workshop on Logic and Communication in Multi-Agent Systems (2004). ENTCS, vol. 126, pp. 77–92. Elsevier, Amsterdam (2005)

JASDL: A Practical Programming Approach Combining Agent and Semantic Web Technologies

Thomas Klapiscak and Rafael H. Bordini

Department of Computer Science
University of Durham, U.K.
{T.G.Klapiscak,R.Bordini}@durham.ac.uk

Abstract. Although various ideas for integrating Semantic Web and Agent Programming techniques have appeared in the literature, as yet no practical programming approach has managed to harness the full potential for declarative agent-oriented programming of currently widely used Semantic Web technologies. When agent programmers are familiar with existing ontologies for the domain of interest, they can take advantage of the knowledge already represented there to make their programs much more compact and elegant, besides the obvious interoperability and reuse advantages. This paper describes JASDL: an extension of the Jason agent platform which makes use of OWL-API to provide features such as plan trigger generalisation based on ontological knowledge and the use of such knowledge in querying the belief base. The paper also includes a running example which clearly illustrates the features and advantages of our approach.

1 Introduction

The idea of agent-oriented programming with underlying ontological reasoning was first put forward in [19]. That paper showed the changes in the operational semantics of AgentSpeak that were required for combining agent-oriented programming with ontological reasoning. The Semantic Web vision depends on the availability of ontologies [24] so that web resources are semantically annotated, but depends also on the availability of *agents* that will be able to make use of such semantically enriched web resources. For this to be possible in practice, a well devised combination of autonomous agents and semantic web technologies is essential. This paper aims to contribute towards addressing this problem.

The main advantages for agent programming that were claimed in [19] to result from the work on that variant of AgentSpeak [22] (called AgentSpeak-DL) based on a Description Logic (DL) [1] are:

(*i*) "queries to the belief base are more expressive as their results do not depend only on explicit knowledge but can be inferred from the ontology;
(*ii*) the notion of belief update is refined so that a property about an individual can only be added if the resulting ontology-based belief base would preserve consistency (i.e., if the ABox assertion is consistent with the concept descriptions in the TBox);

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 91–110, 2009.

(*iii*) retrieving a plan (from the agent's plan library) that is relevant for dealing with a particular event is more flexible as this is not based solely on unification, but also on the subsumption relation between concepts; and
(*iv*) agents may share knowledge by using web ontology languages such as OWL [17]."

The four points above were quoted from [19]. However, that was a formal paper, which set the grounds for this work, but was far removed from the actual technologies — such as an AgentSpeak interpreter and ontological reasoners such as [23,15]. As anyone with experience in applied work in multi-agent systems will agree, there are major research questions to solve and technical obstacles to overcome before a theoretical contribution becomes useful for practical work. That is probably the reason why, so far, only one attempt has been made to implement the ideas in [19], at least to our knowledge. The first initial contribution towards implementing those ideas appeared in [7]; there are, however, limitations to that approach which our approach circumvents (as discussed in Section 3).

This paper describes JASDL (***Jason*** AgentSpeak–DescriptionLogic), which uses ***Jason*** [4] customisation techniques (as well as some language constructs such as annotations) and the OWL-API [13] in order to provide all the features of agent programming combined with ontological reasoning mentioned above, as well as some novel features conceived of during the development of JASDL. To our knowledge, JASDL is the first full implementation of an agent-oriented programming language with transparent use of ontologies and underlying ontological reasoning within a declarative setting.

However, while [19] suggested changes in the operational semantics to achieve this, we did not need to change any of the core classes of ***Jason*** in order to implement AgentSpeak-DL (note that JASDL is a ***Jason***-based implementation of AgentSpeak-DL). This is due to the various customisation and extension techniques which have been built into ***Jason*** [4]. This paper shows how such mechanisms were used, the various choices that had to be addressed in order to make concrete the formal proposal for combining agent-oriented programming and ontologies, and exemplifies how the features of such a combination can be useful in software development using JASDL.

The remainder of this paper is organised as follows. In Section 2 we describe JASDL in detail, including a running example which helps illustrate the main features of agent programming in JASDL. In Section 3, we discuss related work and then conclude the paper and mention future work in Section 4.

2 JASDL

2.1 The General Architecture

We now explain the main components of JASDL, how they fit together, and how each corresponds to the enhancements claimed in this paper. A simplified view of the general architecture can be seen in Figure 1. Note that, throughout this paper we will be referring to points *i–iv* of the advantages of combining agent programming with ontologies as quoted in Section 1.

The extensibility mechanisms of ***Jason*** allow the functioning of certain steps in the agent reasoning cycle to be modified by extending core ***Jason*** classes and overriding methods as required. This technique is adopted to integrate the various mechanisms of JASDL with ***Jason***; this is to ensure that JASDL will work with future releases of ***Jason***. We now describe the four main ***Jason*** components that JASDL overrides and to what end it does so.

Belief Base. A ***Jason*** agent stores information about the world within a data structure known as the *belief base*. Initially, the ***Jason*** belief base was simply a list of ground literals, although this has recently been extended to allow the use of Prolog-like rules [4]. JASDL extends this in such away that the belief base now partly resides within a set of in-memory instantiations of OWL ontology schemas. This, in combination with a DL reasoner, facilitates the use of publicly available knowledge (in web ontologies) to increase the extent of inferences an agent can make based on its beliefs (point *i*; see Section 2.3), and an enhanced assurance of knowledge consistency (point *ii*; see Section 2.4).

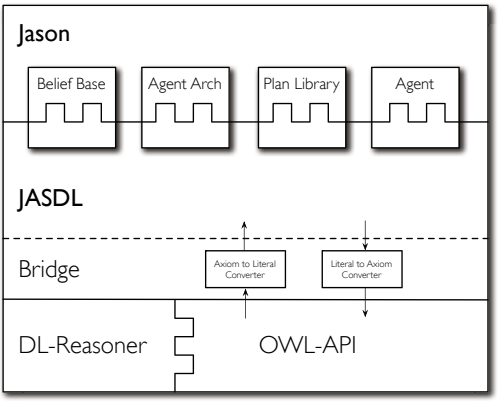

Fig. 1. The Architecture of JASDL

Plan Library. The plan library of a ***Jason*** agent combines a data structure for storing AgentSpeak plans with various modification and retrieval operations. JASDL overrides these operations to facilitate enhanced plan retrieval flexibility to include additional, more *general* (according to ontological knowledge) plans for dealing with an event (point *iii*; see Section 2.8).

Agent Architecture. The "overall agent architecture" provides an interface between the reasoning module of a ***Jason*** agent and the multiagent infrastructure it uses for communication, perception, and action. ***Jason***'s extensibility mechanisms allow us to override certain aspects of this interfacing independently of implementation specifics [4]. In the case of JASDL, we augment the default architecture with message processing to facilitate semantically-enriched inter-agent communication (point *iv*; see Section 2.7).

Agent. We extend ***Jason***'s internal representation of an agent for two reasons. Firstly, we override its configuration routines to include JASDL specific parameters. Secondly, we override various application-dependent functions such as the AgentSpeak select option function, S_O[1], (as part of the implementation of point *iii*) and the agent's belief revision function (as part of the implementation of point *ii*).

The OWL-API can be thought of as a repertoire of data structures and utility classes facilitating a high-level means of representing, modifying and reasoning over

[1] Due to shortness of space, we cannot introduce the AgentSpeak language in this paper. Readers unfamiliar with the language are referred to the cited AgentSpeak literature.

OWL ontologies. Description Logic reasoning services are provided through a general interface of which there are well established implementations for the widely known DL reasoners Pellet [23] and FaCT++ [15]. The OWL-API takes an *axiomatic* approach to representing an OWL ontology, which, as its authors point out, can lead to more elegant implementations than those possible using other approaches (such as the RDF-triple data model adopted by the Jena API [16]). Our experience provides further evidence to that, as a previous JASDL implementation used Jena and it seems to us that the new implementation is significantly clearer. Additionally, the OWL-API exposes various black-box debugging features (presently not supported by Jena to the best of our knowledge) that are necessary for two of JASDL's features which will be discussed later in this paper.

Finally, the *bridge* sub-component of JASDL encapsulates the interfacing between JASDL and the OWL-API. Its primary purpose is to provide various factory classes to conveniently allow the creation of ***Jason*** constructs (such as literals) from the constructs of the OWL-API (axioms), and vice-versa.

2.2 The "Takeaway" Case-Study

Throughout this paper we refer to a simplified version of the "Takeaway" case-study that has been developed using JASDL. Although the code seen within this paper is based upon an actual implementation, it has been heavily adapted in the interest of concise discussion of the most relevant issues. The version we describe is a simple multi-agent system containing three main types of agent: `Customers`, `Takeaways`, and `Companies`. `Customers` act as "personal-assistants" and coordinate with `Takeaways` to purchase types of takeaway food on behalf of a human. `Takeaways` are grouped according to the `Companies` that own them; all those within the same company are assumed to share profits (and thus have a common goal).

The agents in this society make use of two OWL ontologies, *fastfood.owl*[2] and *society.owl*[3], both of which are depicted in Figure 2. We give an overview of each ontology below, whilst deferring a more detailed account until required in later discussions.

The `fastfood.owl` ontology describes a hierarchy of types of fast food. An instance of this hierarchy corresponds to a portion of prepared foodstuff. To describe pizzas, it imports a fragment of the well-known *pizza.owl*[4] ontology. In that ontology, pizzas are classified according to their relationships with instances of `PizzaTopping` over the `hasTopping` object property. On top of that, we have added various classes to describe types of curry and a single data property, `hasPrice` relating all instances of `Fastfood` to a `double` constant value.

The `society.owl` ontology describes a hierarchy of agent types which agents can use to reason about their peers and the society within which they exist. Additionally, we define the `owns` object property used to express agent subservience. Not only does this ontology grant the multi-agent system a shared vocabulary and an expression of social rules, it also acts as a shared data-source since agent instances are defined within the ontology *schema* itself (rather than within local in-memory instantiations as is

[2] Available at `http://www.dur.ac.uk/t.g.klapiscak/onts/fastfood.owl`
[3] Available at `http://www.dur.ac.uk/t.g.klapiscak/onts/society.owl`
[4] Available at `http://www.co-ode.org/ontologies/pizza/2005/10/18/pizza.owl`

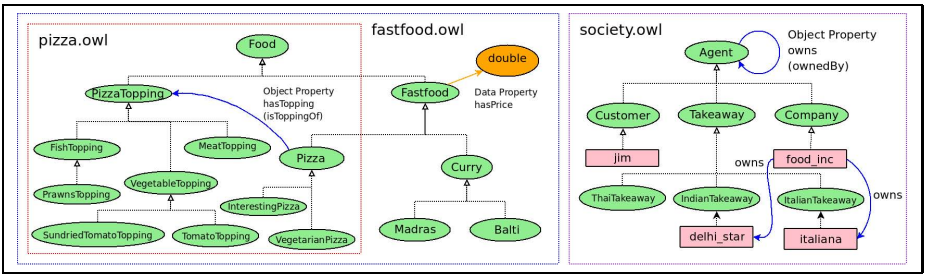

Fig. 2. Ontologies used within the "Takeaway" case-study. Ovals depict classes or data types, rectangles depict instances, dotted lines depict subsumption, and labelled solid lines depict properties (inverses are given in brackets).

the case for instances of `Food`). We define a `Customer` agent `jim`, a `Company` agent `food_inc` and two `Takeaway` agents, `italiana` and `delhi_star`, which have types `ItalianTakeaway` and `IndianTakeaway` respectively.

2.3 More Expressive Queries to the Belief Base

The belief base of a JASDL agent should be seen as being the result of the combination of two reasoning engines and knowledge representation languages. The first is identical to the default ***Jason*** belief base and is used to store normal AgentSpeak beliefs as well as Prolog-like rules that can be used for making inferences over existing beliefs. The second wraps around a set of read/write in-memory instantiations of read-only OWL ontology schemas and is used for storage of those beliefs that correspond to some ontological fact.

Semantically-Enriched Literals. In order to provide a means for the representation of beliefs that correspond to some ontological fact using standard AgentSpeak syntax, we define the notion of a *Semantically-Enriched Literal*, or *SE-literal* for short. An SE-literal is characterised by its association with the *ontology annotation* (o/1) — see [4] for the use of annotations in ***Jason*** literals. The single, atomic term of this annotation is an *ontology label*; a unique, succinct reference to a single in-memory ontology instance. Nested within an SE-literal are one or more terms that define the individual(s) or constant that this assertion is about. The functor of the SE-literal defines the TBox resource the assertion refers to and is simply an AgentSpeak syntax compatible modification of the fragment of the URI that identifies the resource. We refer to *`(functor,label)`* pairs as *aliases* since they provide a level of indirection between SE-literals and ontological resources. For emphasis, a normal ***Jason*** literal that does not have an ontology annotation or alias is referred to as *"semantically-naive"*.

Functor and label mappings can be manually configured for an agent. Additionally, JASDL provides an extensible and configurable *auto-mapping* mechanism to perform common functor mapping operations (*mapping strategies*) across an entire ontology, such as decapitalising the first letter of the URI fragment of each resource if necessary. If an automatic mapping operation results in a non-unique alias, the functor is replaced

with an anonymous one, which can in turn be preemptively overridden by a manual mapping.

In the interests of succinctness, we assume this transformation is implicit in all examples seen in this paper. Further, we consider all agents to share the label "`ff`" for `fastfood.owl` and "`s`" for `society.owl`. The `pizza.owl` ontology is imported by `fastfood.owl` and need not be directly referenced by a label; `ff` can be used to reference resources from this ontology and the specifics are handled transparently by JASDL.

We define four types of SE-literal for the representation of different kinds of assertions within the ontological components of an agent's belief base. We classify an SE-literal implicitly based on its arity and the types of its nested terms.

Class Assertion. This is a *unary* SE-literal corresponding to the assertion that the individual represented by the term given as the parameter is a member of the *class* referenced by the predicate (in the respective ontology).

Object Property Assertion. This is a *binary* SE-literal corresponding to the assertion that the individual represented by the first parameter is related to the second by the *object* property referenced by the predicate.

Data Property Assertion. This is also a *binary* SE-literal. It corresponds to the assertion that the individual represented by the first parameter is related to the constant value given as the second by the *data* property referenced by the predicate.

All Different Assertion. By default, OWL does not make the unique naming assumption [18]. This implies that, unless explicitly stated otherwise, individuals with different names may be treated as identical by a reasoner. When developing an agent using JASDL, we may wish to explicitly state at run-time that a set of individuals are mutually distinct. To this end, we represent such assertions with a unary SE-literal possessing the predefined and global predicate ``` ``all_different'' ```. The parameter of this predicate is a list of individuals that we wish to assert as mutually distinct.

SE-Literal	Axiom
`pizza(pizza1)[o(ff)]`	ClassAssertion(pizza1 Pizza)
`owns(food_inc, italiana)[o(s)]`	ObjectPropertyAssertion(owns food_inc italiana)
`hasPrice(pizza1, 2.2)[o(ff)]`	DataPropertyAssertion(hasPrice pizza1 "2.2")
`all_different([italiana,jim])[o(s)]`	DifferentIndividuals(italiana, jim)

Fig. 3. Example SE-literals and corresponding axiomatic representation

Modification. A belief addition or deletion comes in the form of a ground[5] literal ℓ to be asserted or retracted within the belief base of the agent. In JASDL, these operations can follow one of two flows of execution, determined by intercepting ℓ in our extended belief base ***Jason*** class.

[5] Unground additions fail, whilst unground removals are grounded – using the first unifying literal in the belief base – before removal is attempted.

If ℓ is not semantically enriched, it is simply passed up to the standard ***Jason*** belief base to be handled as per usual. If, on the other hand, ℓ is semantically enriched we proceed as follows. We encode the precise meaning of ℓ into its equivalent OWL-API axiomatic representation. We then assert or retract this encoding within the appropriate ontology instance using the standard mechanisms provided by the OWL-API.

The only complication to this process is related to the storage of annotations associated with ℓ (with the exception of the ontology annotation, since this is implicit in the location of the axiom). ***Jason***'s default belief base, which stores hash-table references to (semantically-naive) literals themselves, handles this implicitly since annotations form part of the literal description. For JASDL this is not so straightforward since our literals are deconstructed for ontological storage and reconstructed upon retrieval. JASDL approaches this issue by exploiting OWL's capacity for annotating axioms with constant values. We simply serialise the literal annotations and apply them to the corresponding ABox axiom ready for retrieval and deserialisation.

Retrieval. A query against the belief base in ***Jason*** amounts to testing whether a supplied literal ℓ is a logical consequence of the beliefs and prolog-like rules contained within the belief base; this is done using standard unification. JASDL augments this with knowledge represented in DL and the use of efficient DL reasoners. As mentioned earlier, the results of a query are no longer formed only of the knowledge available in the standard ***Jason*** belief base, but also of that *inferred* by a DL reasoner operating over the ontology instances known to the agent.

Like the modification operations, processing of a literal ℓ is passed either to the standard ***Jason*** belief base or JASDL's extension depending on the type of literal. As before, ℓ is encoded into its OWL-API axiomatic equivalent. This translation is identical except that, in this case, ℓ can be ground **or** unground. In the former case, ℓ is returned only if it is entailed by the axioms within the ontology instance. In the latter, we return a set of grounded SE-literals corresponding to all possible groundings of ℓ that can be inferred from the ontology. Additionally, literal annotations applied to asserted ABox axioms are deserialised (as discussed above) and added to the appropriate literal before it is returned.

Through extension of the core methods of ***Jason***'s belief base, in combination with the use of SE-literals, we intuitively expose the full power of a DL reasoner to the JASDL agent designer; any inference a DL reasoner is capable of making about individuals can be implicit in the standard AgentSpeak syntax. Additionally, this enhancement is ubiquitous across any AgentSpeak operator that tests logical consequence against the belief base. This includes, for example, test goals, internal actions, and plan contexts when the AgentSpeak interpreter checks for applicable plans.

Figure 4 gives two side-by-side AgentSpeak plan fragments. To the left, we show some belief additions that define the ABox state for the agent in this example. To the right, we show some examples of the DL inferencing power exposed to the agent designer by JASDL. The TBox state is defined as shown in Figure 2.

1. This test goal succeeds for two reasons. Firstly, the domain of `hasTopping` is `Pizza`. Since `pizza1` participates, as part of the domain, in relationships over this property, we can infer that `pizza1` must be of this class. Secondly, by the

```
...                                          ...
+tomatoTopping(t1)[o(ff)];                   /* 1 */ ?food(pizza1)[o(ff)];
+sundriedTomatoTopping(t2)[o(ff)];           /* 2 */ ?isToppingOf(t1, X)[o(ff)];
+prawnsTopping(t3)[o(ff)];                   /* 3 */ ?interestingPizza(pizza1)[o(ff)];
+hasTopping(pizza1, t1)[o(ff)];              /* 4 */ ?~vegetarianPizza(pizza1)[o(ff)];
+hasTopping(pizza1, t2)[o(ff)];              /* 5 */ ?owns(food_inc, Y)[o(s)];
+hasTopping(pizza1, t3)[o(ff)];              /* 6 */ .findall(Thing, thing(Thing)[o(owl)], E);
+all_different([t1, t2])[o(ff)];             ...
+owns(food_inc, italiana)[o(s)];
+owns(italiana, van1)[o(s)];
...
```

Fig. 4. Two plan fragments demonstrating the DL inferencing power exposed by JASDL

transitivity of the subsumption relationship, we can infer that `Pizza` is a sub-class of `Food`.

2. This test goal succeeds, unifying the variable `X` with `pizza1`. This is because these individuals are asserted to participate in the inverse of this relationship (i.e., over `hasTopping`).
3. To be "interesting", a `Pizza` must have at least three **different** toppings. We know this is the case for `pizza1`, since it is related to `t1`, `t2`, `t3` by `hasTopping`. We can infer that `t3` is different to `t1` and `t2` since `FishTopping` (of which `PrawnsTopping` is a sub-class) is asserted to be disjoint to `VegetableTopping` (of which `TomatoTopping` and `SundriedTomatoTopping` are sub-classes of). However, no such inference can be made for `t1` and `t2`. Accordingly, we have **asserted** that the two are different using the `all_different` SE-literal. Consequently, this test goal succeeds.
4. A `VegetarianPizza` is one that does not have any `FishTopping` or `MeatTopping`. Since `pizza1` has `PrawnsTopping`, we can conclude that it cannot be a `VegetarianPizza`, therefore this strongly-negated test goal succeeds.
5. This test goal succeeds, unifying `Y` with either `italiana` or `van1`. This is because the `owns` object property is transitive and we have the asserted chain that `food_inc owns italiana` that `owns van1`.
6. The `.findall` internal action unifies its third argument with a list of groundings of its first argument that renders its second argument a logical consequence of the belief base [4]. Consequently, since `Thing` (whose alias has the label `owl`) is the super-class of all others, this line results in the unification of the variable `E` with a list of all individuals known to the agent (i.e., `[pizza1,t1,t2,t3, delhi_star, ...]`).

2.4 Belief Base Consistency Assurance

As stated by point (*ii*), having the belief base of an agent partially resident within a set of ontologies grants us a refined means for maintaining knowledge consistency when adding a belief to such an ontology. Specifically, we can make use of a DL reasoner to detect when the addition of an assertion within the ABox leads to an inconsistency according to the axioms expressed within the corresponding TBox. Measures can then be taken to restore consistency. Note that since the description logic used to give formal

```
...
+hasTopping(pizza1, t1)[o(ff), source(jim)];
+hasTopping(pizza2, t1)[o(ff), source(self)];
+all_different([pizza1, pizza2])[o(ff), source(self)];
...
```

Fig. 5. A plan fragment demonstrating JASDL's belief base consistency assurance capabilities

semantics to OWL-DL is *monotonic*, only additions to the ontology may lead to the formation of new inconsistencies [10]; thus, we need not worry about removals here.

When faced with an addition of a fact that leads to an inconsistency, we must act to preserve the consistency of the belief base. For this purpose, JASDL implements two alternative mechanisms: *Contradiction Rejection* and *Description-Logic Based Semi-Revision*. Both are implemented within the application-specific *belief revision function* extension point provided by ***Jason***.

We now discuss these mechanisms in more detail. Below, the transformation from SE-literal to its corresponding axiomatic representation is considered implicit; hence the two are interchangeable.

Contradiction Rejection. This mechanism simplifies the process of consistency assurance by making the assumption of *temporal precedence*. This states that beliefs already resident within the belief base take precedence over new, or "incoming" beliefs. As shown in Algorithm 1, we check for belief-base consistency after each new belief addition. If an inconsistency is detected, we simply rollback the addition and notify ***Jason*** accordingly. In this case, we consider the belief addition to have failed, causing the respective AgentSpeak plan to fail accordingly.

Figure 5 shows an AgentSpeak plan fragment that will result in a belief base inconsistency. The property `hasTopping` is asserted to be an *inverse functional* property. That is, it can relate to any particular individual at most once. Asserting that `pizza1` and `pizza2` are distinct is a contradiction because the only way both can validly relate to `t1` over `hasTopping` is if they are the same individual (recall that OWL does not make the unique naming assumption). Accordingly, our mechanism rejects this assertion and fails the plan.

Algorithm 1. Contradiction Rejection

1: $\ell \Leftarrow$ incoming belief
2: $B \Leftarrow$ axioms from agent belief base
3: $B \Leftarrow B \cup \{\ell\}$
4: **if** $B \models \bot$ **then**
5: $\quad B \Leftarrow B/\{\ell\}$
6: $\quad$ **return** *reject*
7: **end if**
8: return *accept*

Description Logic Based Semi-Revision. This mechanism is more complex owing to the avoidance of the temporal precedence assumption. An implication of this is that we

may need to *revise* the belief base by removing resident beliefs in order to accommodate an incoming belief, ℓ, while preserving knowledge consistency. It is known as **semi**-revision because it is still possible to reject the incoming belief addition in the face of inconsistency (as is always the case with Contradiction Rejection).

To implement this mechanism, we draw heavily upon ideas taken from [11]. Due to limited space, we do not give formal definitions for the operators we use. Instead, we align our implementation with their approach and refer the reader to that paper.

The kernel operator $\perp\!\!\!\perp$. This function selects the set of all possible *justifications* for a belief ℓ (or *kernelset*). A justification for ℓ (or ℓ-kernel) is a set of axioms taken from the belief base that imply ℓ. Justifications are minimal in the sense that removal of at least one member *undermines* it causing it to no longer imply ℓ. Thus, undermining all possible justifications for ℓ results in it no longer being a logical consequence of the belief base as a whole. For reasons that will become clear we use a modified form of $\perp\!\!\!\perp$, the *singleton-kernel* operator, $\perp\!\!\!\perp_{single}$, which generates just a single member of the kernelset at a time. We make use of the black-box debugging functionality provided as part of the OWL-API to implement this function.

The ℓ-kernel filter function δ. Given an ℓ-kernel, this function returns a subset representing those axioms which are to be considered "mutable" by our revision mechanism; in other words, those axioms that may be removed to undermine justifications and ultimately accommodate ℓ. JASDL's particular implementation filters out TBox axioms. This is because we do not consider it desirable for our agent to be capable of rendering its own ontology instance *structurally* incompatible with the shared ontology. Future work will explore further the implications and validity of this assumption.

The *incision* function σ. This function chooses a non-empty subset of an ℓ-kernel corresponding to those beliefs that should be retracted to undermine the justification. There may be many different ways of choosing this subset and the specifics are application dependent. JASDL's particular implementation of σ chooses the *least trusted* belief from those available. The standard ***Jason*** *source* annotation gives us the name(s) of the agent(s) responsible for the presence of a belief. Thus, we can establish the degree of trust associated with a belief by performing a look up against a "trust-rating" hash table, maintained by JASDL, that maps known agent names to a numerical trust rating.

The high-level operation of the mechanism is given in Algorithm 2, and is described informally as follows. To begin, ℓ is added to the belief base, perhaps resulting in an inconsistency and thus the invocation of semi-revision. A justification for the inconsistency is generated, to which ℓ is added, allowing this new belief to be selected by the incision function and thus rejected. This set is then filtered and incised, producing a set of beliefs to be retracted. If ℓ belongs to this set, we consider it rejected. Accordingly, we rollback the modifications made by the mechanism thus far and terminate prematurely, causing a ***Jason*** plan failure. If not, we track and retract the chosen revisions. We terminate once consistency has been restored, generating belief addition and deletion events as appropriate.

Since we do not consider TBox axioms mutable, we deal only with a subset of all possible justifications for an inconsistency; it is possible for multiple justifications to exist which differ only in the TBox axioms involved. We avoid generation of these

through use of $\perp\!\!\!\perp_{single}$, which allows us to obtain justifications one at a time and halt as soon as consistency is restored.

It is worth mentioning that the power of this mechanism comes with a high computational cost relative to that incurred by the alternative contradiction-rejection mechanism. Accordingly, the two can be toggled between, both in the agent configuration and at run-time using a special internal action.

Revisiting the example seen in Figure 5, the agent behaves differently when making use of this mechanism. The (filtered) justification generated consists of the three assertions seen here. Supposing we trust assertions made by the agent `jim` less than our own (`self`), our incision function chooses the assertion that `pizza1 hasTopping t1` for retraction, while accepting the assertion that `pizza1` and `pizza2` are distinct. In this case, we have accommodated a new belief by revision and so execution of the plan can continue as normal.

Algorithm 2. Description Logic Based Semi-Revision

1: $R \Leftarrow \{\}$ {Used for tracking revisions so as to rollback in case of belief rejection}
2: $\ell \Leftarrow$ incoming belief
3: $B \Leftarrow$ axioms from agent belief base
4: $B \Leftarrow B \cup \{\ell\}$ {Add incoming belief}
5: **while** $B \models \bot$ **do**
6: $X \Leftarrow B \perp\!\!\!\perp_{single} \bot$ {Apply Singleton Kernel Operator}
7: $X \Leftarrow X \cup \{\ell\}$ {Ensure incoming belief can be rejected}
8: $X \Leftarrow \delta(X)$ {Apply Kernel Filter}
9: $X \Leftarrow \sigma(X)$ {Apply Incision Function}
10: **if** $\ell \in X$ **then**
11: $B \Leftarrow B/\{\ell\}$ {Remove rejected belief}
12: $B \Leftarrow B \cup R$ {Re-establish revised beliefs}
13: **return** *rejected* {Notify ***Jason*** of failure}
14: **else**
15: $R \Leftarrow R \cup X$ {Track revisions}
16: $B \Leftarrow B/X$ {Revise belief base}
17: **end if**
18: **end while**
19: **return** *accepted* {Notify ***Jason*** of success}

2.5 Removal as Contraction

The implementation of the removal operation for an SE-literal, ℓ, as described in Section 2.3, will only remove the **assertion** of ℓ from the ABox. However, since beliefs can now be **inferred** from others according to the rules expressed in the TBox, simply removing the assertion itself is not guaranteed to result in completely abolishing the belief. In fact, there may be multiple justifications for ℓ, all of which need to be undermined to ensure it is no longer an implication of the belief base. Consider, also, that this operation will certainly fail if ℓ corresponds to inferred, rather than asserted, information.

JASDL offers a solution to this issue by allowing the belief base removal operation to be implemented as the *kernel contraction* operator, similar to that defined in [11].

We defer discussion of this feature until now, as it relies upon operators defined in Section 2.4. With these operators in place, implementation of this mechanism becomes straightforward. Informally, we begin by removing the *assertion* made by ℓ (if present). Next, while ℓ is a logical consequence of the belief base, we use the singleton kernel operator to find a justification for ℓ to which we apply the kernel filter and incision functions, the result of which we remove from the belief base.

2.6 Run-Time Class Definition

No feature of JASDL seen thus far allows modification of the TBox component of an ontology instance. Hence, our ABox retrieval capabilities are restricted to querying in terms of existing classes and properties. The only exception to this is when using strongly-negated unary SE-literals, which essentially provide a short-cut to defining the complement of a class at run-time. Beyond this, however, practical experience shows that it is useful for agents to be capable of, at run-time, producing new classes for interaction with the ABox using classes, properties, and logical connectives.

JASDL exposes this functionality in the guise of the "define class" internal action: `jasdl.ia.define_class(functor, expression)`. Its first parameter, *functor*, is an atomic term that will from now on form the predicate symbol of any SE-literal referring to this new class. Its second parameter, `expression`, is a textual description of a class in a variant of the *Manchester OWL Syntax* [14]. This is a concise and easily comprehensible syntax for describing OWL ontologies or fragments thereof, designed specifically with non-logicians in mind. JASDL uses a form of this syntax in which ontological resources are referenced by `label:functor` pairs corresponding to known aliases. In this way we can concisely refer to any resource, including other run-time defined classes, across all known ontologies.

To avoid overlap with *schema*-defined classes, all classes defined locally at run-time are assigned the special ontology label "`self`". This label is assigned a unique (with respect to the known society) URI, enabling it to be referred to unambiguously by communicating agents (see Section 2.7). If an overlap on the `functor` is detected with a previous *run-time* definition, it is simply overridden.

Figure 6 gives an AgentSpeak plan fragment demonstrating the usage of this functionality. In this plan, we first add the assertions that `curry1` is an instance of `Madras` and is related to the value `5.4` by the `hasPrice` data property. Next, we define the class of individuals that are instances of either `Madras` or `Balti` and that are related to some double less than `6.0` by the `hasPrice` data property. The alias `(request, self)` is now assigned to this class. Finally, the test goal results in the unification of `curry1` with the variable `X` since its membership to this class can be inferred owing to our earlier assertions.

```
...
+madras(curry1)[o(ff)];
+hasPrice(curry1, 5.4)[o(ff)];
jasdl.ia.define_class(request, "(ff:madras or ff:balti) and ff:hasPrice some double [< 6.0]");
?request(X)[o(self)];
...
```

Fig. 6. A plan fragment demonstrating JASDL's run-time class definition capabilities

Currently, it is possible to define unsatisfiable classes using this internal action, thus placing the ontology as a whole in an inconsistent state. Future work will explore how we might apply mechanisms similar to the (currently ABox only) belief base consistency assurance mechanisms seen in Section 2.4.

2.7 Semantically-Enriched Inter-agent Communication

As indicated by point (*iv*), the use of ontological reasoning facilitates the sharing of knowledge among agents through the use of standard ontologies for many domains which are increasingly available on the Web. An ontology provides a shared vocabulary between communicating agents. Moreover, a recipient agent can easily be introduced to novel ontologies, thus extending its knowledge of the world as required *at run-time*. The implications of this for Multi-Agent Systems are huge, since the meaning of inter-agent communication can potentially be understood even with no prior agreement on terminology. As contributions from the community doing research on ontological alignment (such as [5]) fully mature, the benefits will be further increased, since two agents will be able to communicate even when using disparate ontologies.

The payload, or *propositional content*, of a message sent by the inter-agent communication mechanism of a ***Jason*** agent can be composed of AgentSpeak literals. JASDL extends this to allow this propositional content to refer to an axiom within the ABox of some ontology. At first sight, it may appear that this can already be achieved simply by making use of SE-literals. While partly true, complications arise when we take into account a JASDL agent's capacity (and need) for local (and largely arbitrary) aliasing. This problem is exacerbated by the fact that agents can themselves locally define new classes (see Section 2.6). Accordingly, in order to ensure the semantic inter-operability of JASDL agents, we must dereference these aliases, thus rendering the corresponding SE-literals contained within a message globally comprehensible (with respect to an ontology) and unambiguous.

A detailed account of this mechanism is beyond the scope of this paper; instead, we give a high-level overview of its operation. The mechanism is invoked upon encountering an outgoing message containing one or more SE-literals (referred to as an *SE-message)*, which has been intercepted by JASDL's customised agent architecture as discussed in Section 2.1. To each SE-literal, ℓ, within this SE-message, we add special annotations (of which all JASDL agents know the meaning), that dereference the locally assigned alias of ℓ thus ensuring the recipient can ascertain its precise meaning. Contained within these annotations are the full URIs of any ontology schemas or resources prerequisite to the understanding of ℓ. Additionally, if ℓ corresponds to a local run-time defined class, these annotations include its normalised (i.e., using only schema-defined resources) rendering in the Manchester OWL Syntax.

For incoming SE-messages (also intercepted in JASDL's extended agent architecture), uninstantiated ontologies are instantiated and assigned anonymous labels, class expressions are compiled, and dereferenced ontologies and resources are mapped back in terms of local aliases.

2.8 Enhanced Plan Retrieval Flexibility

As part of its reasoning cycle, a ***Jason*** agent must establish a set of *relevant* plans for dealing with an *event* resulting from a perceived change in environmental or internal

circumstances. This plan-event relevance is established, in part, through unification of the literals associated with their respective triggers.

Consider that, in JASDL, these literals can have some correspondence to a class or property from the TBox of an ontology. Additionally, this TBox describes a taxonomy relating its members by subsumption. An implication of this is that we now are able to place plans and events whose triggers contain some SE-literal (SE-plans and SE-events respectively) into a taxonomy of identical structure. Thus, for an SE-event it might be possible to obtain a set of plans that are relevant in a *more general sense* by appealing to *subsumption*, when an appropriate *specifically designated plan* cannot be found by unification. This grants us a much more elaborate and precise notion of plan generality than the use of higher-order variables in plan triggers allows (the only currently available means for this in ***Jason***). Hence, a JASDL agent is well equipped to automatically, through re-use of *existing* ontological rules, adopt potentially beneficial courses of action when no specifically designated plans are available. Moreover, this is achieved with little additional effort on the part of the agent designer.

The mechanism for this is defined by [19] in terms of an extension to a rule of the operational semantics of AgentSpeak. Unfortunately, this rule correlates with code within a core class of ***Jason*** for which no extension points are *directly* exposed. Since it is highly desirable that JASDL remains compatible with future ***Jason*** releases, this extension must be performed *indirectly*, which we accomplish by exploiting the extensibility mechanisms built into ***Jason***'s plan library implementation.

We now give a brief overview of our approach to this issue. The code responsible for determining the relevance of a plan to an event is located within ***Jason***'s internal representation of a plan. JASDL extends this for SE-plans by checking, when dealing with an SE-event for which unification has failed, if the plan trigger subsumes that of the event in question. This subsumption detection is not straightforward since we permit comparison of SE-literals nested arbitrarily within semantically-naive ones. Due to space restrictions, we cannot give in this paper a full specification of the algorithm used for this. In high-level terms, however, we consider two literals in parallel, recursively assigning each a specificity "score" according to their respective nestings of SE-literals. We take the literal with highest score at the root of the recursion tree to be the most specific. We substitute these SE-plan representations for their semantically-naive counterparts by interception of plan additions within JASDL's extended plan library.

For the sake of efficiency, ***Jason*** does not perform this full relevancy check against all plans in its library, but rather against a *candidate* subset obtained through a hash-table look up based on the functor and arity of the event trigger. For JASDL, this notion of candidacy is not so amenable to a hash-table look up since it is based also upon ontological rules. Consequently, (additional) candidates are found simply by testing for trigger subsumption across all known SE-plans. Clearly, this is not ideal since we are duplicating work; future research will look at ways of mitigating this added cost.

When a generalisation occurs, the trigger of the plan we have generalised to is replaced by that of the event leading to its generalisation. This allows ***Jason***'s standard unification mechanism to operate correctly when establishing the effect of the plan execution upon the intention stack within which it resides. To ensure we do not modify the actual plan library when we do so, an additional modification is made to the plan

library which causes *cloned* version of candidate SE-plans to be injected into the reasoning cycle.

An important feature of this mechanism is that it assigns precedence to SE-plans according to how specific their corresponding ontological resources are; the rationale behind this is that the more specific a plan is to an event, the better suited it is to deal with it. This is implemented within the application dependent S_O function[6] using the subsumption detection algorithm discussed above.

Practical experience gained during implementation has shown that the original cause of the generalisation is potentially valuable information that we cannot always afford to lose. This is made available in JASDL through provision of the "get trigger generalisation cause" internal action: `jasdl.ia.get_tg_cause(Cause, retain_annots)`. This unifies `Cause` with the event that ultimately resulted in generalisation to this plan. If `retain_annots` is set, the annotations applied to this event are retained, or dropped otherwise (with the exception of the ontology annotation, which is vital to all SE-literals).

We now give an example of a practical scenario that makes use of this functionality. Suppose that the class `request`, as defined earlier in Figure 6, represents a description of a takeaway meal that agent `jim` wishes to purchase on behalf of its human operator. The AgentSpeak plan fragment shown in Figure 7 shows the request made to `italiana`.

```
...
.send(italiana, askOne, request(X)[o(self)]);
...
```

Fig. 7. Sending a request to `italiana`

Upon receipt of this request, `italiana`, being an `ItalianTakeaway`, does not stock any curry and so is unable to find an individual within its belief base that belongs to this class. As standard in ***Jason***, upon failure of a simple unification against the belief base of an agent, a *complex test goal* is generated. This is an internal event intended to result in the adoption of a goal in an attempt to resolve a test goal. In this case, it is of the form `+?request(X)[o(anon_label_0)]`. Notice that JASDL has replaced the `self` ontology label used by `jim` with an anonymous one. It is this label that `italiana` will henceforth use in aliases referencing classes defined at run-time by `jim`.

Figure 8 shows two AgentSpeak plan fragments from the plan library of agent `italiana`. Intuitively, these plans are intended as *general plans* to deal with circumstances under which a request has been made that cannot be directly serviced by this agent. Because of the enhanced plan retrieval flexibility granted by JASDL, and by the inference that `request` is (in part) subsumed by `Curry` and by `Fastfood`, both these plans are considered *relevant* to the complex test-goal event. They are also both applicable, since the trigger of this event (i.e., the original cause of the generalisation, unified with `Cause` using the `jasdl.ia.get_tg_cause` internal action) is **not** a logical consequence of the belief base. The first plan deals with the request of an unknown **general** type of `Fastfood`, in which case we refer it to any other takeaway owned by the same company as `italiana`. We find such an individual by defining its class at run-time and querying against it using a test goal. The second deals with the more **specific** request for an unknown type of `Curry`. Under these circumstances, it would be prudent to refer

[6] In reality, this is implemented as a composable JMCA module allowing it to be used with other S_O functions. See 2.9 for more details on JMCA.

```
+?fastfood(_)[o(ff)] :
    jasdl.ia.get_tg_cause(Cause, false) &
    not Cause
        <-
        jasdl.ia.define_class(
            takeawayInSameCompany,
            "s:takeaway and s:ownedBy value s:food_inc");
        ?takeawayInSameCompany(X)[o(self)];
        .send(X, askOne, Cause);
        ...

+?curry(_)[o(ff)] :
    jasdl.ia.get_tg_cause(Cause, false) &
    not Cause
        <-
        jasdl.ia.define_class(
            indianOrThaiInSameCompany,
            "(s:indianTakeaway or s:thaiTakeaway) and s:ownedBy value s:food_inc");
        ?indianOrThaiInSameCompany(X)[o(self)];
        .send(X, askOne, Cause);
        ...
```

Fig. 8. A fragment of `italiana's` plan library demonstrating *general* plans

the request to an `IndianTakeaway` or a `ThaiTakeaway`. Accordingly, we define such a class of individuals and use it to find a potentially helpful recipient, to whom we then refer the request.

Owing to JASDL's implementation of the S_O function, the most specific of these two plans (`+?curry`), is chosen to deal with the complex test goal event. Consequently, the request is referred to a likely stockist (`delhi_star` in this case), the results of which are passed back to the customer for approval (which is not shown here).

2.9 *Jason* Module Composition Architecture

During the implementation of JASDL, an issue became apparent with the extensibility mechanisms provided by ***Jason*** that allow an agent designer to override the methods that implement the various selection functions of an agent [4]. Conceivably, selection function implementations may be provided as libraries for general use. However, it becomes difficult for an agent designer who wishes to simultaneously make use of multiple such libraries whose implementations overlap, since they are forced to combine code in an ad-hoc fashion (presuming the library source code is available in the first place).

As a solution to this problem, a separate extension to ***Jason*** was devised, namely JMCA [7] (***Jason*** *Module-Composition Architecture*). JMCA permits multiple selection function implementations to interact in a well-defined manner. This is achieved by providing a framework under which an agent designer can encapsulate the implementation of a selection function within a *selection strategy*. Subsequently, an agent making use of JMCA is capable of composing a chain of such selection strategies and specifying a

[7] An initial release of JMCA is available at http://jason.sf.net

mediation strategy to mediate between the choices they make, thus defining the overall behaviour produced by the composition chain and ultimately the choices made by the agent.

As discussed in Section 2.8, the most specific plan is given precedence by JASDL from a generalised set. This is best implemented within the S_O application-specific agent function. However, since JASDL is intended as a development platform for other agents, an agent designer wishing to make use of their own S_O implementation would encounter great difficulty incorporating it with this (important) feature of JASDL. Accordingly, JASDL's S_O function is implemented as a composable JMCA selection strategy, allowing an agent designer to easily and elegantly compose it with their own implementation(s).

3 Related Work

Theoretical foundations for the use of ontologies in agent-oriented programming languages first appeared in [19]. The paper used the formal semantics of AgentSpeak to show a number of features that would result from the use of ontological reasoning, and forms the basis of our work. Similar ideas, restricted to an agent's belief base, and using the Go! language, appeared independently in [6]. However, differently from the main idea in JASDL which is to refer to existing ontologies on the Web at run-time (in particular, using run time instances of such ontologies as extra information available to the agent on top of its current beliefs), and making use of existing (efficient) ontological reasoners, the work in [6] concentrates in showing how to translate an OWL-Lite ontology into Go! so as to use that knowledge as part of an agent's belief. Note, however, that without an explicit link to the ontology as a web resource, and agents being able to change their beliefs, the interoperability aspect that is an important advantage of standard Semantic Web technologies is (possibly) lost.

There is a variety of agent-oriented programming languages [3], some of which have working interpreters with a growing user base in the Multi-Agent Systems community, such as 3APL [8] (and its recent 2APL variant), Jadex [21], SPARK [20], and JIAC [12]; there are also commercial products such as JACK [25]. Some of these platforms allow a reference to an ontology used for agent communication to be given (such as JIAC and Jade [2], and the platforms that use Jade as middleware, such as Jadex, 2APL, and also ***Jason*** itself). However, this is a long way from the functionalities that a full integration (at the declarative level) of agent-oriented programming and ontological reasoning, as first suggested in [19], provides. For example, relevant plans to handle an event can be inferred using ontological relations when an agent is not aware of a specific plan for an event.

An approach combining BDI-like agents and Semantic Web technologies was introduced in [9]. The paper presented an agent architecture called *Nuin* and a scripting language, whose interpreter is based on AgentSpeak, where XML name-space prefixes can be used as part of identifiers, thus allowing particular names to be linked to given ontologies. One disadvantage of the approach is that the BDI agent programming notions are only informally described; by using ***Jason***, in contrast, we are able to take

advantage of the significant body of work formalising the core AgentSpeak constructs used as part of the language interpreted by ***Jason***.

As this paper is concerned with a practical implementation of a combination of an agent programming language and ontology techniques, the closest work to ours is that on Argonaut [7]. Argonaut demonstrates how ontological reasoning can be practically integrated with ***Jason*** to grant some degree of contextual awareness to an agent providing location-based services. The approach taken by Argonaut isolates all ontological interaction within highly specialised, predefined internal actions. JASDL, on the other hand, integrates ontological reasoning more tightly and transparently with the ***Jason*** AgentSpeak interpreter. This has several advantages, some of which are discussed below.

To implement new kinds of ontological interaction, Argonaut requires new *Java* code to be written. JASDL however allows the implementation of new ontological processes at the *AgentSpeak* level. Since JASDL augments the ***Jason*** belief base with the capacity for ontological reasoning, additional inferencing immediately becomes ubiquitous amongst the syntactic operators of AgentSpeak. For example, test goals, plan contexts, predefined internal actions that query the belief base (such as `.findall`), belief additions and deletions, etc., all leverage the combined expressive power and tractability of description logic. Contrast this to the Argonaut approach, under which queries can be made only by means of internal actions made available as part of Argonaut. Plan contexts, for example, cannot be so naturally expressed since they must contain such internal actions.

4 Conclusion and Future Work

In this paper, we have described the implementation of an existing theoretical proposal for combining agent-oriented programming and ontological reasoning. The paper highlights various issues that need to be addressed in practice which are not apparent from the abstract point of view of formal work. The running example used in the paper made reference to an existing ontology on the Web; an important aspect of our approach is precisely to allow programmers to make use of existing knowledge representation to facilitate programming as well as allowing agents to share ontological information.

A recent development in JASDL that we do not have space to discuss in detail here allows agent configuration through the use of OWL ontologies. JASDL agents have, from the start of development, been highly configurable owing to a rich set of parameters that can be specified in the standard ***Jason*** multi-agent system configuration file. The *OWL2MAS* add-on allows for the automatic generation of this based purely upon special OWL ontologies. Applications can extend a standard "skeleton" ontology[8] (or extensions thereof) using the OWL *import* mechanism, adding application-specific parameters and concrete multi-agent system settings. This allows ***Jason*** multi-agent system configurations to be specified in a standard language distributed across the Internet, thus encouraging re-use and reducing redundancy. Moreover, JASDL agents can instantiate these ontologies and use them to reason both about the society they reside within and other ***Jason*** multi-agent systems. Additionally, a framework is included allowing

[8] Available at `http://www.dur.ac.uk/t.g.klapiscak/onts/owl2mas/mas.owl`

parts of the Java code modelling the agents' environment to be instantiated based on this configuration.

We plan a number of other directions for future work, including incorporating means for ontological alignment when an agent detects any ontological mismatch, and facilitating the use of Web services in the context of JASDL by means of OWL-S[9]. Future work will also include mechanisms that permit persistency of the (at present volatile) ontology instances of JASDL agents.

Although everything described in this paper has been implemented and the examples have been executed, it is still very early days for the development of JASDL. Whether its engineering principles will prove natural and useful in programming real-world applications remains to be seen, as does the analysis of performance issues. In the near future, JASDL will be made available *open source*, which should help in our aim to assess its usability and efficiency in practical applications. Nevertheless, our initial experiments show that there is great potential benefit to be obtained from the use of JASDL and performance does not seem to be an issue.

References

1. Baader, F., Calvanese, D., McGuinness, D.N.D., Patel-Schneider, P. (eds.): Handbook of Description Logics. Cambridge University Press, Cambridge (2003)
2. Bellifemine, F., Bergenti, F., Caire, G., Poggi, A.: JADE — a java agent development framework. In: Bordini, et al. (eds.) [3], ch. 5, pp. 125–147
3. Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.): Multi-Agent Programming: Languages, Platforms and Applications. Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 15. Springer, Heidelberg (2005)
4. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak Using *Jason*. Wiley Series in Agent Technology. John Wiley & Sons, Chichester (2007)
5. Castano, S., Ferrara, A., Messa, G.: Islab hmatch results for oaei 2006. In: Proc. of International Workshop on Ontology Matching, held with ISWC 2006, Athens, Georgia, USA (November 2006)
6. Clark, K.L., McCabe, F.G.: Ontology schema for an agent belief store (2005)
7. da Silva, D.M., Vieira, R.: Argonaut: Integrating jason and jena for context aware computing based on owl ontologies. In: Baldoni, M., Baroglio, C., Mascardi, V. (eds.) Proc. of AWESOME 2007 held as part of MALLOW 2007, Durham, September 3–7 (2007)
8. Dastani, M., van Riemsdijk, M.B., Meyer, J.-J.C.: Programming multi-agent systems in 3APL. In: Bordini, et al. (eds.) [3], ch. 2, pp. 39–67.
9. Dickinson, I., Wooldridge, M.: Towards practical reasoning agents for the semantic web. In: Proc. of AAMAS 2003, Melbourne, Australia, July 14-18, 2003, pp. 827–834. ACM, New York (2003)
10. Haase, P., van Harmelen, F., Huang, Z., Stuckenschmidt, H., Sure, Y.: A Framework for Handling Inconsistency in Changing Ontologies. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 353–367. Springer, Heidelberg (2005)
11. Halaschek-Wiener, C., Katz, Y.: Belief base revision for expressive description logics. In: Proc. of OWLED 2006 (2006)

[9] Refer to http://www.w3.org/Submission/OWL-S/.

12. Heler, A., Hirsch, B., Keiser, J.: Collecting Gold. In: Dastani, M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS, vol. 4908, pp. 251–255. Springer, Heidelberg (2008)
13. Horridge, M., Bechhofer, S., Noppens, O.: Igniting the owl 1.1 touch paper: The owl api. In: Proc. of OWLED 2007, Innsbruck, Austria, CEUR-WS (2007)
14. Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wang, H.: The manchester owl syntax. In: Proc. of OWLED 2006, Athens, GA, USA (2006)
15. Horrocks, I.: FaCT and iFaCT. In: Proc. of the International Workshop on Description Logics (DL 1999), pp. 133–135 (1999)
16. McBride, B.: Jena: a semantic web toolkit. IEEE Internet Computing 6(6), 55 (2002)
17. McGuinness, D.L., van Harmelen, F. (eds.): OWL Web Ontology Language overview. W3C Recommendation (February 2004), http://www.w3.org/TR/owl-features/
18. McGuinness, D.L., van Harmelen, F. (eds.): OWL Web Ontology Language Reference. W3C Recommendation (February 2004)
19. Moreira, Á.F., Vieira, R., Bordini, R.H., Hübner, J.F.: Agent-oriented programming with underlying ontological reasoning. In: Baldoni, M., Endriss, U., Omicini, A., Torroni, P. (eds.) DALT 2005. LNCS, vol. 3904, pp. 155–170. Springer, Heidelberg (2006)
20. Morley, D., Myers, K.L.: The spark agent framework. In: AAMAS 2004, New York, USA, August 19-23, 2004, pp. 714–721. IEEE Computer Society, Los Alamitos (2004)
21. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A BDI reasoning engine. In: Bordini, et al. (eds.) [3], ch. 6, pp. 149–174
22. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
23. 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)
24. Staab, S., Studer, R. (eds.): Handbook on Ontologies. International Handbooks on Information Systems. Springer, Heidelberg (2004)
25. Winikoff, M.: JACK^{TM} intelligent agents: An industrial strength platform. In: Bordini, et al. (eds.) [3], ch. 7, pp. 175–193

Leveraging New Plans in AgentSpeak(PL)

Felipe Meneguzzi and Michael Luck

King's College London
Department of Computer Science
Strand, London WC2R 2LS, UK
{felipe.meneguzzi,michael.luck}@kcl.ac.uk

Abstract. In order to facilitate the development of agent-based software, several agent programming languages and architectures, have been created. Plans in these architectures are often self-contained procedures with an associated *triggering event* and *a context condition*, while any further information about the consequences of executing a plan is absent. However, agents designed using such an approach have limited flexibility at runtime, and rely on the designer's ability to foresee all relevant situations an agent might have to handle. In order to overcome this limitation, we have created AgentSpeak(PL), an interpreter capable of performing state-space planning to generate new high-level plans. As the planning module creates new plans, the plan library is expanded, improving performance over time. However, for new plans to be useful in the long run, it is critical that the *context conditions* associated with new plans are carefully generated. In this paper we describe a plan reuse technique aimed at improving an agent's runtime performance by deriving optimal context conditions for new plans, allowing an agent to reuse generated plans as much as possible.

1 Introduction

Software based on autonomous agents is often advocated as a solution to addressing highly dynamic environments in which human intervention is impractical or impossible. In order to facilitate the development of such agent-based software, several agent programming languages, as well as associated agent architectures, have been created. So far, however, for reasons of efficiency, the set of practical agent architectures developed has mainly focused on providing a *plan execution* framework for a plan library defined at design time [1,2]. Plans in these architectures are often self-contained procedures with an associated *triggering event*, while any additional information about the consequences of executing a plan is absent. For example, PRS [3] and its successors [4,5] provide concrete agents which, while efficient, are noticeably inflexible in handling anything not foreseen at design-time.

Traditional BDI agents [2] are designed using a procedural approach, which requires a designer to create detailed procedural plans for every relevant situation in which an agent may find itself prior to deployment. Situations in which plans must be executed are encoded in a plan header in two parts: an invocation condition, identifying the moment when a plan may be necessary; and a context condition describing the prerequisites for the plan to be applicable, as shown in Figure 1. Both the triggering event and the context condition are defined statically at design time, and ensure that a plan

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 111–127, 2009.

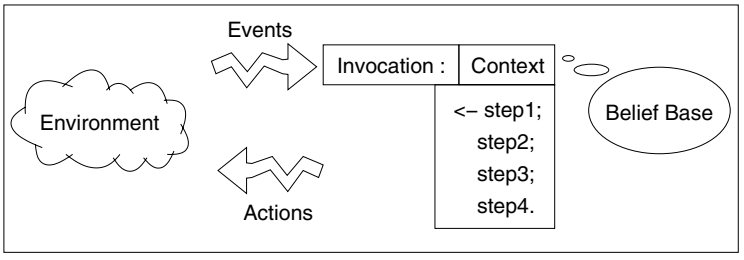

Fig. 1. AgentSpeak(L) plan and dynamics

can execute successfully when it is required. However, agents designed using such an approach have limited flexibility at runtime, and rely on the designer's ability to foresee all relevant situations an agent might have to handle.

In order to overcome this limitation, we have created AgentSpeak(PL) [6], an interpreter capable of generating new high-level plans when no suitable plans exist in the plan library. These high-level plans are created by sequencing existing lower-level plans from the plan library, from which key information about their declarative preconditions and consequences is extracted. AgentSpeak(PL) uses state-space planning to create new plans, and since state-space planners are inherently declarative, AgentSpeak(PL) is able to reason about declarative goals and create plans which, when executed, ensure that a certain world-state is true. The approach taken in AgentSpeak(PL) consists of evaluating the consequences of procedural plans in terms of belief additions and deletions and converting these plans into a STRIPS-like representation, which can then be supplied to a classical planner [7] along with the current belief base and the desired goal state. In this setting, STRIPS operators are essentially an analogue for lower-level AgentSpeak(L) plans and, therefore, plans generated by the planning module represent high-level AgentSpeak(L) plans, which consist entirely of lower-level plan invocations. As the planning module creates new plans, the plan library is expanded, improving performance over time.

However, for new plans to be useful for an agent in the long run, it is critical that the *context condition* associated with new plans is simple enough so that plans can be executed whenever they accomplish their goals, and restrictive enough, so that plans do not execute in situations in which they would fail. As an example, suppose that a certain agent has a car and a motorcycle available to move it from home to work, and an action to drive each one of these vehicles having a precondition that the vehicle being used must have enough fuel for the journey. Furthermore, suppose that at one point in time this agent generated a plan to drive its car to work, while believing that both the motorcycle and the car had enough fuel. The precondition of the high-level plan involving the car surely must contain the belief regarding the car's fuel level, but not the motorcycle's, as it is irrelevant to that plan.

While previous work [6] focused on the integration of the interpreter with the planner through a translation process, in this paper we focus on specific aspects of adding new plans to the plan library. The key contribution is an improvement in an agent's runtime performance by deriving optimal context conditions for new plans, allowing an agent

to reuse generated plans as much as possible. We evaluate the resulting system against a naive strategy of plan reuse, as well as a similar agent designed using AgentSpeak(L), in order to demonstrate the efficiency of our approach. Although we use AgentSpeak(L) as a demonstration platform, our approach can also be applied to other planning-capable agent architectures.

This paper is organised as follows: Section 2 reviews previous work on AgentSpeak(PL), providing the necessary background for this paper; Section 3 describes our plan reuse strategy, including an algorithm for context generation; Section 4 reports on the experiments performed using an implementation of our strategy and its results for a production cell scenario; Section 5 provides a brief overview of recent related work in comparison to ours; finally, Section 6 draws conclusions from our results and proposes future research based on them.

2 AgentSpeak(PL)

AgentSpeak(PL) [6] is an extended AgentSpeak(L) interpreter that uses a planning component to reason about declarative goals. In this section we briefly describe both the original AgentSpeak(L) interpreter and language, and the extensions provided in AgentSpeak(PL).

2.1 AgentSpeak(L)

AgentSpeak(L) [2] is an agent language, as well as an abstract interpreter for the language, that follows the *beliefs, desires and intentions* (BDI) model of practical reasoning [8]. In simple terms, a BDI agent tries to realise the *desires* it *believes* are possible by committing to carrying out certain courses of action through *intentions*. The language of AgentSpeak(L) allows the definition of *reactive procedural plans*, so that plans are defined in terms of an event to which an agent should react by executing a sequence of steps (*i.e.* a procedure). Plan execution is further constrained by the *context* in which these plans are relevant. Here, a plan is executed under the assumption that some implicit goal is being accomplished by that plan at that particular moment.

The control cycle of an AgentSpeak(L) interpreter is driven by events relating to either new beliefs (including perceptions) or new goals. These events are used as invocation conditions for the adoption of plans, so that adding an achievement goal means that an agent desires to fulfil the goal, and plans whose invocation condition includes that goal (*i.e.* are *relevant* to the goal) should lead to that goal being achieved. Moreover, a plan includes a logical *context* condition that specifies when the plan is *applicable* (*i.e.* possible to be executed) in any given situation. Whenever a goal addition event is generated (as a result of the currently selected plan having subgoals), the interpreter searches the set of relevant plans for applicable plans; if one (or more) such plan is found, it is pushed onto an intention structure for execution. Elements in the intention structure are popped and handled by the interpreter. If the element is an action it is executed, while if the element is a goal, a new plan for that goal is added to the intention structure and processed. During this process, failures may take place either in the execution of actions, or during the processing of subplans. When such a failure takes place, the plan

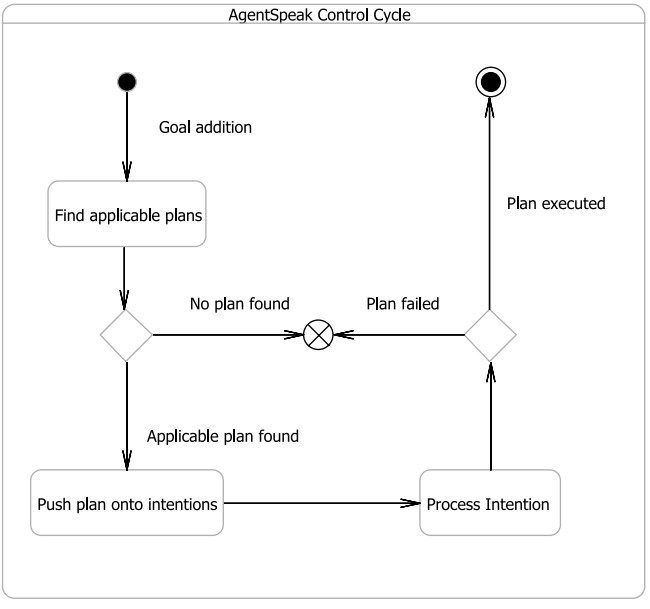

Fig. 2. AgentSpeak(L) reasoning cycle

that is currently being processed also fails. Thus, if a plan selected for the achievement of a given goal fails, the default behaviour of an AgentSpeak(L) agent is to conclude that the goal that caused the plan to be adopted is not achievable. This control cycle is illustrated in the diagram of Figure 2,[1] and strongly couples plan execution to goal achievement.

In order to better understand the relationship between the control cycle and the plan library, it is necessary to introduce the notation of AgentSpeak(L) plans. The events on an agent's data structures that can trigger the adoption of plans consist of additions and deletions of goals and beliefs, and are represented by the plus ($+$) and minus ($-$) sign respectively. Goals are distinguished into *test goals* and *achievement goals*, denoted by a preceding question mark ($?$), or an exclamation mark ($!$), respectively. For example, the addition of a goal to achieve g is represented by $+!g$, whereas the addition of a goal to test the truth value of a belief b is represented by $+?b$. Belief additions and deletions arising as the agent perceives the environment are outside its control, while goal additions and deletions and some belief modifications only arise as part of the execution of an agent's plans. Plans in AgentSpeak(L) are represented by a header comprising an invocation condition and a context, as well as a body describing the steps the agent takes when a plan is selected for execution, as illustrated in Figure 1. Thus, if e is a triggering event, $b_1, \ldots, b_m$ are belief literals, and $h_1, \ldots, h_n$ are goals or actions, then $e : b_1 \& \ldots \& b_m \leftarrow h_1; \ldots; h_n.$ is a plan. As an example, consider a plan associated with the invocation condition `!move(O,A,B)` corresponding to an achievement goal to move an object O from A to B, where:

[1] For a full description of AgentSpeak(L), refer to d'Inverno *et al.*[9].

- e is `!move(O,A,B);`
- `at(O,A)` and **not** `at(O,B)` are belief literals; and
- `-at(O,A)` and `+at(O,B)` are two steps in the plan body, consisting of information about belief additions and deletions.

The plan is then as follows:

```
+!move(O,A,B) : at(O,A) & not at(O,B)
    <- -at(O,A);
       +at(O,B).
```

When this plan is executed, it should result in the agent believing O is no longer in position A, and then believing it is in position B. For an agent to rationally want to move O from A to B, it must believe O is at position A and not already at position B.

2.2 Planning in AgentSpeak(PL)

In order to overcome the limitations of traditional AgentSpeak(L) programming in terms of dynamic plan generation and declarative goal representation, previous work has introduced AgentSpeak(PL) [6], which is an extended AgentSpeak(L) interpreter coupled with a planning module able to perform STRIPS-like planning. The agent interpreter communicates with the planning module through a translation process that relies on the similarities between AgentSpeak(L) plans and STRIPS operators. This is possible because both formalisms describe world modification functions that can be applied if certain preconditions hold, resulting in changes to the world-state.

The Planning Action. In addition to the traditional way of encoding goals for an AgentSpeak(L) agent implicitly as invocation conditions consisting of achievement goals ($!goal$), AgentSpeak(PL) allows desires including multiple beliefs $(b_1, \ldots, b_n)$ describing a desired world-state in the form $goal\_conj([b_1, \ldots, b_n])$. An agent desire description thus consists of a conjunction of beliefs the agent wishes to be true simultaneously at a given point in time. The execution of the planning component is triggered by an event $+goal\_conj([b_1, \ldots, b_n])$ as shown in Listing 1.

In this approach, planning in AgentSpeak is introduced through a special *planning action*, denoted $plan(G)$, where G is a conjunction of desired goals. This action is bound to an implementation of a planning module, and allows all of the process regarding the conversion between formalisms to be encapsulated in the action implementation, making it completely transparent to the remainder of the interpreter. Note that there are two different steps in invoking the planning action: the declarative goal, represented by the $+goal\_conj(Goals)$ event; and the planner invocation action $plan$, which may occur as a consequence of adopting a declarative goal.

Whenever an agent needs to achieve a goal that involves planning, it uses this special planning action that converts the low-level procedural plans of AgentSpeak(L) into STRIPS operators and invokes the planning module. If the planner succeeds in finding a plan, it is converted back into a high-level AgentSpeak(L) plan and is added to the intention structure for execution, as illustrated in Figure 3. This conversion process is detailed in Section 2.2. If the newly created plan fails, the planner may again be invoked

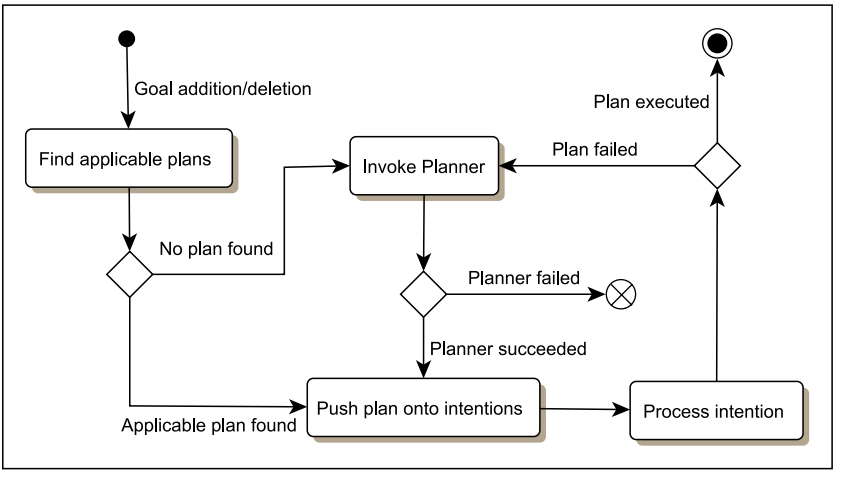

Fig. 3. AgentSpeak(PL) reasoning cycle

$$+goal\_conj(Goals) : true \leftarrow plan(Goals).$$

Listing. 1. Planner invocation plan

to try to find another plan to achieve the desired state of affairs, taking into consideration any changes in the beliefs.

Note that the planning action is included in a standard AgentSpeak(L) plan with the same invocation condition as the plans generated by it. Moreover, new plans are always added to the plan library *before* the plan that executes the planning action. With this arrangement, previously-created plans are consulted first when the interpreter searches for relevant plans, hence having higher priority for execution. If no such plan is found to be applicable, the plan containing the special planning action is invoked as the last remaining option.[2]

Translating AgentSpeak into STRIPS. Once the need for planning is detected, the plan in Listing 1 is invoked so that the agent can tap into a planning component. The process of linking an agent to a propositional planning algorithm includes converting an AgentSpeak(L) plan library into propositional planning operators, declarative goals into goal-state specifications, and the agent beliefs into the initial-state specification for a planning problem. After the planner yields a solution, the ensuing STRIPS plan is translated into an AgentSpeak(L) plan in which the operators resulting from the planning become subgoals. That is, the execution of each operator listed in the STRIPS

[2] An important limitation imposed by our current implementation is that, since goal conjunctions are represented as lists, different goal orderings correspond to different declarative goals and, therefore, a goal to achieve $[a, b]$ is not seen as the same goal to achieve $[b, a]$.

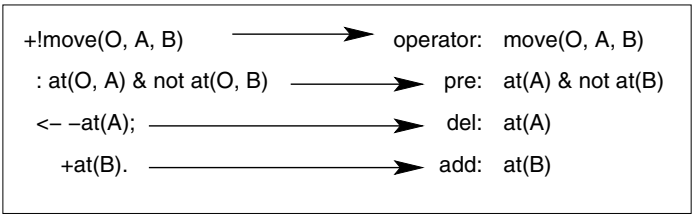

Fig. 4. AgentSpeak plan versus STRIPS operator

plan is analogous to the insertion of the AgentSpeak(L) plan that corresponded to that operator when the STRIPS problem was created.

In classical STRIPS notation, operators have four components: an identifier, a set of preconditions, a set of predicates to be added (add), and a set of predicates to be deleted (del). For example, the same `move` operator can be represented in STRIPS following the correspondence illustrated in Figure 4, in which AgentSpeak(PL) converts the invocation condition into a STRIPS operator header, a context condition into an operator precondition, and the plan body is used to derive add and delete lists.

A relationship between these two definitions is not hard to establish, and AgentSpeak(PL) uses the following algorithm for converting AgentSpeak(L) (low-level) plans into STRIPS operators. Let e be a triggering event, $b_1 \& \ldots \& b_m$ a conjunction of belief literals representing a plan's context, $a_1, \ldots, a_n$ be belief addition actions, and $d_1, \ldots, d_o$ be belief deletion actions within a plan's body. All of these elements can be represented in a single AgentSpeak(L) plan. Moreover let $opname$ be the operator name and parameters, pre be the preconditions of the operator, add the predicate addition list, and del the predicate deletion list. In summary, mapping an AgentSpeak(L) plan into STRIPS operators is thus accomplished as follows:

1. $opname = e$
2. $pre = b_1 \& \ldots \& b_m$
3. $add = a_1, \ldots, a_n$
4. $del = d_1, \ldots, d_o$

This deals with STRIPS operators, but we also need two other elements to create a valid STRIPS problems, namely an initial state and a goal state. Previously, we have introduced the representation of a conjunction of desired goals as the predicate $goal\_conj([b_1, \ldots, b_n])$. The list $[b_1, \ldots, b_n]$ of desires is directly translated into the goal state of a STRIPS problem. Moreover, the initial state specification for a STRIPS problem is generated directly from the agent's belief database.

Executing Generated Plans. The STRIPS problem generated from the set of operators, initial state and goal state is then processed by a propositional planner. If the planner fails to generate a propositional plan for that conjunction of literals, the plan in Listing 1 fails immediately and the goal is deemed unachievable, otherwise the resulting propositional plan is converted into an AgentSpeak(L) plan and added to the intention structure. In order to convert this plan back to an AgentSpeak(L) representation, we consider that a propositional plan from a STRIPS planner is in the form of a sequence

$$+goal\_conj(Goals) : true \leftarrow !op_1; \ldots; !op_n.$$

Listing. 2. AgentSpeak plan generated from a STRIPS plan

$op_1, \ldots, op_n$ of operator names and instantiated parameters. AgentSpeak(PL) creates a new plan as in Listing 2, where $goal\_conj(Goals)$ is the event that caused the planner to be invoked.

Immediately after adding the new plan to the plan library, the event $goal\_conj(Goals)$ is reposted to the agent's intention structure, causing the generated plan to be executed. The abstract language of AgentSpeak(L) does not include constructs for plan library modification, but this type of functionality is generally accomplished through internal actions by many interpreters, including *Jason* [10]. As a consequence, there is no commonly agreed semantics for the addition of new plans into an agent's plan library. The method we use is that of the *Jason* interpreter, which consists of inserting the new plan either at the beginning of the plan library or at its end. Plans generated in this fashion are admittedly simple, and in order for an agent to take full advantage of the planning module, we need to consider how plans should be added to the plan library for future reference.

2.3 Limited Plan Reusability

The addition of the new plan to the intention structure raises the problem of how newly formed plans can be integrated into the agent's existing plan library, or indeed if they should be integrated into the plan library at all. Modifying the plan library at runtime through the addition of new plans effectively changes agent behaviour in at least two ways: first, new plans may cause undesired interactions with the plans that are already part of the plan library, possibly jeopardising the agent's viability in the long term; and second, adding a large number of plans with the same invocation condition may impair the agent's ability to respond in adequate time.

In order for a plan to be usefully added to the plan library, therefore, the *context* in which this plan is relevant must be carefully described. If the context is too restrictive, for example by using the entire belief base at the time of planning, the inclusion of a number of irrelevant beliefs will severely limit the future applicability of the new plan. On the other hand, if the context is minimised to only the preconditions of the first operator, the plan may fail later on due to the requirements of subsequent operators. In consequence, an algorithm that generates the minimum context condition necessary for a newly generated plan to be reused usefully is required.

3 Leveraging New Plans

In order to address the need for a minimum context condition for newly created plans, we have developed an algorithm to extract the minimum necessary context condition

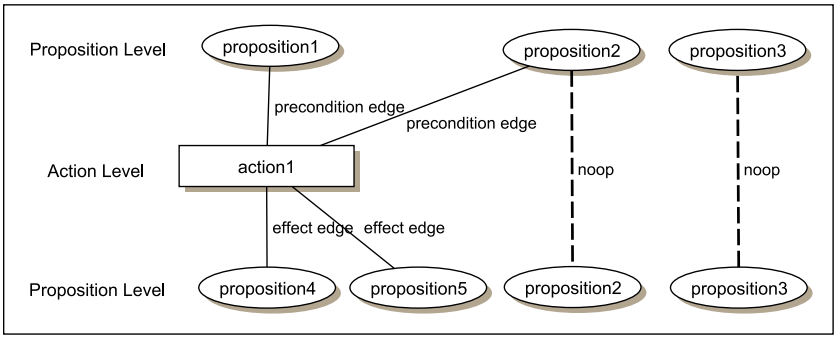

Fig. 5. A planning graph example

from a planner-generated plan that ensures that if the context is true when the plan is adopted, it will succeed if no external interference takes place.

3.1 Data Structure

We base our context-generation algorithm on a modified version of the *planning graph* data structure from Graphplan [11], which is a graph-based planning algorithm. The properties of this method of plan representation are key to our algorithm and, therefore, before describing the algorithm, we introduce the planning graph. Since a plan is composed of temporally ordered actions, and these actions alter propositions in the intermediate world states, graph levels are divided into alternating proposition and action levels, making it a directed and levelled graph. A graphical representation of one such graph is shown in Figure 5, in which oval shapes denote propositions and boxes denote actions. Proposition levels are composed of proposition nodes labelled with propositions, and are connected to the actions in the subsequent action level through precondition arcs. Here, action nodes are labelled with operators and are connected to the nodes in the subsequent proposition nodes by effect arcs, and both added and deleted propositions are possible effects of an operator.

Every proposition level denotes literals that are possibly true at a given moment, so that the first proposition level represents the literals that are possibly true at time t_1 (the initial time), the next proposition level represents the literals that are possibly true at time t_2 and so on. Similarly, action levels denote operators that can be executed at a given moment in time in such a way that the first action level represents the operators that may be executed at time t_1, the second action level represents the operators that may be executed at time t_2 and so on.

The process of building a graph in Graphplan consists of initialising it with a proposition level containing the initial state of the planning problem, and adding all actions that have their entire set of preconditions present in that proposition level. New proposition levels are then created, including all the effects of the preceding action level. In order to guarantee a static frame for all actions in the graph (that is, to ensure propositions not affected by plan operators remain unchanged between points in time), *no operation* (or *noop*) edges are inserted between propositions to represent the possibility that these propositions are not changed between two proposition levels (*i.e.* points in

time). The planning graph used in Graphplan has a number of other characteristics that we do not explain in this paper because they are not relevant to our algorithm, but are discussed in [11].

3.2 Generating Context Information

Intuitively, the preconditions of any given plan step must have either been made true during the execution of previous plan steps or must have been true from the start of the plan. Therefore, the minimum context condition for any generated plan must specify the preconditions of the first operator, plus the preconditions of any subsequent operators that are not included in the effects of previous operators. We consider this process in more detail in Algorithm 1, which describes the generation of such a context condition.

Algorithm 1. Propagation of preconditions

Require: Plan $\Delta = \{a_1, \ldots, a_n\}$, with n steps
Require: Action descriptions $\mathbf{O} = \{\langle a_1, Pre_1, Post_1\rangle, \langle a_n, Pre_n, Post_n\rangle\}$
1. create a proposition level P_0 with no propositions;
2. **for all** $a_i \in \Delta$ **do**
3. create an action level A_i containing a node a_i;
4. add the preconditions of a_i to proposition level P_{i-1};
5. connect all $p \in P_i$ to a_{i-1} with precondition edges;
6. create a proposition level P_i containing the effects of a_i;
7. connect all $p \in P_i$ to a_i with effect edges;
8. **end for**
9. **for** $i = n$ to 1 **do**
10. **for all** $p \in P_{i-1}$ **do**
11. **if** p is not connected to any node in level A_i **then**
12. create an action $noop(p)$ in level A_i;
13. connect $noop(p)$ to p through an effect edge;
14. **if** $p \notin P_i$ **then**
15. create a node p in P_i;
16. **end if**
17. connect p to $noop(p)$ with a precondition edge;
18. **end if**
19. **end for**
20. **end for**
21. **return** P_0

The algorithm initially builds a planning graph populated with the actions of the plan we wish to create a context for, as well as the preconditions and effects of these actions, with edges connecting actions to their preconditions in the previous level, and their effects in the subsequent level. Once the initial graph is generated, proposition levels are iterated backwards and, for each proposition that is connected with a precondition edge to a subsequent action level and not connected with an effect edge to the previous action level, a new *noop* action is created, allowing a proposition to be propagated to the previous proposition level. As the graph is traversed, propositions that are required at one action level are created at the preceding proposition levels until they are either

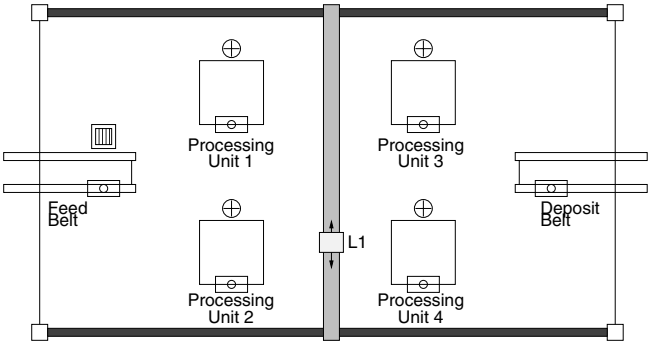

Fig. 6. Diagram of the production cell

connected to an original action of the plan, or they are propagated through *noop* actions, ensuring that the first proposition level contains all of the preconditions that did not result from the actions in the plan. In principle, this algorithm can be greatly simplified, however using the planning graph approach allows us to extend this algorithm to richer operator descriptions, for example, for operators with conditional effects.

3.3 Complexity

In terms of computational effort, this algorithm has similar complexity to the graph expansion phase of Graphplan, which has polynomial complexity [7] in the size of the planning problem for both the size of the graph and the time required to build it. If a plan has m distinct steps, and n distinct propositions, the graph our algorithm creates will have at most $((2 * n) + 1) * m$ nodes, one node for each action and all possible *noop*s at each graph level, plus all possible propositions at each proposition level, indicating that the size and time complexity of our algorithm is in the low polynomial scale. Regarding the correctness of the algorithm and its termination guarantee, since the graph building part of the algorithm is a subset of Graphplan, for which a proof of completeness and termination exists, and the rest of the algorithm is an iteration in a directed acyclic graph, it is trivial to show that the algorithm does terminate for any input.

3.4 A Production Cell Example

To illustrate how our algorithm derives a context condition, we introduce a production cell scenario, shown in Figure 6, and consisting of a production cell composed of four processing units (`u1`, `u2`, `u3` and `u4`) and two conveyor belts controlled by our agent. Parts enter the production cell through a *feed belt*, and are moved by the agent to different *processing units*, depending on the type of part being processed. Once a part has been processed at the appropriate processing units, it is moved to the *deposit belt* to be shipped. Even though there is no particular order specified for the processing of parts, the order in which they are specified is generally followed. We consider three different types of part for processing, in the following processing units:

Table 1. Operations in the production cell scenario

Operator	Preconditions	Effects
`move(P,A,B)`	`empty(B)`	`~empty(B)`
	`over(P,A)`	`~over(P,A)`
		`over(P,B)`
		`empty(A)`
`process(P,A)`	`over(P,A)`	`processed(P)`

1. Type one must be processed by processing units 1, 2 and 3;
2. Type two must be processed by processing units 2 and 4; and
3. Type three must be processed by processing units 1 and 3.

In addition, we assume two operations are available in this scenario, summarised in Table 1.[3] The first operator, `move(P,A,B)`, moves a part from one device to another, requiring the part to be over the initial device and the target device to be empty, and causing the initial device to become empty, the part to be over the target device and the deletion of the preconditions (note that while we represent explicitly negated propositions in the graph, we do not require the world to be described with explicitly negated conditions). The second operator, `process(P,A)`, causes a processing unit to process a part located over it, requiring `over(P,A)` and causing `processed(P)`.

Now let us consider in more detail the process of generating the context for this example. Nodes in an action level are connected to nodes in a proposition level either through precondition edges, denoting that a proposition is a precondition of a given action, or through effect edges, denoting that a proposition is an effect of a given action. In the example of Figure 7, the operator `process(p1,u2)` in Level 4 is connected by a precondition edge to the proposition `over(p1,u2)` in Level 3, and by precondition edges to the proposition `processed(p1,u2)` in Level 5. Besides the actions included in the planning problem, the planning graph includes *noop* (or maintenance) actions, which connect identical propositions between adjacent proposition levels representing that their truth values remain unchanged between plan steps, an example of which can be seen connecting the proposition `empty(u3)` from Levels 1 to 5.

Figure 7 shows the graph generated in the process of deriving the context condition for a plan composed of three actions: `move(p1,u1,u2)`, `process(p1,u2)` and `move(p1,u2,u3)`. The initial graph created by our algorithm contains no maintenance actions and no instances of `empty(u3)` in Levels 1 and 3. Then, while iterating the graph backwards, the algorithm detects that none of the preconditions of `move(p1,u2,u3)` were caused by the immediately preceding action, and adds a *noop* connecting the instances of `over(p1,u3)` in Levels 3 and 5. Furthermore, it creates instances of `empty(u3)` in Levels 1 and 3, connecting them with maintenance operators in Levels 2 and 4, thus propagating `empty(u3)` to the initial plan level. Since no action in the plan resulted in `empty(u3)` being true, this must have been true before the plan

[3] We use a Prolog-like notation, with variable names starting in uppercase and constants in lowercase.

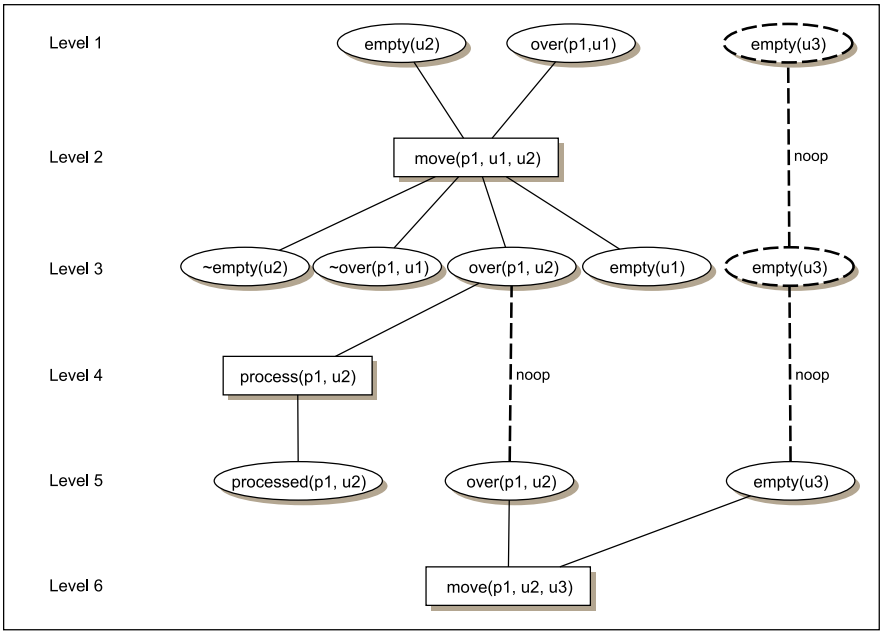

Fig. 7. A planning graph used in context extraction

was adopted to make the last action possible. These additions to the planning graph can be seen in Figure 7 as the dashed oval shapes and lines.

4 Experiments and Results

Traditional AgentSpeak(L) agents require a plan library containing plans for every conceivable situation an agent might find itself in, since no plans can be created at runtime to deal with unexpected events. Therefore, the ability to generate new plans at runtime both increases an agent's flexibility and eases agent development. On the other hand, state space planners are complex, and a decrease in runtime performance is expected over standard AgentSpeak(L). With the addition of an effective plan reuse strategy, however, the time spent in the planning process can be mitigated over time, since new plans will have the same runtime efficiency as traditional AgentSpeak(L).

Our prototypes were implemented using modified versions of Jason [10], a Java-based AgentSpeak(L) interpreter with a few additional constructs such as plan annotations and plan failure handling. Experiments with traditional AgentSpeak(L) agents were conducted in an unmodified Jason interpreter, whereas planning agents were created using the open-source implementation of AgentSpeak(PL) [6], unmodified for experiments without plan reuse; and extended with our algorithm for context generation.

The experiment consists of simulating the arrival of parts of three types in three production cells, one controlled by a traditional AgentSpeak(L) agent (which we label AS), another controlled by a *naive* version of AgentSpeak(PL) (which we label NaiveAS)

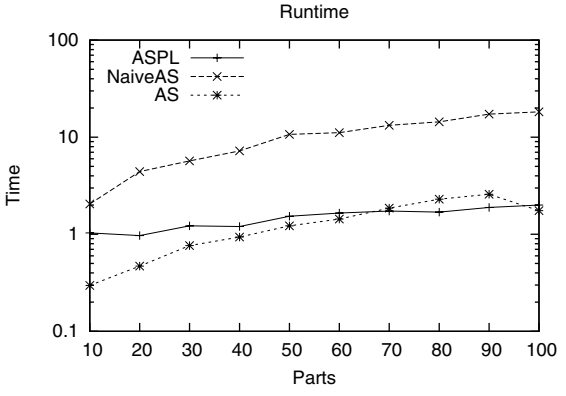

Fig. 8. Running times for the Production Cell scenario

Table 2. Comparison of initial plan library sizes

	AS	**ASPL**	**NaiveAS**
# plans	12	7	7

that does not reuse plans and one controlled by the *complete* AgentSpeak(PL) (which we label ASPL) capable of reusing plans. Here, whenever a new part arrives for processing at the cell controlled by NaiveAS, the full planning process is invoked to generate a new plan, regardless of previous instances of the same problem having been considered in the past. The time spent planning and achieving the final processing of every part is measured for each agent for an increasing number of parts, ranging from 10 to 100 in 10 part increments.

The results of this experiment can be seen in the graph of Figure 8, which shows that, though NaiveAS takes significantly more time to perform its reasoning cycle, this overhead is constant. Now, when the plan reuse strategy is used by ASPL, runtime performance improves considerably, approaching that of AS. With three different part types, the number of possible world configurations at the time of planning is limited, and most of the planning effort occurs at the beginning of the agent execution. As more parts of the same type are introduced in the production cell, the plans generated previously are invoked rather than the planning module, *amortising* the cost of the initial planning. Evidence of this effect is provided by the ASPL curve approaching that of AS as the number of total parts increases. Moreover, since the plans generated through planning are a linear sequence of actions, which do not rely on the tests distributed throughout a branching structure of plans in the plan library, they are inherently faster to be executed than the equivalent AS representation, surpassing it in the long term.

It is important to note that, although ASPL can create plans for situations in which AS would fail, we have avoided using these problems in our benchmark, focusing only on runtime, by considering an AS agent with plans for all situations possible during testing. By relying on a planning approach, we also diminish the size of the agent specification, since we no longer need to create a procedural plan to cope with every

world configuration relevant to the accomplishment of an individual plan. The numbers of plans necessary in the (initial) plan libraries are shown in Table 2.

5 Related Work

Work on using planning modules to augment existing architectures has been conducted by several researchers, such as in Propice-Plan [12] and JADEX [13,14]. These efforts provide some insight into many practical issues that may arise from the integration of BDI with AI planners, such as how to modify a planning algorithm to cope with changes in the initial state during planning [12], and how to cope with conflicts in concurrently executing plans [13].

Propice-Plan [12] is a PRS-based system that includes planning capabilities through a modified version of the IPP planner [15]. It includes refinements to allow an agent to anticipate alternative execution paths for its plans, as well as the ability to update the state of the planning process in order to cope with a highly dynamic world. Propice-Plan is similar in principle to the architecture described here, but it differs in two key aspects: its reliance on a modified PRS description formalism for agents, and its reliance on a tailor-made planner implementation, limiting the choice of planners to be used in tandem with the agent interpreter.

The work of Walczak *et al.*[13] is a recent approach to merging BDI reasoning with planning capabilities, and is based on a continuous planning and execution framework implemented in the JADEX agent framework [16]. The system uses a modified HTN state-based planner with domain-specific information to select the actions to achieve goals or refine goals in an agent's agenda. The emphasis in this system is on performance and reaction time rather than generality, since JADEX uses a Java-like representation for the agent's data structures, such as goals and actions. Admittedly, an HTN planner could be used to generate new plans following a task-oriented approach (and hence not for declarative goals), but this is not what is accomplished in JADEX.

Considering the many similarities between BDI programming languages and HTN planning, Sardina *et al.*[14] formally define how HTN planners can be integrated into a BDI architecture. Sardina shows that the HTN process of systematically substituting higher-level goal tasks until concrete actions are derived is analogous to the way in which a PRS-based interpreter pushes new plans onto an intention structure, replacing an achievement goal with an instantiated plan. Taking advantage of this almost direct correspondence, an HTN planner is used to add *lookahead* capabilities to an agent, allowing it to optimise plan selection and maximise an agent's chance of successfully achieving goals. By verifying beforehand the selection of plans for achieving subgoals, the agent minimises the chance of failure as a result of poor plan selection.

The idea of analysing one formalism to derive planning-like pre and post conditions has been attempted previously in the context of web service composition through planning. Initial efforts by McIlraith and Fadel [17] at a theoretical level, involved converting web services described by hand using Golog into PDDL and ADL. However, this lacked generality due to its heavy reliance on human intervention in the process,

preventing it from being used in a completely automated fashion, as is needed by our work. Later, this idea was refined by Pistore *et al.*[18], converting web services defined in BPEL4WS into PDDL, allowing for automation. However, BPEL is much more complex than AgentSpeak(L), and understandably the conversion algorithm has polynomial complexity, though on the *exponential* scale. In this respect, our approach compares favourably by having non-exponential polynomial complexity.

6 Conclusions

In this paper we have described a plan reuse strategy for AgentSpeak(PL), a planning-capable extension of AgentSpeak(L), able to reason about declarative goals. This strategy is based on the generation of the simplest context information necessary for newly created plans to be successful when responding to the same event that triggered the agent to plan. Since planning is a computationally intensive task, being able to effectively reuse previously generated plans offsets time spent on the planning process, as new plans are as efficient as AgentSpeak(L) procedural plans. This bridges the performance gap between traditional AgentSpeak(L) agents and declarative goal-based agents developed in AgentSpeak(PL), which is the primary contribution of this paper.

The strategy used in the generation of context information is based on the generation of a greatly simplified planning graph analogous to the graph used in Graphplan [11]. Building this graph has low polynomial complexity in both time and space, and therefore the performance overhead imposed on the planning process is negligible. Moreover, we believe that the process used to generate initial context information can be expanded to allow reasoning about interference from concurrent plans, and we intend to explore this possibility as future work.

Acknowledgments. The first author is supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) of the Brazilian Ministry of Education.

References

1. Meneguzzi, F., Zorzo, A.F., da Costa Móra, M., Luck, M.: Incorporating planning into BDI agents. Scalable Computing: Practice and Experience 8, 15–28 (2007)
2. Rao, A.S.: AgentSpeak(L): BDI agents speak out in a logical computable language. In: de Velde, W.V., Perram, J.W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
3. Rao, A.S., Georgeff, M.P.: BDI-agents: from theory to practice. In: Proceedings of the First International Conference on Multiagent Systems, San Francisco, pp. 312–319 (1995)
4. d'Inverno, M., Luck, M., Georgeff, M., Kinny, D., Wooldridge, M.: The dMARS Architecture: A Specification of the Distributed Multi-Agent Reasoning System. Autonomous Agents and Multi-Agent Systems 9(1 - 2), 5–53 (2004)
5. Bordini, R.H., Hübner, J.F., Vieira, R.: Jason and the golden fleece of agent-oriented programming. In: Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E. (eds.) Multi-Agent Programming: Languages, Platforms and Applications, pp. 3–37. Springer, Heidelberg (2005)

6. Meneguzzi, F., Luck, M.: Composing high-level plans for declarative agent programming. In: Baldoni, M., Son, T.C., van Riemsdijk, M.B., Winikoff, M. (eds.) DALT 2007. LNCS, vol. 4897, pp. 115–130. Springer, Heidelberg (2008)
7. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Elsevier, Amsterdam (2004)
8. Bratman, M.E.: Intention, Plans and Practical Reason. Harvard University Press, Cambridge (1987)
9. d'Inverno, M., Luck, M.: Engineering AgentSpeak(L): A formal computational model. Journal of Logic and Computation 8(3), 233–260 (1998)
10. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. Wiley, Chichester (2007)
11. Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artificial Intelligence 90(1-2), 281–300 (1997)
12. Ingrand, F., Despouys, O.: Extending procedural reasoning toward robot actions planning. In: Proceedings of the 2001 IEEE International Conference on Robotics and Automation, Seoul, Korea, pp. 9–10 (2001)
13. Walczak, A., Braubach, L., Pokahr, A., Lamersdorf, W.: Augmenting BDI Agents with Deliberative Planning Techniques. In: Proceedings of the Fifth International Workshop on Programming Multiagent Systems (2006)
14. Sardina, S., de Silva, L., Padgham, L.: Hierarchical Planning in BDI Agent Programming Languages: A Formal Approach. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1001–1008 (2006)
15. Köhler, J.: Solving complex planning tasks through extraction of subproblems. In: Simmons, R., Veloso, M., Smith, S. (eds.) Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, Pittsburg, pp. 62–69. AAAI Press, Menlo Park (1998)
16. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A BDI reasoning engine. In: Bordini, R.H., Dastani, M., Dix, J., Fallah-Seghrouchni, A.E. (eds.) Multi-Agent Programming: Languages, Platforms and Applications, pp. 149–174. Springer, Heidelberg (2005)
17. McIlraith, S.A., Fadel, R.: Planning with complex actions. In: Benferhat, S., Giunchiglia, E. (eds.) Proceedings of the 9th International Workshop on Non-Monotonic Reasoning, pp. 356–364 (2002)
18. Pistore, M., Marconi, A., Bertoli, P., Traverso, P.: Automated composition of web services by planning at the knowledge level. In: Kaelbling, L.P., Saffiotti, A. (eds.) Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, pp. 1252–1259 (2005)

Increasing Bid Expressiveness for Effective and Balanced E-Barter Trading

Azzurra Ragone[1,3], Tommaso Di Noia[1], Eugenio Di Sciascio[1], and Francesco M. Donini[2]

[1] SisInfLab, Politecnico di Bari, Bari, Italy
{t.dinoia,disciascio}@poliba.it
[2] Università della Tuscia, Viterbo, Italy
donini@unitus.it
[3] Artificial Intelligence Laboratory–University of Michigan, Ann Arbor, USA
aragone@umich.edu

Abstract. We present a novel knowledge-based approach for automated electronic barter trade systems. An e-barter is basically a closed e-marketplace, where agents may exchange (buy/sell) goods –or equivalent trade dollars– only with other participants to the e-barter. Obviously, in such systems one of the major issues is keeping exchanges as balanced as possible. If the description of goods or services to be exchanged is simple and limited to a well defined set, *e.g.*, oil, wheat, transport, etc., then an exchange based only on price and quantity is enough. But, what if goods or services to be exchanged are described in a complex way? Is it a suitable exchange the one involving *mobile phones supporting video streaming with a QWERTY keyboard* if the agent is looking for *smart phones*? Those two descriptions, although very different form a syntactic point of view, are very similar with respect to their meaning (semantics). How could an agent manage and exploit the knowledge on a given domain to deal with such a semantic information and optimize exchanges?

We focus on how to find most promising matches, in a many-to-many matchmaking process, between bids (supplies/demands), taking into account not only the price and quantities as in classical barter trade systems, but also a semantic similarity among bid descriptions while keeping exchanges balanced.

To this aim we use a logical language to express agent preferences, thereby enhancing bid expressiveness. We also define a logic-based utility function that allows to evaluate the semantic similarity between bids. Finally we illustrate the optimization problem we solve in order to clear the market.

1 Introduction

An electronic marketplace can be basically described as a system that facilitates business activities by providing users with one, or more, added value services, usually including: discovery/matchmaking (finding partners to engage in a commercial interaction), negotiation and deal (establishing trust, negotiating and agreeing on terms of business transaction), exchange (payment and actual execution of the business transaction) [29]. The main interest of the service provider is obviously to maximize successful transactions (*i.e.*, what is usually termed as clearing the market). The service

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 128–142, 2009.

provider revenues depend in fact either on charges placed on each successful transaction or on fixed fees paid for membership required to benefit of offered services. Successful and renowned existing e-marketplaces are either person-to-person auction sites such as EBay (www.ebay.com) or business-to-business (B2B) marketplaces such as Covisint (www.covisint.com) or Alibaba.com (www.alibaba.com), or procurement services such as CombineNet (www.combinenet.com).

In this paper we focus on a particular –and definitely ancient– form of commercial interaction, namely barter trade exchange. Historically barter trade was a bilateral form of exchange of goods and services without currency. Obviously, we refer here to a *modern multilateral* barter trade [14], where traders do not exchange goods directly, but use a form of private label currency, named *trade dollar*. Therefore if they sell a good they receive credits in trade dollars, that can be used to purchase other goods. An electronic barter trade system is then basically a *closed* B2B e-marketplace, where the trade of goods/services among companies is managed by an intermediary (broker). In automated *e-barter* exchange agents play the role of managers (acting on behalf of companies) or brokers (acting on behalf of the trade exchange system).

Usually, commercial e-barter systems make money by charging a commission on each transaction done, so the revenue of the system is higher the more the e-marketplace is lively. Therefore the role of the broker agent is to stimulate trade exchanges, recommending possible promising exchanges given a set of demands and supplies from the *barter pool* (the set of companies involved in the e-marketplace).

One of the aims of the barter trade broker is maintaining exchanges as far as possible balanced, so that the total income of trade dollars by a company equals to the amount bought by the company itself. In fact maintaining the balance of trade helps the traders to make purchases in the future increasing the trade volume over the long run [14]. Furthermore, given a demand (supply), there are typically several possible supplies (demands) to choose from, so the pivotal question in an e-barter system is: how can the broker choose and consequently suggest an exchange to the other agents? The obvious answer is: by finding the most *promising* matches such that agents can be equally satisfied by the exchange. Obviously, price cannot be the only parameter when goods to be exchanged are not simply undifferentiated ones, and moreover traders do not make their decision based only on price [14]. Usually price is negotiated later between buyer and seller, so price other than not being the only criterion, might not be the most important one. Other parameters to take into account are, *e.g.*, similarity between demand and supply descriptions, quantity and trade balance.

We introduce logical languages, in particular Description Logics [4], to model bids. In this way we enhance bid expressiveness, and are able to catch relations among features, exploiting basic inference services such as satisfiability and subsumption. Yet our aims are manifold and go well beyond simple matchmaking:

- Maximize utilities of each agent finding the best *overall semantic match* among several demands and supplies;
- Maintain the balance of trade;
- Maximize the trade volume.

While achieving these goals separately can be straightforward, it is quite challenging trying to fulfill all of them at the same time; contributions of this paper therefore

include a many-to-many matchmaking process between bid descriptions modeling both mandatory requirements and preferences, taking into account not only price and quantities as in classical barter trade systems; a logic-based utility function that allows to evaluate the semantic similarity between bids; the optimization problem we solve in order to clear the market keeping exchanges balanced. The rest of the paper is structured as follows: in the next section we illustrate features and motivations for e-barter trading outlining the scenario we refer to; then we illustrate the logical language we adopt to model agents bids. In Section 4 we define the logic-based utility function we introduce to catch the semantic similarity between bids. Section 5 outlines the optimization problem we solve to clear the market, taking into account both quantity constraints (balance of trade) and utility function. Related work and conclusion close the paper.

2 The Barter Trade Scenario

The World Trade Organization estimated in 2004 that 15% of international trade was conducted on non-cash basis, and approximately $8.25 billion was traded through reciprocal trade companies [16]. It should be noted that the interest for bartering does not depend on the possibility to avoid/reduce value added taxes (VAT), as practically all countries have long ago introduced specific legislation that make barter income equal to cash-based income; the reciprocal trade among firms is considered appealing as it allows the exchange of unproductive assets and surplus inventory for valuable products or services, opening at the same time new outlets for excess inventory and unused capacity [16]. Noteworthy examples of working e-barter marketplaces, among many others, are `www.tradia.net`, `www.U-Exchange.com`, `www.trashbank.com`, `www.tradefirst.com`, `www.barterbart.com`, and BizXchange[1]. The range of products/services that it is possible to buy/sell is very wide; among many others: administrative services, business consultation, legal and accounting services, automotive services, computer and technology services, telephone and telecommunication systems, commercial furniture.

Let us consider the following scenario[2]: A chain of hotels needs to buy mobile phones for a large number of its employees. The company has two choices: the first one is simply buy the mobile phones in the open market, the second one is pursuing an exchange of accommodations for mobile phones. Their occupancy rate is about 60%. The chain of hotels therefore may trade hotel stays for a complete mobile phone supply and increase its occupancy. This is accomplished without the use of cash, and with mutually benefical results. Hence, the idea is that barter exchange enables a company to use its excess capacity to finance its purchases.

An e-barter system shares several characteristics with generic B2B e-marketplace, where agents enter their demands/supplies (*bids*) to search for potential commercial partners. Differently from a peer-to-peer (P2P) e-marketplace, and similarly to an auction-based market, there is a central entity, the *broker*, which finds, on behalf of the company agents, most promising transactions based on some constraints, as will be

[1] `www.bizx.bz`

[2] This scenario is inspired by an example found on the web site http://www.tradia.net/

discussed later on. In this paper we model the trade balance problem extending the approach by Haddawy et al. [14], who modeled the setting as a minimum circulation problem [2] on a network, considering expressive demand/supply descriptions referred to a common ontology, similarity among goods, preferences and utilities of buyers/sellers. As in [14] we consider the trade occurring in business cycles: agent's bids are entered in the system, a matching is determined and then the market is cleared, and the cycle repeats.

In our proposed framework a transaction in the e-barter trade system is therefore initiated by the following phases:

- Agents enter the e-marketplace and submit a bid description, modeled using a logical language(*supplies*/*demand*).
- Supply and demand descriptions are effectively matched trying to maximize agent utilities, taking into account (semantic) similarity between bid descriptions, the market price of each good and the barter trade constraints.
- Given the quantities and the type of good supplied/requested as well as the market price of each product, the broker tries to maintain the balance of trade, as requested in an e-barter system, solving an optimization problem, see Section 5.
- The broker finds the most promising matches among bids and proposes them to the agents in the e-marketplace. While finding the most promising matches, the broker has to take into account the utility of each agent. Such a utility is related to preferences that each agent expresses using our logical language (see Section 3).

In such a way it is possible both to balance the trade (as required by a barter trade system) and also find for each agent the most promising counterpart, based on the semantic similarity between bids.

3 The Logical Setting

In the rest of the paper we refer, for the sake of clarity and without loss of generality, to a mobile phone domain. Clearly, in an e-barter marketplace several categories of goods/services will be traded, and an agent looking for mobile phones will probably sell at the same time another good in the trade system. We assume the background knowledge $\mathcal{T}$ (*i.e.*, an ontology) be modeled using Description Logics (DL) [4].

3.1 Basic of Description Logics

Here, we provide a little survey on DLs, referring to [4] for a more comprehensive description. Description Logics (DLs) are a family of logic formalisms for Knowledge Representation, whose basic syntax elements are *concept names*, *properties* and *individuals*. Concepts names stands for set of objects in the domain (`WindowsMobile`, `Bluetooth`, `WebBrowser`) while properties link (sets of) objects in the domain (`hasOS`, `supportedNetwork`, `hasComponent`). Individuals are used for special named elements belonging to concepts (`NokiaN80`, `MotorolaRazor`).

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 A and a property name R we have:

$$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$$
$$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$$

The two symbols $\top$ and $\bot$ are used to represent the most generic concept and the most specific one 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*, *e.g.*, $\texttt{MobilePhone} \sqcap \exists \texttt{supportedNetwork}.\texttt{3G}$ describing the set of mobile phones supporting at least 3G (third generation) networks, and *universal role quantification e.g.*, $\texttt{MobilePhone} \sqcap \forall \texttt{hasOS}.\texttt{Symbian}$ describing the set of mobile phones having only Symbian operating system installed. The formal semantics of universal and existential quantification is as follows:

$$\exists R.C = \{x \in \Delta^{\mathcal{I}} \mid \exists y, (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$$
$$\forall R.C = \{x \in \Delta^{\mathcal{I}} \mid \forall y, (x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$$

Well formed formulas in DLs (in DLs jargon knows as **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. Basically, every DL allows one to form a *conjunction* of concepts, usually denoted as $\sqcap$; some DL include also disjunction $\sqcup$ and complement $\neg$ to close concept expressions under boolean operations.

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$$
$$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$$
$$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$$

The Description Logic closed under Boolean operators is referred as $\mathcal{ALC}$. Depending on the adopted Description Logic one is also allowed to use construct involving concrete domains as $\texttt{Camera} \sqcap \geq_2 \texttt{megaPixel}$ describing a camera with at least 2 megapixel of resolution. Notice that while properties, as `hasOS`, are mapped to a subset of of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, concrete properties, as `megaPixel` are mapped to a subset $\Delta^{\mathcal{I}} \times D$ where D is a concrete domain.

$$(\leq_k R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | R^{\mathcal{I}}(x) \leq k\}$$
$$(\geq_k R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | R^{\mathcal{I}}(x) \geq k\}$$
$$(=_k R)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} | R^{\mathcal{I}}(x) = k\}$$

Actually, more espressive operators can be added to this logic as number restrictions, transitive roles and inverse roles just to cite a few [4]. The expressiveness of a DL depends on the type of constructors allowed; we point out that our approach is completely independent of the particular DL chosen to describe the domain knowledge.

In order to formally represent the domain knowledge and constraints intercurring among elements of the domain, we can model an Ontology $\mathcal{T}$ (for Terminology) containing axioms $D \sqsubseteq C$ where D and C are well formed formulas in the adopted DL and $R \sqsubseteq S$ where both R and S are properties. 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 can also write $C \equiv D$ to represent both $C \sqsubseteq D$ and $D \sqsubseteq C$.

In the rest of the paper we refer to the Ontology $\mathcal{T}$ depicted in Figure 1.

$$\texttt{W-CDMA} \sqsubseteq \texttt{3G} \tag{1}$$
$$\texttt{WindowsMobile} \sqsubseteq \neg\texttt{Symbian} \tag{2}$$
$$\texttt{MobileOS} \equiv \texttt{WindowsMobile} \sqcup \texttt{Symbian} \tag{3}$$
$$\texttt{Camera} \sqsubseteq \texttt{Component}$$
$$\texttt{Display} \sqsubseteq \texttt{Component}$$
$$\texttt{Keyboard} \sqsubseteq \texttt{Component}$$
$$\texttt{Display} \sqsubseteq \neg\texttt{Camera}$$
$$\texttt{Display} \sqsubseteq \neg\texttt{Keyboard}$$
$$\texttt{Camera} \sqsubseteq \neg\texttt{Keyboard}$$
$$\exists\texttt{type} \sqsubseteq \texttt{Component} \tag{4}$$
$$\top \sqsubseteq \forall\texttt{hasComponent.Component} \tag{5}$$
$$\texttt{SmartPhone} \equiv \texttt{MobilePhone} \sqcap \exists\texttt{hasSoftware.WebBrowser} \sqcap \exists\texttt{hasComponent.}(\texttt{Display} \sqcap \exists\texttt{type} \sqcap \forall\texttt{type.Graphical}) \sqcap \exists\texttt{supportedNetwork.3G} \sqcap \exists\texttt{hasComponent.}(\texttt{Keyboard} \sqcap \exists\texttt{type.QWERTY}) \tag{6}$$
$$\exists\texttt{hasOS.Symbian} \sqsubseteq \exists\texttt{supports.MPEG4} \tag{7}$$

Fig. 1. Reference Ontology

In axiom (1) a simple subclass relation is represented (*W-CDMA is a 3G technology*). Axiom (4) forces the domain of `type` to be a `Component` while axiom 5 forces the range of `hasComponent` to be a `Component`. Noteworthy are also axiom (2) and axiom (3). The first one represents a disjointness relation (*a Symbian OS is not a Microsoft Windows Mobile OS and vice versa*). Together with axiom (3) (*Mobile operating systems are Microsoft Windows Mobile or Symbian ones*) they represent the complete partition of `MobileOS` [3]. Using axioms in the ontology we can also relate properties. This is the case of axiom (7) where it is stated that *a mobile phone equipped with Symbian Operating System supports the MPEG-4 mutimedimedia format*. Finally, axioms

[3] This is a simplified model of the mobile phones domain where many other operating systems exist.

can be used to define terms to be used as synonyms of complex descriptions as in axiom (6) where a *smart phone* is defined as *a mobile phone supporting web browsing, provided with a graphical display, 3G connectivity and a QWERTY keyboard*.

Using concepts and roles defined within the ontology $\mathcal{T}$, it is possible to describe items to be sold as well as items to be bought with corresponding preferences. As an example, consider three agents in the e-marketplace selling mobile phones[4] (for the sake of conciseness we translate in DL only the first description):

[S^1 – **LG enV VX9900**:] Mobile phone with a digital camera, 2 mega pixels and 4X digital zoom, W-CDMA network technology, supported media format: MPEG-4, 3gp, MP3.

$$\begin{aligned}\textbf{LG enV VX9900} = \\ \texttt{MobilePhone} \sqcap \exists \texttt{hasComponent.}(\texttt{Camera} \sqcap \\ =_2 \texttt{megaPixel} \sqcap =_4 \texttt{zoom}) \sqcap \\ \exists \texttt{supportedNetwork.W-CDMA} \sqcap \exists \texttt{supports.MPEG4} \sqcap \\ \exists \texttt{supports.3GP} \sqcap \exists \texttt{supports.MP3}\end{aligned}$$

[S^2 – **Nokia N95**:] Mobile phone with Digital Camera, 5.0 mega pixels and 10X zoom, WCDMA and GSM network technology, supported media format: WMA, AAC, MP3. Infrared, Wi-Fi and Bluetooth wireless technology, phone endowed with Symbian OS.

[S^3 – **Samsung BlackJack SGH i607**:] Smart phone with digital camera, 1.3 mega pixels and 2X digital zoom, supporting the MP3 format. Bluetooth wireless technology, phone endowed with Microsoft Windows Mobile Operating System.

On the other hand, let us suppose an agent **i** enters the e-marketplace looking for a "*mobile phone with a digital camera with at least 2 mega pixel resolution and digital zoom at least 4X, supporting MP3 format and optionally the MPEG-4, Bluetooth connection and infrared port, endowed preferably with a Symbian operating system, and a 3G mobile telephony communications protocol*". Clearly, in such a bid it is possible to distinguish ***strict requirements***, features the agent wants to be specified in the description of the item to be bought, *i.e.*, they have to be provided by the seller, and ***preferences***, features that, although not strictly necessary, make the agent more satisfied and happier.

***Strict requirements*:** "*mobile phone with a digital camera, supporting MP3 format, Bluetooth, endowed with a 3G mobile telephony communications protocol*"

***Preferences*:** "*at least 2 mega pixel resolution and digital zoom at least 4X, supporting MPEG-4 format, infrared port, endowed with a Symbian operating system*"

We can represent both strict requirements and preferences as DLs formulas.

$$\begin{aligned}B^{\mathbf{i}} = \texttt{MobilePhone} \sqcap \exists \texttt{hasComponent.Camera} \sqcap \\ \exists \texttt{supports.MP3} \sqcap \exists \texttt{connections.Bluetooth} \sqcap\end{aligned}$$

[4] The mobile phone descriptions we refer to have been taken from the web site: `http://shopping.yahoo.com` Category: Electronics → Cell-Phones.

$\exists$supportedNetwork.3G
("*mobile phone with a digital camera, supporting MP3 format, Bluetooth connection, endowed with a 3G mobile telephony communications protocol*")

$P_1^{\mathbf{i}}$ = $\geq_2$ megaPixel$\sqcap \geq_4$ zoom
("*at least 2 mega pixel resolution and digital zoom at least 4X*")

$P_2^{\mathbf{i}}$ = $\exists$supports.MPEG4
("*supporting the MPEG-4 format*")

$P_3^{\mathbf{i}}$ = $\exists$connections.Infrared
("*infrared port*")

$P_4^{\mathbf{i}}$ = $\forall$hasOS.Symbian
("*endowed with a Symbian operating system*")

Which will be the supply most suitable for the buyer's agent? How can we determine the items more appealing for the agent **i**? Looking at formulas, we can say that offer $S^{\mathbf{1}}$ does not satisfy the strict requirement for *Bluetooth*, while $S^{\mathbf{2}}$ and $S^{\mathbf{3}}$ fulfill all the strict requirements $B^{\mathbf{i}}$. Hence offer $S^{\mathbf{1}}$ has not to be considered as a promising partner for agent **i**. Notice that because of the axiom in the ontology $\exists$hasOS.Symbian $\sqsubseteq$ $\exists$supports.MPEG4 we know that the mobile phone described by $S^{\mathbf{2}}$ supports MPEG-4 format, even if this is not explicitely stated. Similarly, we know the same phone supports a 3G protocol because $\mathcal{T} \models$ W-CDMA $\sqsubseteq$ 3G. For what concerns $S^{\mathbf{3}}$ description, thanks to the axiom in the ontology relative to *smartphone* we know that the phone described by $S^{\mathbf{3}}$ supports a 3G communication protocol. Given the ontology $\mathcal{T}$, in formulas we have the following relations: $\mathcal{T} \not\models S^{\mathbf{1}} \sqsubseteq B^{\mathbf{i}}$, $\mathcal{T} \models S^{\mathbf{2}} \sqsubseteq B^{\mathbf{i}}$, $\mathcal{T} \models S^{\mathbf{3}} \sqsubseteq B^{\mathbf{i}}$ Similarly to the one for strict requirements, we can establish a criterion to decide, between offers $S^{\mathbf{2}}$ and $S^{\mathbf{3}}$, which is the one better fulfilling the buyer preferences. In other words we are going to define a logic-based utility function that allow to measure the satisfaction degree of an agent (see Section 4.1).

4 Bid Expressiveness

Let $\mathcal{L}$ be a Description Logic and $\mathcal{T}$ an ontology expressed as a set of formulas over $\mathcal{L}$.

We name the n agents of the e-barter as $\{$ **1**,...,**n** $\}$, and use **i** as a variable over $\{$ **1**,...,**n** $\}$. Hereafter, whenever we use indexes is with the following meaning: in $\bullet_k^{\mathbf{i}}$, **i** represents an agent, while k, the k-th element of a set. Each agent **i** looks for ("buys") some good $B^{\mathbf{i}}$, and offers ("sells") some other good $S^{\mathbf{i}}$. Both $B^{\mathbf{i}}$ and $S^{\mathbf{i}}$ are formulas in $\mathcal{L}$. We also let $\min QB^{\mathbf{i}}$, $\max QB^{\mathbf{i}}$, $\min QS^{\mathbf{i}}$, and $\max QS^{\mathbf{i}}$ be the *minimum and maximum quantities* that agent **i** is willing to buy and to sell, respectively. For the sake of simplicity, we assume that each agent "buys" only items of one type of good and "sells" items of only one other type of good.

4.1 Preferences and Utility Functions

If the agent **i** is willing to buy a certain quantity of a good, then it can formulate its request setting some characteristics as *strict* and others as *preferred*. *Strict* characteristics represent what has to be specified in a good description $S^{\mathbf{j}}$ in order to be considered by **i**. *Preferred* characteristics make **i** happier if $S^{\mathbf{j}}$ exposes them. Hence, if **i** is looking for some good to be traded, it expresses its request as a set of formulas:

$B^{\mathbf{i}}$: a Description Logic formula representing strict requirements;
$\{P_k^{\mathbf{i}}\}$: a set of Description Logic formulas representing preferred requirements.

The preferences of each agent $\mathbf{i}$ are formalized by a utility function $\mathsf{U}^{\mathbf{i}} : \mathcal{L} \mapsto \Re^+$, assigning a worth to $B^{\mathbf{i}}$ and to each formula in $\{P_k^{\mathbf{i}}\}$.

Of course, we assume that $\mathsf{U}^{\mathbf{i}}$ is very sparse, and only a few number of formulas in $\mathcal{L}$ have a non-zero utility, corresponding to those characteristics—single concept names, or generic Description Logics formula—an agent considers important. $P_k^{\mathbf{i}}$ are formulas to which agent $\mathbf{i}$ assigns some worth (*i.e.*, a non-zero utility). In formulas:

$$u^{ij} = \mathsf{U}^{\mathbf{i}}(B^{\mathbf{i}}) + \sum \{\mathsf{U}^{\mathbf{i}}(P^{\mathbf{i}}) \mid \mathcal{T} \models S^{\mathbf{j}} \sqsubseteq P^{\mathbf{i}}\} \tag{8}$$

where u_{ij} is the utility gained by agent $\mathbf{i}$ when buying good $S^{\mathbf{j}}$ from agent $\mathbf{j}$, computed as the sum of utilities set by $\mathbf{i}$ of characteristics which are fulfilled by good sold by $\mathbf{j}$.

We now explain why we require agents to put a utility over their strict requirements. In Classical Negotiation Theory [20], the existence of a *disagreement payoff* is always hypothesized. Such a payoff is the minimum utility each agent requires to pursue the negotiation, and usually represents both the attitude of an agent towards negotiation—a high disagreement payoff models the fact that the agent is rather unwilling to negotiate at all—and some fixed costs which be repaid by the agreement. Since the behavior of our agents is that if at least the strict requirements are fulfilled, they may accept the barter, it follows that such strict requirements should have a utility which is equal to, or greater than, the disagreement payoff hypothesized by the theory.

4.2 Prices

Prices could be set in two ways: we call them *exogenous* prices and *endogenous* prices.

In exogenous prices, we suppose that every offered good—whoever offers it—has a market price, given by the value of a global function $p(S^{\mathbf{i}}) \in \Re^+$, in barter dollars. This price is set by the barter autonomously with reference to the market price, and we require that whenever two agents $\mathbf{i}$ and $\mathbf{j}$ sell the same good—that is, when $\mathcal{T} \models S^{\mathbf{i}} \equiv S^{\mathbf{j}}$—then $p(S^{\mathbf{i}}) = p(S^{\mathbf{j}})$[5].

On the other hand, endogenous prices are fixed by the well-known result by Arrow and Debreu [3] that states that there exists a unique price vector $\boldsymbol{p}$ such that if every agent maximizes her own utility, the market clears, subject to the constraints that every agent cannot spend more than the worth she initially owes. Deng et al. [8] proved that the result can be extended to indivisible goods—as in our case—and that one can find a price that minimizes the deficiency of the market, although that price is hard to approximate.

We now discuss the two options. Endogenous prices are advisable for a *closed* market; one in which no other agent can enter in the future, and that must find an equilibrium in itself. In fact, Arrow& Debreu's result does not take into account any intrinsic value of the bartered objects—*e.g.*, their production cost. So it may happen that an agent

[5] Observe that since in $\mathcal{L}$ there could be more than one way of describing the same good—*e.g.*, in the simplest case, two synonyms, made equivalent by a formula in $\mathcal{T}$—we use logical equivalence $S^{\mathbf{i}} \equiv S^{\mathbf{j}}$ to express the fact that agent $\mathbf{i}$ and agent $\mathbf{j}$ sell in fact the same good.

selling a good which is not required by other agents at a given moment of the e-barter marketplace gets a very low price for it, because the utility that the other agents assign to the good is very low. However, the situation might change when another agent, requiring exactly that good, enters the marketplace. These considerations suggest us to opt for exogenous prices. Hence, from now on, we assume that every good has a price which is fixed by the barter independently of the market status.

5 The Barter Trade Optimization

Our optimization problem is finding non-negative integral values for n^2-n variables q^{ij} $i, j \in \{\mathbf{1}, \ldots, \mathbf{n}\}, \mathbf{i} \neq \mathbf{j}$, representing the quantity of good sold by agent $\mathbf{j}$ to agent $\mathbf{i}$, at the price of $p(S^{\mathbf{j}})$ barter dollars. Each variable q^{ij} is subject to the following constraints:

$$\min QB^{\mathbf{i}} \leq \textstyle\sum_j q^{ij} \leq \max QB^{\mathbf{i}} \tag{9}$$

$$\min QS^{\mathbf{j}} \leq \textstyle\sum_i q^{ij} \leq \max QS^{\mathbf{j}} \tag{10}$$

These constraints express the fact that globally, the quantities traded by agents should be within the range they specified. Intuitively, an agent might not want to exchange a good below a given minimal quantity in order to, say, reduce its marginal costs, or reduce packaging and shipping costs. Analogously, an upper bound on the quantity could model production/consumption physical limits.

Moreover, we force $q^{ij} = 0$ if the strict characteristics of the good required by agent $\mathbf{i}$ are not implied by the ones offered by agent $\mathbf{j}$, *i.e.*, if $\mathcal{T} \not\models S^{\mathbf{j}} \sqsubseteq B^{\mathbf{i}}$.

To ease summations, we also let $q^{ii} = 0$ for n "fake" variables. Then, we let n new variables $b^{\mathbf{i}}, \ldots, b^{\mathbf{n}}$ ("balances") be defined by

$$b^{\mathbf{i}} = \sum_j q^{ji} \cdot p(S^{\mathbf{i}}) - \sum_j q^{ij} \cdot p(S^{\mathbf{j}}) \cdot (1/u^{ij}) \tag{11}$$

We now explain Formula (11). The balance of an agent is usually made by the barter dollars gained by giving (several items of) good $S^{\mathbf{i}}$ minus the barter dollars it owes the barter exchange to get (several items of) good $B^{\mathbf{i}}$. Hence, one would expect the balance for the agent $\mathbf{i}$ to be defined simply by $b^{\mathbf{i}} = \sum_j q^{ji} \cdot p(S^{\mathbf{i}}) - \sum_j q^{ij} \cdot p(S^{\mathbf{j}})$ However, we have to remember that the items that Agent $\mathbf{i}$ gets might not be exactly the ones it looked for: some preferred characteristics might not be satisifed by the items it buys. We have to weigh the barter dollars spent with "how bad" are the bought items w.r.t. $\mathbf{i}$ preferences. This is the reason why, in Formula (11), barter dollars spent by $\mathbf{i}$, *i.e.*, $\sum_j q^{ij} \cdot p(S^{\mathbf{j}})$, are scaled by its corresponding utility. This allows us to compare different matches and hence discover the most promising ones, as the higher is the value of the logic-based utility function the higher will be the semantic similarity between the two bids.

Let us clarify the idea behind the flow of barter dollars with the aid of a graph representation. Starting from $B^{\mathbf{i}}$ and $S^{\mathbf{j}}$, with $i, j \in \{\mathbf{1}, \ldots, \mathbf{n}\}, \mathbf{i} \neq \mathbf{j}$, we can build a weighted directed graph (see Figure 2) following these simple rules:

— for each agent, draw a node and label it with its name;
— given two nodes $\mathbf{i}$ and $\mathbf{j}$, if $\mathcal{T} \models S^{\mathbf{j}} \sqsubseteq B^{\mathbf{i}}$ then draw an edge from node $\mathbf{i}$ to node $\mathbf{j}$;

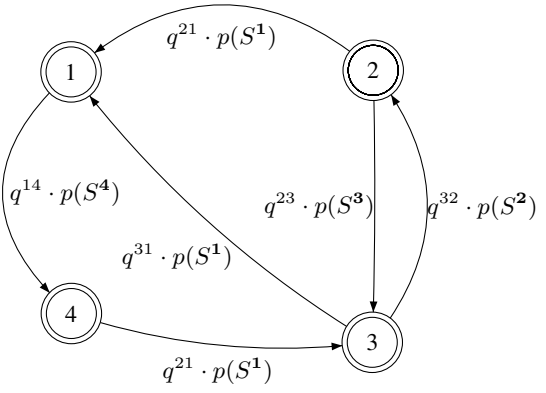

Fig. 2. The flow of *barter dollars*: Nodes correspond to agents, edges (**i**,**j**) to barter dollars agent **i** will pay to agent **j** in exchange of q^{ij} items of Supply $S^{\mathbf{j}}$

— assign to each edge from node **i** to node **j**, a label $q^{ij} \cdot p(S^{\mathbf{j}})$ representing the barter dollars **i** will pay to **j**.

Given an agent **i**, the corresponding balance $b^{\mathbf{i}}$ is computed as the summation of weights asociated to the edges (i, j) and (j, i). Weights labelling (i, j) edges represent barter dollars **i** pays to **j** —their value is considered negative in the summation in equation (11)— while the ones labelling (j, i) edges represent barter dollars that **i** receives from **j** —their value is considered positive in the summation in equation (11).

We impose that the balances are "close enough" to zero with the following constraints:

$$-\varepsilon \leq b^{\mathbf{i}} \leq \varepsilon \quad \text{for } \mathbf{i} = \mathbf{1}, \ldots, \mathbf{n} \tag{12}$$

for a suitable value for ε. When the exchange of goods lasts several rounds (possibly, forever) we record in each round the value of each agent's balance $pb^{\mathbf{i}}$, and modify (12) as $-\varepsilon \leq b^{\mathbf{i}} + pb^{\mathbf{i}} \leq \varepsilon$, in order to make compensations for unbalanced agents in subsequent rounds.

Finally, our *objective function* is: $\max \sum_{ij} q^{ij}$, in order to maximize the barter capabilities of the barter exchange.

Taking the overall system of disequations (9)–(12), we get a Mixed-Integer Programming, solvable with standard techniques [15].

We remark the fact that our framework is a true extension of the proposals for barter exchange regarding fixed goods, as the one of Haddawy et al. [14]: in fact, it is sufficient to set the utility of each $B^{\mathbf{i}}$ to 1, and to let every agent have no preferences. In this case, Formula (11) becomes the usual balance for trade dollars.

6 Related Work

Literature on e-marketplaces is huge and ever increasing, we refer the interested reader to [29,19] and focus this section on works having some relationship with our approach.

With reference to logic-based matchmaking, there has been a growing interest, motivated by the Semantic Web initiative. Matchmaking as satisfiability of concept conjunction in DLs was first proposed in the same venue by Gonzales-Castillo et al. [12] and by Di Sciascio et al. [11], and precisely defined by Bartolini et al. [28]. A specific language for agent advertisement in the framework of the Retsina Multiagent infrastructure was proposed in [26]. A matchmaking engine was developed [27,22], which carries out the process on five possible levels. Such levels exploit both classical text-retrieval techniques and semantic match using Θ-subsumption. Nevertheless, standard features of a semantic-based system, as satisfiability check were unavailable. It is noteworthy that in this approach, the notion of *plug-in* match was borrowed from research on matching software components [30], to overcome in some way the limitations of a matching approach based on exact matches. Two new levels for matching classification, along with properties that a matchmaker should have in a DL-based framework, and algorithms to classify and semantically rank matches within classes were introduced by Di Noia et al. [10]. The Difference Operator in DLs for semantic matchmaking was proposed by Benatallah et al. [5]. The approach uses Concept Difference, followed by a covering operation optimized using hypergraph techniques, in the framework of web services discovery. An initial DL-based approach adopting penalty functions ranking was proposed by Calí et al. [6], in the framework of dating systems. An extended matchmaking approach, with negotiable and strict constraints in a DL framework has been proposed by Colucci et al. [7], using both Concept Contraction and Concept Abduction [9]. Matchmaking in DLs with locally-closed world assumption applying autoepistemic DLs has been proposed by Grimm et al. [13]. The need to work in someway with approximation and ranking in DL-based approaches to matchmaking has also recently led to adopting fuzzy-DLs, as proposed by Ragone et al. [24] and in Smart [1] or hybrid approaches, as in the OWLS-MX matchmaker [17]. Lukasiewicz and Schellhase propose [18] a language able to express conditional preferences to matchmake in Description Logics based on *strength* values (*i.e.*, weights) assigned to preferences. A ranking procedure was also proposed. The main aim of the approach was to retrieve a set of appealing available resources with respect to a request.

Nevertheless in all such approaches the matchmaking process is defined according to one player's perspective, according to a different purpose. Namely, the purpose is to rank a set of promising offers according to buyer's preferences or viceversa. In our framework we model the matchmaking process as a many-to-many one, taking into account both buyers' and sellers' preferences.

As first pointed out by Segev and Beam [25] the role of an electronic mediator in marketplaces is becoming increasingly important, as the broker can help agents to *search* for potential partners as well as to *negotiate*. Moreover, being a trusted third party, it can collect information from the agents and then suggest to the parties win-win solutions otherwise difficult to discover by agents themselves. Segev and Beam assumed in their model that for each product the quantity requested/supplied is equal to one. The negotiation process is based on price, basically the seller's bid with a price lower than the buyer's one is chosen. So they do not consider in their analysis either the problem of different quantities that can be requested/supplied or the semantic similarity between bids. Haddawy et al. [14] modeled the trade balance problem as a minimum cost

circulation problem (MCC) on a network. They addressed the typical problem in a barter exchange: matching buyers and sellers such that the trade volume is maximized while the balance of trade is maintained as much as possible. Furthermore the matching algorithm presented by Haddaway et al. is a *quantitative* one; they suppose that goods requested/supplied in the e-marketplace are exactly the same. Therefore no semantic relation among goods is taken into account, neither are agents' preferences, while it is intuitive that, as long as the market does not deal only with undifferentiated goods, structured and complex descriptions should be taken into account, and preferences handling allows one to find many more possible matches and business opportunities.

Núñez et al.[21] modeled an e-barter system using a hierarchical structure. Agents are grouped in local markets and when, in turn, each market gets completed, an agent, representing the whole market, is created and it is grouped in a new market. A utility function models agents preferences on basket of goods, hence only quantitative preferences are taken into account and no semantic relations among attributes are modeled. In our framework we define a *logic-based* utility function, thus taking into account both qualitative and quantitative preferences. The use of a logical language enhances the bid expressiveness and it allows one to catch semantic relations between attributes, through basic inference services such as subsumptions and satisfiability. Ragone et al. [23] modeled a negotiation process among agents in an open e-marketplace and semantic relations among attributes are taken into account, using a propositional logic to model preferences. Here we use much more expressive DLs and introduce an approach to deal with quantities and to balance the (closed) barter market.

7 Conclusion

We presented a knowledge-based approach to e-barter trading, exploiting a broker. In particular we focused here on an approach to find most promising matches, in a many-to-many matchmaking process, between supplies and demands, taking into account not only the price and quantities as in current barter trade systems, but also a semantic similarity among bid descriptions, while keeping exchanges balanced.

We proposed the use of a logical language to express agent preferences, thereby enhancing bid expressiveness and defined a logic-based utility function that allows to evaluate the semantic similarity between bids. Finally we outlined the optimization problem to be solved in order to clear the market.

A future development of this research is the adoption of a fuzzy DL for modeling the partial fulfillment of a preference [24] as fuzzy subsumption. An evaluation, to numerically support our approach is currently under way with selected SMEs in the framework of Apulia Region DIPIS project, while future research work will be devoted to extend our framework to a *call market*, and generally speaking to auction-based markets.

Acknowledgements

We thank the three anonymous reviewers for comments that helped us improving the paper. This paper was written while Azzurra Ragone was visiting the Artificial Intelligence Laboratory of University of Michigan at Ann Arbor, which she thanks for

hospitality. This research was funded in part by projects PS_092 *DIPIS* and EU-FP6-IST-026896 *TOWL*.

References

1. Agarwal, S., Lamparter, S.: Smart - a semantic matchmaking portal for electronic markets. In: Proceedings of the 7th International IEEE Conference on E-Commerce Technology CEC 2005, pp. 405–408 (2005)
2. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows. Prentice-Hall, Englewood Cliffs (1993)
3. Arrow, K.J., Debreu, G.: Existence of an equilibrium for a competitive economy. Econometrica 22(3), 265–290 (1954)
4. Baader, F., Calvanese, D., Mc Guinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook. Cambridge University Press, Cambridge (2002)
5. Benatallah, B., Hacid, M.-S., Rey, C., Toumani, F.: Request Rewriting-Based Web Service Discovery. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 242–257. Springer, Heidelberg (2003)
6. Calì, A., Calvanese, D., Colucci, S., Di Noia, T., Donini, F.M.: A description logic based approach for matching user profiles. In: Proceedings of the 17th International Workshop on Description Logics (DL 2004). CEUR Workshop Proceedings, vol. 104 (2004)
7. Colucci, S., Di Noia, T., Di Sciascio, E., Donini, F., Mongiello, M.: Concept Abduction and Contraction for Semantic-based Discovery of Matches and Negotiation Spaces in an E-Marketplace. Electronic Commerce Research and Applications 4(4), 345–361 (2005)
8. Deng, X., Papadimitriou, C., Safra, S.: On the complexity of equilibria. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing STOC 2002, pp. 67–71. ACM, New York (2002)
9. Di Noia, T., Di Sciascio, E., Donini, F.: Semantic Matchmaking as Non-Monotonic Reasoning: A Description Logic Approach. Journal of Artificial Intelligence Research 29, 269–307 (2007)
10. Di Noia, T., Di Sciascio, E., Donini, F., Mongiello, M.: A system for principled matchmaking in anelectronic marketplace. International Journal of Electronic Commerce 8(4), 9–37 (2004)
11. Di Sciascio, E., Donini, F., Mongiello, M., Piscitelli, G.: A Knowledge-Based System for Person-to-Person E-Commerce. In: Proceedings of the KI 2001 Workshop on Applications of Description Logics (ADL 2001). CEUR Workshop Proceedings, vol. 44 (2001)
12. Gonzales-Castillo, J., Trastour, D., Bartolini, C.: Description Logics for Matchmaking of Services. In: Proceedings of the KI 2001 Workshop on Applications of Description Logics (ADL 2001). CEUR Workshop Proceedings, vol. 44 (2001)
13. Grimm, S., Motik, B., Preist, C.: Matching Semantic Service Descriptions with Local Closed-World Reasoning. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 575–589. Springer, Heidelberg (2006)
14. Haddawy, P., Cheng, C., Rujikeadkumjorn, N., Dhananaiyapergse, K.: Optimizing ad hoc trade in a commercial barter trade exchange. Electronic Commerce Research and Applications 4(4), 299–314 (2005)
15. Hillier, F., Lieberman, G.: Introduction to Operations Research. McGraw-Hill, New York (2005)
16. International Reciprocal Trade Association. Reciprocal trade statistics (2006), http://irta.com/
17. Klusch, M., Fries, B., Khalid, M., Sycara, K.: Owls-mx: Hybrid owl-s service matchmaking. In: Proceedings of 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web (2005)

18. Lukasiewicz, T., Schellhase, J.: Variable-strength conditional preferences for matchmaking in description logics. In: Proc. of KR 2006, pp. 164–174 (2006)
19. MacKie-Mason, J., Wellman, M.: Automated markets and trading agents. In: Tesfatsion, L., Judd, K.L. (eds.) Handbook of Computational Economics. Agent-Based Computational Economics, vol. 2. North-Holland, Amsterdam (2006)
20. Nash, J.F.: The bargaining problem. Econometrica 18(2), 155–162 (1950)
21. Núñez, M., Rodríguez, I., Rubio, F.: Formal specification of multi-agent e-barter systems. Science of Computer Programming 57(2), 187–216 (2005)
22. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic Matching of Web Services Capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, p. 333. Springer, Heidelberg (2002)
23. Ragone, A., Di Noia, T., Di Sciascio, E., Donini, F.: Extending propositional logic with concrete domains in multi-issue bilateral negotiation. In: Baldoni, M., Son, T.C., van Riemsdijk, M.B., Winikoff, M. (eds.) DALT 2007. LNCS, vol. 4897, pp. 211–226. Springer, Heidelberg (2008)
24. Ragone, A., Straccia, U., Di Noia, T., Di Sciascio, E., Donini, F.: Vague knowledge-bases for matchmaking in p2p e-marketplaces. In: Franconi, E., Kifer, M., May, W. (eds.) ESWC 2007. LNCS, vol. 4519, pp. 414–428. Springer, Heidelberg (2007)
25. Segev, A., Beam, C.: Brokering strategies in electronic commerce markets. In: Proc. of ACM Conference on Electronic Commerce, pp. 167–176 (1999)
26. Sycara, K., Paolucci, M., Van Velsen, M., Giampapa, J.: The RETSINA MAS infrastructure. Autonomous agents and multi-agent systems 7, 29–48 (2003)
27. Sycara, K., Widoff, S., Klusch, M., Lu, J.: LARKS: Dynamic Matchmaking Among Heterogeneus Software Agents in Cyberspace. Autonomous agents and multi-agent systems 5, 173–203 (2002)
28. Trastour, D., Bartolini, C., Priest, C.: Semantic Web Support for the Business-to-Business E-Commerce Lifecycle. In: Proc. International World Wide Web Conference (WWW 2002), pp. 89–98. ACM, New York (2002)
29. Wellman, M.: Online marketplaces. In: Singh, M.P. (ed.) Practical Handbook of Internet Computing. CRC Press, Boca Raton (2004)
30. Zaremski, A.M., Wing, J.M.: Specification matching of software components. ACM Transactions on Software Engineering Methodologies 6(4), 333–369 (1997)

Inductive Negotiation in Answer Set Programming

Chiaki Sakama

Department of Computer and Communication Sciences
Wakayama University, Sakaedani, Wakayama 640-8510, Japan
sakama@sys.wakayama-u.ac.jp

Abstract. This paper provides a logical framework of negotiating agents who have capabilities of evaluating and building proposals. Given a proposal, an agent decides whether it is acceptable or not. If the proposal is unacceptable as it is, the agent seeks conditions to accept it. This attitude is captured as a process of making hypotheses by *induction*. If an agent fails to find a hypothesis, it would concede by giving up some of its current belief. This attitude is characterized using *default reasoning*. We provide a logical framework of such think-act cycle of an agent, and develop a method for computing proposals using *answer set programming*.

1 Introduction

Negotiation is a process of reaching agreement between different agents. In a typical one-to-one negotiation, an agent makes a proposal on his/her request and the opponent agent decides whether it is acceptable or not. If it is unacceptable, the opponent tries to make a counter-proposal. Negotiation proceeds in a series of rounds and each agent makes a proposal at every round until it reaches a (dis)agreement. Our primary interest of this paper is a process of evaluation and construction of proposals. A proposal is acceptable if it does not conflict with the interest of an agent. When a proposal is unacceptable for an agent, he/she seeks conditions to accept it. Those conditions would be found by updating his/her current beliefs: in one way, by introducing new beliefs, and in another way, by giving up some of his/her current belief.

Consider the following dialogue between a buyer B and a seller S (subscripts represent rounds in negotiation).

B_1: "I want an external HDD with 200GB".
S_1: "It costs 120USD".
B_2: "I want to get it at 100USD."
S_2: "We can provide it at the discount price if you pay by cash."
B_3: "I don't want to pay by cash".
S_3: "We can provide an external HDD with 180GB at 100USD".
B_4: "OK, I accept it".

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 143–160, 2009.

In this dialogue, the buyer does not accept the initial offer S_1 made by the seller. Then, the buyer made a new proposal B_2 for a discount price. In response to this, the seller provides a condition to meet the request (S_2). The buyer does not accept it (B_3), and the seller proposes downgrade of the product (S_3). The buyer accepts it, and negotiation ends.

In the second round, the seller seeks conditions to accept B_2. The process of finding a condition to accept a proposal is logically characterized as follows. Suppose a knowledge base K represented by a first-order theory, and a proposal G represented by a formula. Then, K could accept G under the condition H if the next relation holds:

$$K \cup H \models G\,.$$

Here, H is a set of formulas and bridges the gap between the current belief K of an agent and the request G made by another agent. At this point, there are structural similarities between the problem presented above and the problem of *induction*, a method of machine learning in artificial intelligence. In fact, viewing G as an observed evidence, the problem of finding H is considered a process of building a *hypothesis* to explain G under K. Induction is an ampliative reasoning and extends the original theory to explain observed new phenomena. In negotiation, an agent also extends his/her original belief to accommodate another agent's request. Back to the negotiation dialogue, in response to the proposal S_3, the buyer concedes to accept it. This is done by withdrawing her original request. The process of concession is also formulated as follows. Given a knowledge base K of an agent and a proposal G by another agent, K could conditionally accept G by concession if the next relation holds:

$$(K \setminus J) \cup H \models G\,.$$

Here, J is a part of belief included in K, which could be given up to accept G.

In this paper, we provide a logical framework of negotiating agents who have capabilities of evaluating and building proposals. We first consider an agent who has a knowledge base represented by first-order logic and characterize a process of making proposals using induction. We show that different types of proposals are built in terms of induction. Next, we formulate a process of making a concession in negotiation. We show that concession is done by inference from a default theory. Finally, the proposed method is realized using *answer set programming* [9], a logic programming framework for nonmonotonic reasoning. The rest of this paper is organized as follows. Section 2 characterizes processes of building proposals and making a concession in terms of induction and default inference. Section 3 provides methods for computing proposals in answer set programming. Section 4 discusses related work, and Section 5 concludes the paper. This is an extended version of the paper [18]. Section 3 and a part of Section 4 are entirely new, and all proofs of technical results, which are not in [18], are attached.

2 Negotiation by Induction

2.1 Induction

A *first-order theory* is a set of formulas defined over the first-order language. The definition of the first-order language is the standard one in the literature. A first-order theory T *entails* a formula F (written as $T \models F$) if F is *true* in every model of T. A first-order theory T is *consistent* if it has a model; otherwise, T is *inconsistent*.

Induction in first-order logic is defined as follows. Let K be a consistent first-order theory which represents the *background knowledge base*. Given a formula G as an *observation* and $K \not\models G$, *induction* produces a set H of formulas as a *hypothesis* satisfying the condition:

$$K \cup H \models G \tag{1}$$

where $K \cup H$ is consistent. When H satisfies the above condition, we say that a hypothesis H *covers* (or *explains*) G with respect to K. This type of induction is known as *explanatory induction* [4] and is popularly used in the context of *inductive logic programming* [12].

Example 1. Suppose the knowledge base K and the observation G:[1]

$$K: \ swan(a) \wedge swan(b),$$
$$G: \ white(a) \wedge white(b).$$

Then,

$$H: \ \forall x \ (swan(x) \rightarrow white(x))$$

covers G with respect to K.

Thus, induction extends a knowledge base to cover an observation. We use induction to characterize the process of building proposals in the next section.

2.2 Building Proposal

We consider an agent who has a knowledge base K represented by a consistent first-order theory.

Definition 1. (proposal) A *proposal* G is a formula. In particular, G is called a *critique* if $G = accept$ or $G = reject$ where *accept* and *reject* are the reserved propositions.

A critique is a response as to whether or not a given proposal is accepted. It is decided by evaluating a proposal in a knowledge base of an agent.

[1] Throughout the paper, we shall omit braces { } in examples to represent the sets K and H of formulas, but the meaning is clear from the context.

Definition 2. (acceptability) Given a knowledge base K and a proposal G,

- G is *accepted* in K if $K \models G$.
- G is *acceptable* in K if $K \cup \{G\}$ is consistent.
- G is *unacceptable* (or *rejected*) in K if $K \cup \{G\}$ is inconsistent.

If a proposal G made by an agent Ag_1 is accepted/rejected by another agent Ag_2, Ag_2 returns the critique *accept*/*reject* to Ag_1. On the other hand, if a proposal is acceptable, an agent seeks conditions to accept it.

Definition 3. (conditional acceptance) Given a knowledge base K and a proposal G, G is *conditionally accepted* (with H) in K if

$$K \cup H \models G \tag{2}$$

holds for a set H of formulas such that $K \cup H$ is consistent. A set H of formulas is called an *accepting set of conditions* (with respect to K and G). In particular, H is called a *minimal* accepting set of conditions if H is a minimal set (under set inclusion) satisfying (2).

By the definition, it is easily seen that G is conditionally accepted in K if and only if it is acceptable in K. The notion of acceptance in Definition 2 is a special case of conditional acceptance with $H = \emptyset$. By Definition 3, we can see that the problem of finding a condition H for accepting a proposal G is identical to the problem of finding inductive hypothesis in (1). That is, by viewing a proposal G as an observation, an accepting set H of conditions is considered a hypothesis which covers G with respect to the background knowledge base K. This correspondence is not only in the definition of formulas, but also in the ground of their usage. In induction, when an agent observes a new evidence that cannot be explained in its current knowledge base, the agent induces a hypothesis which well accounts for the evidence and updates the knowledge base if necessary. In negotiation, on the other hand, an agent also observes a new proposal that is not entailed by its current knowledge base. Then, the agent constructs a hypothesis which well accounts for the proposal. Among accepting sets of conditions, we are interested in minimal accepting sets of conditions which represent minimal requirements for accepting a proposal. For this reason, we hereafter consider minimal accepting sets of conditions unless stated otherwise.

There are different types of accepting sets of conditions satisfying the relation (2). We provide some typical types of proposals in negotiation based on this definition. Suppose that an agent Ag_1 makes a proposal G and another agent Ag_2 who has a knowledge base K builds a counter-proposal in response to G.

Consent: When $H = \{G\}$, it holds that $K \cup H \models G$. In this case, Ag_2 accepts a proposal G if it is acceptable. Then, Ag_2 returns the critique $G' = accept$ to Ag_1.

Constraint: When $H = \{ G \wedge C \}$, it holds that $K \cup H \models G$. In this case, Ag_2 accepts a proposal G with a constraint C. Then, Ag_2 returns the counter-proposal $G \wedge C$ to Ag_1. For example, let $on(weekday) \rightarrow \neg\, go(restaurant)$ be a rule in K. Then, given $G = go(restaurant)$, $C = on(weekend)$ represents a constraint for accepting G.

Generalization: When $H = \{ G' \}$ such that $G'\theta = G$ for some substitution θ, it holds that $K \cup H \models G$. In this case, Ag_2 returns the counter-proposal G' which is more general than G. For example, given $G = show\_product(TV, b)$ with some specific brand-name b, $G' = show\_product(TV, x)$ with a variable x represents TV of any brand.

Subsumption: When H is a concept which subsumes G and K contains subsumption knowledge between H and G, it holds that $K \cup H \models G$. In this case, Ag_2 returns a counter-proposal H to Ag_1. For example, let $G = go(bookstore)$ and K contains $go(shopping\text{-}mall) \rightarrow go(bookstore)$, then $go(shopping\text{-}mall)$ becomes a counter-proposal.

Implication: When $H = \{ F \rightarrow G \}$ and $K \cup H \models G$, F represents a condition to accept G. In this case, Ag_2 returns the counter-proposal F to Ag_1. For example, let $G = want(chocolate)$ and K contains $want(biscuit)$, then $H = \{ want(biscuit) \rightarrow want(chocolate) \}$ represents exchange of sweets and $want(biscuit)$ becomes a counter-proposal.

In the above, *Consent* characterizes very generous attitude of an agent. *Constraint* and *Generalization* are considered special cases of *Implication* as both $G \wedge C \rightarrow G$ and $G' \rightarrow G'\theta$ hold. *Subsumption* is also a special case of *Implication* such that K contains a dependence relation between F and G. In case of subsumption, *abduction* [8] is used for the purpose instead of induction. Abduction is also hypothetical reasoning satisfying the relation (1). In contrast to induction which constructs a rule $F \rightarrow G$ from K and G, abduction extracts a fact F from G and a rule $F \rightarrow G$ which is derived from K.

2.3 Concession

An agent rejects a proposal if it is unacceptable. On the other hand, an agent can take an action of *concession* if he/she wants to reach an agreement in negotiation. To characterize agents who may concede in negotiation, we suppose agents who have two different types of knowledge: the one is strong belief and the other is weak belief. Strong belief is persistent belief or strong desire that cannot be abandoned. By contrast, weak belief can be given up depending on situation. Formally, a first-order theory K is divided into two disjoint sets:

$$K = \Sigma \cup \Gamma$$

where Σ represents *strong belief* and Γ represents *weak belief*. We assume that an agent gives up weak belief but not strong one when he/she makes a concession.

Definition 4. (acceptable by concession) Let K be a knowledge base such that $K = \Sigma \cup \Gamma$ as above. Then, a proposal G is *acceptable by concession* in K if there is a set J of formulas such that $J \subseteq \Gamma$ and $(K \setminus J) \cup \{G\}$ is consistent.

Definition 5. (conditional acceptance by concession) Let K be a knowledge base such that $K = \Sigma \cup \Gamma$. Then, a proposal G is *conditionally accepted by concession* (with H) in K if

$$(K \setminus J) \cup H \models G \tag{3}$$

holds for some sets H and J of formulas such that $J \subseteq \Gamma$ and $(K \setminus J) \cup H$ is consistent. A set J of formulas is called an *accepting set of concessions* (with respect to K and G). In particular, J is called a *minimal* accepting set of concessions if J is a minimal set (under set inclusion) satisfying (3).

Proposition 1 *A proposal G is conditionally accepted by concession in K iff G is acceptable by concession in K.*

Proof. If G is acceptable by concession in K, $(K \setminus J) \cup \{G\}$ is consistent for some $J \subseteq \Gamma$. As $(K \setminus J) \cup \{G\} \models G$, G is conditionally accepted by concession in K. Conversely, if G is conditionally accepted by concession in K, $(K \setminus J) \cup H \models G$ holds for some $J \subseteq \Gamma$ and H such that $(K \setminus J) \cup H$ is consistent. Then, any model M of $(K \setminus J) \cup H$ satisfies G, so M becomes a model of $(K \setminus J) \cup G$. Hence, $(K \setminus J) \cup G$ is consistent and G is acceptable by concession. □

Comparing Definition 5 with Definition 3, concession may give up (a part of) the current belief of an agent for accepting proposals. In particular, the relation (3) reduces to (2) when $J = \emptyset$. We assume that an agent wants to give up his/her current belief as little as possible, so we hereafter consider minimal accepting sets of concessions as well as minimal accepting sets of conditions.

Example 2. ([13]) Suppose that an agent Ag_1 has the knowledge base K:

$$
\begin{aligned}
f_1 &: \ have(mirror) \wedge have(nail) \rightarrow hang(mirror),\\
f_2 &: \ have(mirror) \wedge have(screw) \rightarrow hang(mirror),\\
f_3 &: \ give(nail) \rightarrow \neg have(nail),\\
f_4 &: \ have(screw) \rightarrow give(nail),\\
f_5 &: \ \forall x\, get(x) \rightarrow have(x),\\
f_6 &: \ have(mirror),\\
f_7 &: \ have(nail),
\end{aligned}
$$

where the strong belief Σ consists of f_1–f_6 and the weak belief Γ consists of f_7. The meaning of each formula is: f_1 and f_2 represent conditions to hang a mirror. If Ag_1 gives a nail, he/she does no longer have the nail (f_3). If Ag_1 has a screw, he/she can give a nail (f_4). If one gets an object, one has the object (f_5). Ag_1 has both a mirror (f_6) and a nail (f_7). Suppose that Ag_1 has the intention of hanging a mirror. Consider that another agent Ag_2 makes the request

$$G : \ give(nail).$$

This proposal is unacceptable in K because $K \cup \{G\}$ is inconsistent. The agent Ag_1 may reject G with this reason, but he/she could look for conditions for concession. Ag_1 finds the solution

$$J : \ have(nail)$$

and

$$H : \ get(screw)$$

where $(K \setminus J) \cup H$ is consistent and satisfies the relation $(K \setminus J) \cup H \models G$. Then, Ag_1 offers a counter-proposal H to Ag_2.

Our next question is how to distinguish different types of belief in both syntactic and semantic ways. For this purpose, we use *default logic* [14] for representing a knowledge base. Default logic distinguishes two types of knowledge as first-order formulas and default rules. Formally, a *default theory* is defined as a pair $\Delta = (D, W)$ where D is a set of default rules and W is a set of first-order formulas. A default rule (or simply *default*) is of the form:

$$\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma}$$

where $\alpha, \beta_1, \ldots, \beta_n$ and γ are quantifier-free formulas and respectively called the *prerequisite*, the *justifications* and the *consequent*. A default is *ground* if it contains no variable. Any default with variables represents the set of its ground instances over the language of Δ. As defaults and first-order formulas are syntactically distinguishable, we often put a default theory $\Delta = W \cup D$ for convenience. A set S of formulas is *deductively closed* if $S = Th(S)$ where Th is the deductive closure operator as usual. A set E of formulas is an *extension* of (D, W) if it coincides with a minimal deductively closed set E' of formulas satisfying the conditions: (i) $W \subseteq E'$, and (ii) for any ground default $\alpha : \beta_1, \ldots, \beta_n / \gamma$ from D, $\alpha \in E'$ and $\neg\beta_i \notin E$ $(i = 1, \ldots, n)$ imply $\gamma \in E'$. An extension E is *consistent* if E is a consistent set of formulas. A default theory may have none, one or multiple extensions in general.

To represent weak belief of an agent, we use default rules of the form:

$$\frac{: \gamma}{\gamma}. \tag{4}$$

This type of rule is called *super-normal* and a *super-normal default theory* is a default theory in which every default has the form (4). The rule (4) is read as "if it is consistent to assume γ, then believe γ". We represent weak belief of an agent by super-normal defaults in D, and distinguish them from strong belief represented by first-order formulas in W.

Definition 6. (default representation) Let K be a first-order theory such that $K = \Sigma \cup \Gamma$. Then, a *default representation* of K is defined as a super-normal default theory $\Delta_K = (D, W)$ such that $D = \{ \frac{:\gamma}{\gamma} \mid \gamma \in \Gamma \}$ and $W = \Sigma$.

Concession is characterized in a default theory as follows.

Theorem 2. *Let K be a first-order theory such that $K = \Sigma \cup \Gamma$.*

(i) A proposal G is acceptable by concession in K iff $\Delta_K \cup \{G\}$ has a consistent extension.

(ii) A proposal G is conditionally accepted by concession with H in K iff $\Delta_K \cup H$ has a consistent extension E such that $G \in E$.

Proof. (i) Let $\Delta_K = (D, W)$ be a default representation of K. When G is acceptable by concession in K, $(K \setminus J) \cup \{G\}$ is consistent for a minimal set $J \subseteq \Gamma$. As $(K \setminus J) \cup \{G\} = W \cup (\Gamma \setminus J) \cup \{G\}$, put $E = Th(W \cup (\Gamma \setminus J) \cup \{G\})$. If E is not a consistent extension of $\Delta_K \cup \{G\}$, there is a minimal deductively closed consistent set $E' = Th(W \cup \Gamma' \cup \{G\})$ such that $\Gamma' \subseteq \Gamma$. In case of $E' \subset E$, $\Gamma' \subset (\Gamma \setminus J)$. Put $\Xi = (\Gamma \setminus J) \setminus \Gamma'$. Then, $E = Th(W \cup \Gamma' \cup \Xi \cup \{G\})$ becomes inconsistent. This contradicts the assumption that E is consistent. In case of $E \subset E'$, $(\Gamma \setminus J) \subset \Gamma'$. Put $\Gamma' = \Gamma \setminus J'$ for some $J' \subset J$. Then, $(K \setminus J') \cup \{G\}$ becomes consistent. This contradicts the assumption that J is a minimal set which makes $(K \setminus J) \cup \{G\}$ consistent. Hence, E becomes a consistent extension of $\Delta_K \cup \{G\}$. Conversely, if E is an extension of $\Delta_K \cup \{G\}$, E is a minimal deductively closed set $E = Th(W \cup \Gamma'' \cup \{G\})$ where $\Gamma'' = \{\, \gamma \mid \frac{:\gamma}{\gamma} \in D$ and $\neg\gamma \notin E \,\}$. Then, there is a (minimal) set $J \subseteq \Gamma$ such that $\Gamma'' = \Gamma \setminus J$. For this J, it holds that $(K \setminus J) \cup \{G\}$ is consistent. Hence, G is acceptable by concession.

(ii) If $\Delta_K \cup \{G\}$ has a consistent extension, by putting $H = \{G\}$, $\Delta_K \cup H$ has a consistent extension E such that $G \in E$. Conversely, if for a set H of formulas $\Delta_K \cup H$ has a consistent extension E such that $G \in E$, E is also a consistent extension of $\Delta_K \cup H \cup \{G\}$ ([14, Theorem 2.6]). In this case, $\Delta_K \cup \{G\}$ has a consistent extension. (If $\Delta_K \cup \{G\}$ is inconsistent, $\Delta_K \cup H \cup \{G\}$ is inconsistent [14].) Thus, $\Delta_K \cup \{G\}$ has a consistent extension iff for a set H of formulas, $\Delta_K \cup H$ has a consistent extension E such that $G \in E$. Then, the result holds by Proposition 1. □

Theorem 2 represents that conditional acceptance by concession is characterized in terms of default inference of G from $\Delta_K \cup H$.

Example 3. (cont. Example 2) The knowledge base is represented by the default theory $\Delta_K = (D, W)$ where

$$
\begin{aligned}
W:\ & have(mirror) \wedge have(nail) \rightarrow hang(mirror),\\
& have(mirror) \wedge have(screw) \rightarrow hang(mirror),\\
& give(nail) \rightarrow \neg have(nail),\\
& have(screw) \rightarrow give(nail),\\
& \forall x\ get(x) \rightarrow have(x),\\
& have(mirror),\\
D:\ & \frac{:\, have(nail)}{have(nail)}.
\end{aligned}
$$

Given the request $G = give(nail)$, G is included in a default extension of $\Delta_K \cup \{\, get(screw) \,\}$.

3 Computing Proposals in Answer Set Programming

3.1 Answer Set Programming

Answer set programming (ASP) [9] represents incomplete knowledge in a logic program and realizes nonmonotonic default reasoning. In ASP a logic program is given by an *extended disjunctive program* (EDP). An EDP (or simply a *program*) is a set of rules of the form:

$$L_1 ; \cdots ; L_l \leftarrow L_{l+1}, \ldots, L_m, \mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n \tag{5}$$

($n \geq m \geq l \geq 0$) where each L_i is a positive/negative literal, i.e., A or $\neg A$ for an atom A, and *not* is *negation as failure* (NAF). $\mathit{not}\, L$ is called an *NAF-literal*. The above rule is read "some of $L_1, \ldots, L_l$ is believed if all $L_{l+1}, \ldots, L_m$ are believed and all $L_{m+1}, \ldots, L_n$ are disbelieved". The left-hand side of the arrow is the *head*, and the right-hand side is the *body*. For each rule r of the form (5), $\mathit{head}(r)$, $\mathit{body}^+(r)$ and $\mathit{body}^-(r)$ denote the sets of literals $\{L_1, \ldots, L_l\}$, $\{L_{l+1}, \ldots, L_m\}$, and $\{L_{m+1}, \ldots, L_n\}$, respectively. Also, $\mathit{not\_body}^-(r)$ denotes the set of NAF-literals $\{\mathit{not}\, L_{m+1}, \ldots, \mathit{not}\, L_n\}$. A rule r is often written as $\mathit{head}(r) \leftarrow \mathit{body}^+(r), \mathit{not\_body}^-(r)$ or $\mathit{head}(r) \leftarrow \mathit{body}(r)$ where $\mathit{body}(r) = \mathit{body}^+(r) \cup \mathit{not\_body}^-(r)$. A rule r is *disjunctive* if $\mathit{head}(r)$ contains more than one literal. A rule r is a *constraint* if $\mathit{head}(r) = \emptyset$; and r is a *fact* if $\mathit{body}(r) = \emptyset$. A program is *NAF-free* if no rule contains NAF-literals. A program, rule, or literal is *ground* if it contains no variable. A program P with variables is a shorthand of its *ground instantiation* $\mathit{Ground}(P)$, the set of ground rules obtained from P by substituting variables in P by elements of its Herbrand universe in every possible way.

The semantics of an EDP is defined by the *answer set semantics* [6]. Let Lit be the set of all ground literals in the language of a program. Suppose a program P and a set S of ground literals. Then, the *reduct* P^S is the program which contains the ground rule $\mathit{head}(r) \leftarrow \mathit{body}^+(r)$ iff there is a rule r in $\mathit{Ground}(P)$ such that $\mathit{body}^-(r) \cap S = \emptyset$. Given an NAF-free EDP P, let S be a set of ground literals which is (i) *closed* under P, i.e., for every ground rule r in $\mathit{Ground}(P)$, $\mathit{body}(r) \subseteq S$ implies $\mathit{head}(r) \cap S \neq \emptyset$; and (ii) *logically closed*, i.e., it is either consistent or equal to Lit. An *answer set* of an NAF-free program P is a minimal set S satisfying both (i) and (ii). Given an EDP P and a set S of ground literals, S is an *answer set* of P if S is an answer set of P^S. A program has none, one, or multiple answer sets in general. The set of all answer sets of P is written as $\mathit{AS}(P)$. An answer set is *consistent* if it is not Lit. A program P is *consistent* if it has a consistent answer set; otherwise, P is *inconsistent*.

A literal L is a consequence of *cautious inference* in a program P if L is included in every answer set of P. On the other hand, a literal L is a consequence of *brave inference* in a program P if L is included in some answer set of P.[2] We write $P \models_b L$ if a literal L is a consequence of brave inference in P.

[2] Cautious/brave inference is also called *skeptical*/*credulous* inference.

Example 4. The program:

$$\begin{aligned} &tea\,;\, coffee \leftarrow,\\ &milk \leftarrow tea,\, not\ lemon,\\ &lemon \leftarrow tea,\, not\ milk,\\ &milk \leftarrow coffee, \end{aligned}$$

has the three answer sets: $\{\, tea,\, milk \,\}$, $\{\, tea,\, lemon \,\}$, and $\{\, coffee,\, milk \,\}$, which represent possible options for drink.

3.2 Computing Proposals

As presented in Definition 6, weak beliefs are represented by super-normal default rules. In an EDP, a super-normal default rule

$$\frac{:\, L}{L}$$

with a literal L is represented as a rule

$$L \leftarrow not\ \neg L\,.$$

An EDP can represent other forms of default knowledge in terms of rules with NAF. By contrast, persistent belief is represented by rules without NAF. Thus, both weak and strong beliefs are uniformly represented in an EDP.

In what follows, we assume that a knowledge base is represented by an EDP and a proposal is given as a ground literal. Based on Theorem 2, we rephrase the notion of conditional acceptance by concession in the context of ASP as follows.

Definition 7. (conditional acceptance by concession) Let P be a program representing a knowledge base, and G a ground literal representing a proposal. Then, G is *acceptable by concession* in P iff $P \cup \{G\}$ is consistent. And G is *conditionally accepted by concession* in P iff $P \cup H \models_b G$ holds for some set H of rules such that $P \cup H$ is consistent. In this case, a set H of rules is called a *solution* for conditional acceptance by concession.

By Definition 7, given a program P and a ground literal G, our goal is to compute a set H of rules satisfying

$$P \cup H \models_b G \tag{6}$$

where $P \cup H$ is consistent. This is the problem of *brave induction* [19], induction based on brave inference. Note that in Section 2.2 conditional acceptance (2) is related to induction in first-order logic. By incorporating concession (3), the problem is characterized as brave inference from a default theory (Theorem 2). The relation (6) represents this default inference in ASP. When $P \models_b G$, $H = \emptyset$ becomes a solution of (6). This represents the situation that G is accepted by concession with no additional condition. In this case, G is accepted in the present knowledge base P and no hypothesis is required.

The problem of our interest is the case of $P \not\models_b G$. We next consider a method of constructing solutions in this case. Suppose a program P and a proposal G. Then, consider the set $\mathbf{R}$ of rules such that

$$\begin{aligned} r \in \mathbf{R} \quad \text{iff} \quad & \mathit{head}(r) = \{G\}\,,\ \mathit{body}(r) \neq \emptyset, \\ & \emptyset \subseteq \mathit{body}^+(r) \subseteq S\,, \text{ and} \\ & \mathit{body}^-(r) \subseteq \mathit{Lit} \setminus S \end{aligned}$$

where $S \in AS(P)$ and there is no $S' \in AS(P)$ such that $(S' \setminus T) \cup (T \setminus S') \subset (S \setminus T) \cup (T \setminus S)$ for an answer set T of $P \cup \{G\}$.

By Definition 7, $P \cup \{G\}$ has a consistent answer set if G is acceptable by concession. The condition $(S' \setminus T) \cup (T \setminus S') \subset (S \setminus T) \cup (T \setminus S)$ for no $S' \in AS(P)$ presents that an answer set S of P is selected such that S is closest to an answer set of $P \cup \{G\}$. The set $\mathbf{R}$ provides a hypothesis space for solutions.

Theorem 3. *Let P be a program and G a proposal such that $P \not\models_b G$. For any non-empty subset $H \subseteq \mathbf{R}$ such that $P \cup H$ is consistent, $P \cup H \models_b G$ holds.*

Proof. First, $(P \cup H)^S = P^S \cup H^S$ holds for any answer set S of P. For any non-empty subset $H \subseteq \mathbf{R}$, $\mathit{body}^-(r) \cap S = \emptyset$ for any $r \in H$. Then, there is a rule $G \leftarrow \mathit{body}^+(r)$ in H^S such that $\emptyset \subseteq \mathit{body}^+(r) \subseteq S$. Since $P \cup H$ is consistent, a consistent set $S \cup \{G\}$ becomes a minimal closed set of $P^S \cup H^S$. Then, $S \cup \{G\}$ becomes a consistent answer set of $P \cup H$. Hence, the result holds. □

By Theorem 3, a possible solution H of (6) is selected from the set $\mathbf{R}$ of a hypothesis space. Generally speaking, however, there are many candidates of hypotheses satisfying the condition. We remark some methods of eliminating hypotheses which are useless in the process of negotiation. First, we eliminate hypotheses by syntactic restriction. Suppose that two rules r_1 and r_2 satisfy the relation $P \cup \{r_1\} \models_b G$ and $P \cup \{r_2\} \models_b G$. In this case, if $\mathit{body}(r_1) \subset \mathit{body}(r_2)$ holds, r_2 is eliminated. This is because r_1 provides a condition simpler than r_2 for accepting G. A rule r in $\mathbf{R}$ is said *minimal* if there is no rule r' in $\mathbf{R}$ such that $\emptyset \subset \mathit{body}(r') \subset \mathit{body}(r)$. We can eliminate every non-minimal rule from $\mathbf{R}$. Moreover, for any $H_1 \subseteq \mathbf{R}$ and $H_2 \subseteq \mathbf{R}$, if both H_1 and H_2 satisfy the condition of Theorem 3 and the relation $H_1 \subseteq H_2$ holds, the minimal set H_1 of hypotheses is preferred to non-minimal H_2. Note that we are interested in any rule $r \in \mathbf{R}$ such that $\mathit{body}(r) \neq \emptyset$. If $\mathit{body}(r) = \emptyset$, r becomes a fact $G \leftarrow$ and $P \cup \{G \leftarrow\} \models_b G$ immediately holds. This corresponds to "Consent" in Section 2.2 and is a trivial solution. So in what follows we consider rules in $\mathbf{R}$ having non-empty bodies.

Second, we eliminate hypotheses by semantic restriction. Agents in negotiation are involved in matters of mutual interests. For example, in negotiation between a buyer and a seller, they are interested in buying or selling some designated product. In this case, hypotheses are related to conditions for buying-selling the objective product. They include price, service, quality of products, and so on. Some belief with respect to one's taste for foods is irrelevant to the buying-selling activities (except for the case of buying-selling foods). Let C be a set of literals and NAF-literals specifying specific *context* of negotiation. Then, any rule r in

$\mathbf{R}$ is eliminated if $body(r) \not\subseteq C$. In other words, we are interested in getting any rule having conditions in the specific context. We assume that negotiating agents have their contexts in advance. This assumption appears natural because no agent negotiates without any interest. Different agents could have different contexts depending on their interests. These syntactic and semantic restrictions help to reduce candidate hypotheses in $\mathbf{R}$. Such restrictions on a hypothesis space are called *induction bias* [12]. After applying these restrictions to $\mathbf{R}$, there could still exist multiple choices of proposals. In this case, we assume that an agent nondeterministically selects a proposal from $\mathbf{R}$.

Given a proposal G by an agent Ag_1, another agent Ag_2 evaluates G in its knowledge base P. If $P \models_b G$, it is accepted with concession and an agreement is reached. Else if $P \not\models_b G$, Ag_2 computes a hypothesis H to satisfy $P \cup H \models_b G$ and builds a counter-proposal based on H. If no such H exists, the negotiation ends in a disagreement. By iterating these steps, a negotiation proceeds until it reaches an agreement/disagreement.

Example 5. Suppose a buying-selling situation in Section 1. A seller agent has the knowledge base P_s which consists of the following rules:

$$product(hdd, 200G, 120\$) \leftarrow not\ \neg\, product(hdd, 200G, 120\$), \tag{7}$$
$$product(hdd, 180G, 100\$) \leftarrow not\ \neg\, product(hdd, 180G, 100\$), \tag{8}$$
$$\leftarrow product(x, y, z),\ product(x, y, w),\ z \neq w, \tag{9}$$
$$\neg\, product(hdd, 200G, 120\$) \leftarrow product(hdd, 200G, x), x < 120\$,\ pay\_cash, \tag{10}$$
$$pay\_cash\,;\ pay\_card\ \leftarrow . \tag{11}$$

Here, the rules (7) and (8) represent products and their normal prices. They are represented by super-normal defaults because prices are subject to change. The rule (9) represents a constraint that the same product cannot have different prices at the same time. The rule (10) represents if discount is made by payment with cash, the normal price is withdrawn. The rule (11) represents two options for payment. P_s has two answer sets:

$$S_1 :\ \{\, product(hdd, 200G, 120\$),\ product(hdd, 180G, 100\$),\ pay\_cash\,\},$$
$$S_2 :\ \{\, product(hdd, 200G, 120\$),\ product(hdd, 180G, 100\$),\ pay\_card\,\},$$

which represent the seller's initial belief.

A buyer agent has the knowledge base P_b which consists of rules:

$$product(hdd, 200G, 100\$) \leftarrow not\ \neg\, product(hdd, 200G, 100\$), \tag{12}$$
$$\leftarrow product(hdd, x, y),\ y > 100\$, \tag{13}$$
$$\leftarrow pay\_cash, \tag{14}$$
$$product(hdd, 180G, 100\$) \leftarrow not\ product(hdd, 200G, 100\$), \tag{15}$$
$$\neg\, product(hdd, 200G, 100\$) \leftarrow product(hdd, 180G, 100\$). \tag{16}$$

The rule (12) represents the intention of getting a HDD of 200GB with 100USD. It is represented by a super-normal default because the intention could be

changed. The rule (13) represents a constraint that the budget is less than 100USD. The rule (14) represents a constraint that she does not pay by cash. The rule (15) represents that if a HDD with 200G is unavailable under the budget, the buyer would have an option of downgrading the specification. The rule (16) represents that the original request is withdrawn if the specification changes. P_b has two answer sets:

$$\{\, product(hdd, 200G, 100\$)\,\},$$
$$\{\, \neg\, product(hdd, 200G, 100\$),\ product(hdd, 180G, 100\$)\,\},$$

which represent the buyer's initial belief. With this setting, negotiation starts.

(1st round) First, the buyer asks the price of a HDD with 200GB. As $P_s \models_b product(hdd, 200G, 120\$)$, the seller replies the price:

$$G_s^1 :\ product(hdd, 200G, 120\$).$$

(2nd round) The buyer does not accept G_s^1 because it violates the constraint (13) and $P_b \cup \{G_s^1\}$ is inconsistent. As $P_b \models_b product(hdd, 200G, 100\$)$, the buyer returns the proposal

$$G_b^2 :\ product(hdd, 200G, 100\$)$$

to the seller. The seller evaluates G_b^2 in its knowledge base P_s and knows that $P_s \cup \{\, product(hdd, 200G, 100\$)\,\}$ is consistent. Hence, G_b^2 is acceptable by concession in P_s. The seller then seeks a hypothesis H to accept G_b^2 and constructs the set $\mathbf{R}$ satisfying the condition:

$$\begin{aligned} r \in \mathbf{R} \quad \text{iff} \quad & head(r) = \{G_b^2\},\ body(r) \neq \emptyset, \\ & \emptyset \subseteq body^+(r) \subseteq S_1\,, \text{ and} \\ & body^-(r) \subseteq Lit \setminus S_1 \end{aligned}$$

where S_1 is selected because S_1 is closer to the answer set $\{\, product(hdd, 200G, 100\$), \neg\, product(hdd, 200G, 120\$), product(hdd, 180G, 100\$), pay\_cash\,\}$ of $P_s \cup \{\, G_b^2\,\}$ than S_2 is. The set $\mathbf{R}$ includes rules such that

$$\begin{aligned} r_1 :\ & product(hdd, 200G, 100\$) \leftarrow product(hdd, 200G, 120\$), \\ r_2 :\ & product(hdd, 200G, 100\$) \leftarrow product(hdd, 180G, 100\$), \\ r_3 :\ & product(hdd, 200G, 100\$) \leftarrow pay\_cash, \\ r_4 :\ & product(hdd, 200G, 100\$) \leftarrow not\ pay\_card, \\ r_5 :\ & product(hdd, 200G, 100\$) \leftarrow product(hdd, 200G, 120\$), \\ & \qquad\qquad\qquad\qquad\quad product(hdd, 180G, 100\$), \\ r_6 :\ & product(hdd, 200G, 100\$) \leftarrow product(hdd, 200G, 120\$),\ pay\_cash, \\ & \cdots\cdots\cdots \end{aligned}$$

Among them, non-minimal rules, for instance, r_5 and r_6, are eliminated. The seller also considers the context C as

$$C : \{\, product(hdd, 200G, 120\$),\ pay\_cash,\ pay\_card\,\},$$

because the present deal between the buyer and the seller is on the product $product(hdd, 200G, 120\$)$ and its payment. Then, rules containing any literal $L \notin C$, for instance r_2, are eliminated. After the elimination, the seller selects rules such that $P_s \cup \{r_i\}$ is consistent. Since r_1 violates the constraint (9), two rules r_3 and r_4 remain as candidates. Then, the seller constructs possible hypotheses as $H_1 = \{r_3\}$ and $H_2 = \{r_4\}$. Both $P_s \cup H_1$ and $P_s \cup H_2$ have two answer sets:

$$\{\, product(hdd, 200G, 100\$),\ \neg\, product(hdd, 200G, 120\$),$$
$$product(hdd, 180G, 100\$),\ pay\_cash\,\},$$
$$\{\, product(hdd, 200G, 120\$),\ product(hdd, 180G, 100\$),\ pay\_card\,\},$$

so that $P_s \cup H_i \models_b product(hdd, 200G, 100\$)$ for $i = 1, 2$. r_3 provides a condition to hold $product(hdd, 200G, 100\$)$, and $P_s \cup H_1 \models_b pay\_cash$, the seller returns the condition

$$G_s^2 :\ pay\_cash$$

as a counter-proposal.

(3rd round) The buyer does not accept G_s^2 because it violates the constraint (14). The buyer then returns the critique

$$G_b^3 :\ reject$$

to the seller. As there is no way to meet the request of the buyer, the seller makes a new proposal

$$G_s^3 :\ product(hdd, 180G, 100\$)$$

which satisfies $P_s \models_b G_s^3$.

(4th round) As $P_b \models_b G_s^3$, the buyer accepts the proposal and an agreement is reached.

3.3 Computational Complexity

We finally address computational complexity results. Throughout the section, any knowledge base P is assumed to be a ground program and H is a set of ground rules. A proposal G is a ground literal as before.

Theorem 4. *Let P be a program and G a proposal. Then, the following complexity results hold.*

1. *Deciding whether G is acceptable by concession in P is Σ_2^P-complete.*
2. *Deciding whether H is a solution for conditional acceptance by concession is Σ_2^P-complete.*

Proof. By Definition 7, G is acceptable by concession in P iff $P \cup \{G\}$ is consistent. Deciding the existence of a consistent answer set is Σ_2^P-complete [3]. Hence, the result (1) holds. Next, by Definition 7, G is conditionally accepted by concession in P iff $P \cup H \models_b G$ holds for a set H of rules such that $P \cup H$ is consistent. Deciding the existence of a consistent answer set in which G is true is Σ_2^P-complete [3]. Hence, the result (2) holds. □

By Theorem 4, the task of building possible solutions is generally expensive. The decision problems become NP-complete when P and H contain no disjunction. One reason for the complexity of computing solutions is *nonmonotonicity* of default reasoning that arises in making concession. If no concession is made, proposals are built by induction only. In this limited problem setting, some efficient induction algorithms that are developed in the field of machine learning and ILP could be used. Detailed techniques of using efficient induction algorithms are outside the scope of this paper.

4 Related Work

Several studies use logic-based *abduction* or *abductive logic programming* [8] as a representation language of negotiating agents [10,15,17]. Kakas and Moraitis [10] propose a negotiation protocol which integrates abduction within an argumentation framework. In their negotiation protocol, an agent considers another agent's goal and searches for conditions under which the goal is acceptable. They use abduction to seek conditions to support arguments. In their protocol, counter-proposals are chosen among candidates based on preference knowledge of an agent. In [15] an abductive logic program is used for specifying dialogue primitives and negotiation protocol. Once a dialogue is uttered by an agent, another agent that observed the utterance thinks and acts according to a given *observe-think-act cycle*. There are two important differences between [10,15] and our present work. First, those studies have no mechanism of constructing new counter-proposals in response to a proposal made by an agent. The behavior of an agent is completely pre-specified in either a knowledge base of an agent or a negotiation protocol, and possible responses are prepared in advance. In our framework, proposals are newly constructed using induction. It enables us to build proposals that are independent of particular negotiation protocols. Most theories of automated negotiation specify possible responses in advance, while [17] is an exception. Sakama and Inoue [17] propose methods for building new proposals by *extended abduction* and *relaxation*. Extended abduction is an extension of abduction proposed by Inoue and Sakama [7], which can not only introduce hypotheses to a knowledge base but remove hypotheses from it to explain an observation. Relaxation is a technique of weakening constraints in database queries. They use extended abduction to compute *conditional proposals* and use relaxation to compute *neighborhood proposals*. An essential difference from our present work is that they use (extended) abduction for computing conditions of accepting a proposal, while we use induction for that purpose.

Formally, extended abduction is defined as follows. An *abductive framework* is a pair $\langle K, \Gamma \rangle$ in which both K and Γ are first-order theories.[3] Given an observation G as a formula, a pair (E, F) is an *explanation* of G (with respect to $\langle K, \Gamma \rangle$) if (1) $(K \setminus F) \cup E \models G$, (2) $(K \setminus F) \cup E$ is consistent, and (3) both E and F consist of instances of elements from Γ. They also introduce the

[3] The paper [7] introduces the framework in the context of autoepistemic logic, and another paper [16] uses it in the context of abductive logic programming.

notion of *anti-explanations* to *un*explain negative observations. The above definition appears similar to the notion of "conditional acceptance by concession" which is defined as the relation $(K \setminus J) \cup H \models G$ in Definition 5 of this paper. However, there is an important difference between two definitions. In extended abduction, a hypothesis space Γ is given in advance. An explanation (E, F) is selected from the direct product $\Gamma \times \Gamma$. In our Definition 5, a set J is selected as a subset of weak belief Γ in a knowledge base K, while a hypothesis H is *newly* built by a knowledge base K and an observation G. This difference comes from the inherent characteristics of abduction and induction [4]. In (extended) abduction, the goal is to compute causes of some observed events using a background knowledge base. In this case, possible causes are extracted from information in the knowledge base. In induction, on the other hand, the goal is to discover unknown general rules that would lie between observed events and the current belief in a knowledge base. We make use of this style of inference in the context of negotiation. A proposal given by another agent is not always explained using information included in a knowledge base only. In this case, an agent tries to bridge the gap between the proposal and its current belief. The method in [17] extracts conditions to satisfy a given proposal, which is useful for making the "Subsumption" style of proposals in Section 2.2. By contrast, our method proposed in this paper is useful for making the "Implication" style of proposals, which is more general than the subsumption style. To the best of our knowledge, no study characterizes the process of making proposals in terms of induction. On the other hand, [17] proposes another method for building proposals based on relaxation, which is different from both abduction and induction.

Meyer et al. [11] introduce a logical framework for negotiating agents. They introduce two different modes of negotiation: *concession* and *adaptation*. Concession weakens an initial demand of an agent, while adaptation expands an initial demand to accommodate a demand of another agent. They provide rational postulates to characterize negotiated outcomes between two agents, and describe methods for constructing outcomes. Compared with our present work, they consider classical propositional theories and provide logical conditions for negotiated outcomes to satisfy, but they do not provide a method of constructing proposals in negotiation. Amgoud et al. [1] develop a formal theory of argumentation-based negotiation. It provides a protocol that allows agents to exchange offers and arguments, and to make concessions when necessary. In their framework, offers that can be exchanged during a negotiation dialogue are given in advance, and concessions are made by proposing/accepting less preferred offers. Preference between offers is defined by the existence of arguments supporting them. So it does not construct new offers nor modify the current belief in a knowledge base. In the context of answer set programming, Foo et al. [2,5] formalize one-to-one negotiation between two extended logic programs. In their framework, two agents exchange answer sets to produce a common belief set. Their goal is coordinating belief sets of two agents, and it has no mechanism of constructing new proposals.

5 Conclusion

This paper introduced a logical framework of negotiating agents. An agent evaluates a proposal and constructs a counter-proposal by building hypotheses using induction, and concedes by abandoning a part of weak belief of its knowledge base. Such behavior of an agent is formalized using default logic and realized in answer set programming. The result of this paper shows that default reasoning and induction as well as abduction, which are all developed as commonsense reasoning for a single agent in AI, are also useful for social reasoning in multi-agent systems. They provide formal methodologies for reasoning about agents, and are used as general inference mechanisms which are independent of particular negotiation protocols.

Several issues remain for further work. We used default logic to distinguish strong and weak beliefs. In realistic negotiation, however, there would be different degrees of preferences over beliefs, and simple distinction between strong and weak beliefs might be insufficient. For further refinement, a logic for prioritized reasoning is needed, together with the capability of revising the preferences attached to beliefs. We developed a method for computing proposals in answer set programming, which enables us to realize automated negotiation on top of the existing answer set solvers. Prototyping an implementation and evaluating the proposed framework on practical negotiating tasks are in the next step.

References

1. Amgoud, L., Dimopoulos, Y., Moraitis, P.: A unified and general framework for argumentation-based negotiation. In: Proc. 6th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1018–1025. ACM Press, New York (2007)
2. Chen, W., Zhang, M., Foo, N.: Repeated negotiation of logic programs. In: Proc. 7th Workshop on Nonmonotonic Reasoning, Action and Change (2006)
3. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: propositional case. Annals of Mathematics and Artificial Intelligence 15, 289–323 (1995)
4. Flach, P.A., Kakas, A.C. (eds.): Abduction and Induction — Essays on their Relation and Integration. Kluwer Academic Publishers, Dordrecht (2000)
5. Foo, N., Meyer, T., Zhang, Y., Zhang, D.: Negotiating logic programs. In: Proc. 6th Workshop on Nonmonotonic Reasoning, Action and Change (2005)
6. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9, 365–385 (1991)
7. Inoue, K., Sakama, C.: Abductive framework for nonmonotonic theory change. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence, pp. 204–210. Morgan Kaufmann, San Francisco (1995)
8. Kakas, A.C., Kowalski, R.A., Toni, F.: The role of abduction in logic programming. In: Gabbay, D.M., et al. (eds.) Handbook of Logic in AI and Logic Programming, vol. 5, pp. 235–324. Oxford University Press, Oxford (1998)
9. Lifschitz, V.: Answer set programming and plan generation. Artificial Intelligence 138, 39–54 (2002)

10. Kakas, A.C., Moraitis, P.: Adaptive agent negotiation via argumentation. In: Proc. 5th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 384–391. ACM Press, New York (2006)
11. Meyer, T., Foo, N., Kwok, R., Zhang, D.: Logical foundation of negotiation: outcome, concession and adaptation. In: Proceedings of the 19th National Conference on Artificial Intelligence, pp. 293–298. MIT Press, Cambridge (2004)
12. Nienhuys-Cheng, S.-H., De Wolf, R.: Foundations of Inductive Logic Programming. LNCS, vol. 1228. Springer, Heidelberg (1997)
13. Parsons, S., Sierra, C., Jennings, N.: Agents that reason and negotiate by arguing. Journal of Logic and Computation 8(3), 261–292 (1988)
14. Reiter, R.: A logic for default reasoning. Artificial Intelligence 13, 81–132 (1980)
15. Sadri, F., Toni, F., Torroni, P.: An abductive logic programming architecture for negotiating agents. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS, vol. 2424, pp. 419–431. Springer, Heidelberg (2002)
16. Sakama, C., Inoue, K.: An abductive framework for computing knowledge base updates. Theory and Practice of Logic Programming 3(6), 671–715 (2003)
17. Sakama, C., Inoue, K.: Negotiation by abduction and relaxation. In: Proc. 6th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1018–1025. ACM Press, New York (2007)
18. Sakama, C.: Negotiation by induction (short paper). In: Proc. 7th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1459–1462. ACM Press, New York (2008)
19. Sakama, C., Inoue, K.: Brave induction. In: Železný, F., Lavrač, N. (eds.) ILP 2008. LNCS, vol. 5194, pp. 261–278. Springer, Heidelberg (2008)

Mental State Abduction of BDI-Based Agents*

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. In this paper we present mental state abduction, a technique for inferring the mental states (beliefs, goals) of BDI-based agents from observations of their actions. Abduced mental states are considered to be explanations of observed behavior, which is assumed to be part of a (goal-directed) plan. The observer, who attempts to explain an agent's behavior, is assumed to know all of the agent's behavioral rules that can account for observed actions. Three explanatory strategies are introduced, based on different perceptory conditions.

1 Introduction

Intelligent computational agents implemented in BDI-inspired programming languages are suited for developing virtual (non-player) characters for computer games and simulations [1]. In this work we tackle the problem of providing explanations for the observed behavior of virtual characters implemented as BDI-based agents in terms of their mental states.

Agents in BDI-inspired programming languages such as 2APL [2,3], Jack [4], Jadex [5] and Jason [6], are programmed in terms of mentalistic notions like beliefs, goals, and plans. In order for agents to deliberately cooperate with or obstruct the plans of other agents, it is a prerequisite that they can draw (defeasible) conclusions about those other agents' mental states, a capacity we refer to as mental state inference. Because agent languages with declarative mental states are often logic-based, and because a logical relation mental state $\Rightarrow$ behavior can (approximately) be identified, logical abduction — a way for inferring explanations for an observed fact — is a promising approach to providing explanations of agents' observed behavior in terms of mental state descriptions.

We envision the use of techniques for mental state inference as a means of designing characters that show a higher degree of believability [7]. Agents that incorporate beliefs about the mental state of other agents into their plans can be expected to show more *socially aware* behavior. When explaining the behavior of others, they either correctly infer their goals or beliefs and act accordingly, or

* 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 2008, LNAI 5397, pp. 161–178, 2009.

they arrive at incorrect conclusions based on explanations that are nonetheless plausible, which gives rise to a form of erroneous behavior the user understands and tolerates. Based on their hypotheses, agents can also form expectations with respect to other agents' future actions. Note that to actually *use* beliefs about the mental state of others, agents would need mental models of the mental states of other agents. Abduced explanations would have to be incorporated into these mental models, requiring a revision mechanism and preservation of integrity constraints. This is something we do not concern us with in this work; instead we separate the agent from the observer, and focus solely on ways to find plausible explanations, without worrying how these explanations will be used.

The structure of this paper is as follows. In Section 2 we sketch the outline of our approach, and relate it to existing similar work. Basic notions are introduced in Section 3, and put to use in the next Section 4 to define the technique of mental state abduction. In Section 5 this technique is illustrated through a simple example, and we conclude by discussing our results and pointing out possible future work in the last Section 6.

2 Our Approach and Related Work

In this paper, we investigate an approach to inferring the mental state of BDI agents based on their observed behavior, and take the agent programming language 2APL [2,3] as a departure point for our investigations. We have tried to make the framework as general as possible, though, in the hope that it will apply to any agent programming language in which declarative mental states drive the operation of the agent. It should be noted that our intention is not to capture the full dynamics of current agent programming languages, and therefore we present a simplified general language for programming BDI-based agents.

One assumption we make is that the actions of agents are observable as atomic, unambiguous facts. Most of the accepted definitions of the concept 'agent' presume operation in some dynamic, uncertain environment. If this environment is the real world, then the assumption that actions can be unambiguously observed as discrete facts is disputable, to say the least. However, since we consider virtual characters that operate in a game or simulation setting, the environment and therefore also the information that virtual characters (agents) receive are under our direct control. This means that we have the freedom to choose how to represent actions, and also the perception of actions. In the present case, we think of agents as performing atomic external actions, the execution of which is presented to observers in the form of perceptional events.

Not all actions are necessarily perceived. The environment might determine that the observer will not receive a perceptory event because the action occurs outside its perception radius, or perception might simply fail with some predetermined or context-dependent probability. For our case, we assume that *if* an action is perceived, this occurs as an event that is an unambiguous representation of this action, which still leaves open the possibility that the action isn't perceived at all.

BDI agents operate by means of goal-directed behavioral rules, which are selected and executed by way of some deliberation process. A strong restriction we make is that the observing entity, which perceives an agent's actions and attempts to infer its mental state, has knowledge of the entire set of operational rules of this agent. This assumption might seem unrealistic, but for the sake of simplicity and clarity of presentation we have adopted it. In future work, we hope to relax this requirement, which would mean that some observations might not be explainable at all because no known rule can account for them, or that erroneous conclusions are reached because some known rule was used to abduce an explanation in a case where an unknown rule actually produced the action. We believe, though, that it will become clear from this paper that things are already complicated enough even with the restriction enforced.

The classical logical approach to explaining observations is abduction [8], which is perhaps best described as reasoning from observation to explanation. In our approach to the problem of explaining the actions of agents in terms of their mental states we have drawn inspiration from work on abduction in propositional logic. For this approach to be successful, we have made the assumption that it is possible to describe the procedural rules of agents in the form of logical implications. It is a debatable assumption, but one we think we can successfully defend, as we will attempt to do in Section 3.2. Inferring the mental states of agents based on their observed behavior is what we call *mental state abduction*. Before going into the details of this technique, we will explain how the concept of abduction is generally understood.

2.1 Abduction

Logicians agree about the existence of several distinct modes of reasoning, of which *deduction*, *induction*, and *abduction* [9] are complementary. Deduction is the classical syllogism of modus ponens $\{\phi, \phi \rightarrow \psi\} \models \psi$, where ψ can be validly inferred if both ϕ and $\phi \rightarrow \psi$ are true. Induction can be generalized to the inference of $\phi \rightarrow \psi$ as a rule from the observation of ϕ followed by ψ, and abduction to the inference of ϕ from knowledge of the rule $\phi \rightarrow \psi$ and the observation ψ. Deduction is the only analytic form of inference, whereas induction and abduction both are defeasible, meaning that new observations can invalidate prior conclusions. With indefeasible inference we cannot use the classical entailment relation between premise and conclusion, so instead we have to use some non-monotonic relation, represented by $\rightsquigarrow$. The inductive and abductive syllogisms are $\{\phi, \psi\} \rightsquigarrow (\phi \rightarrow \psi)$ and $\{\phi \rightarrow \psi, \psi\} \rightsquigarrow \phi$, respectively.

The process of abduction is best described as *reasoning from observation to explanation*. The fact that an observed phenomenon ψ (called the *explanandum*) requires explanation, is taken to mean that this phenomenon does not follow directly from a body of knowledge, the background theory Θ. In logical terms, $\Theta \not\models \psi$. Moreover, for some fact ϕ to count as an explanation, a restriction is often enforced that the so-called *explanans* ϕ does not account for the explanandum on its own, so $\phi \not\models \psi$. There are also issues of consistency; the explanans ϕ should not be contradicted by the theory, so $\Theta \cup \phi \not\models \bot$ (or some other criterion

for consistency, allowing for revision of the background theory). The schema for abductive reasoning, where ϕ explains ψ with respect to the background theory Θ, can then be defined by the following conditions [10]:

Definition 1 (Abductive Explanation). *The conditions which define when a fact ϕ (the explanans) qualifies as a valid abductive explanation for an observed fact ψ (the explanandum).*

$$\Theta \cup \phi \models \psi$$
$$\Theta \cup \phi \not\models \bot$$
$$\Theta \not\models \psi \quad \textit{and} \quad \phi \not\models \psi$$

Abductive inference, where observing ψ leads to inference of ϕ as an explanation w.r.t. background theory Θ, is defined by these conditions.

$$\Theta, \psi \rightsquigarrow \phi \quad \text{iff the conditions in Definition 1 apply.}$$

In logic programming, the restriction that explanations should belong to a special set of so-called *abducibles* is often enforced, to prevent explaining observations in terms of other effects instead of causes, and for reasons of computational efficiency [11,12]. We will use this restriction in our approach as well, to limit the space of candidate explanations.

It should be noted that the notion of abduction is a murky one, and there is no general agreement on its definition [12,13,14]. Some logicians only discern between deduction and induction, and see what we have called abduction as belonging to the latter. What is important for this account, though, is not so much complete knowledge of these inferential concepts, but a precise definition of these concepts *as we understand and use them.* In Section 4 we define mental state abduction, and link it to classical abduction as presented here.

2.2 Related Work

This work is largely related to work in the area of plan and intention recognition; see [15] for an overview of this field. In plan recognition, a common dichotomy is that of opposing *keyhole recognition* to *intended recognition.* The former deals with the recognition of an agent's plan through unobtrusive observation, implying that the agent is unaware of the fact that it's being observed. In the latter case the agent knows that it's being watched, which allows for the agent to actively thwart the recognition of its intentions by performing misleading actions.

Keyhole recognition best describes our approach, as we don't deal with deception on part of the agent in the form of it deliberately performing misleading actions. However, we would be able to deal with an agent that tries to hide its actions from sight. The robustness of a plan recognition technique is described in [15] as its ability to deal with incomplete observations. Incomplete perception is what would occur if an agent were actively trying to prevent the observer from perceiving its actions, and this is something we deal with explicitly by proposing several explanatory strategies in Section 4.2.

An approach which is related to ours is that of Albrecht et al. [16], who use a Bayesian network to identify users' plans and goals in an adventure game setting. For this to work, extensive data of users' actions and target quests is processed, and probability distributions are extracted in the form of dynamic belief networks. This kind of approach can work for applications where such data is available, but often this is not the case, simply because logs of users' actions and motivations are not recorded. For inter-agent plan recognition it is easier to use the implementation of the agent as a starting point, which is possible using our approach. Furthermore, if the behavior of the user is to be explained, a description of his or her behavior can be naturally made in BDI terms, which is possible to do without having prior knowledge of the way users solve quests, but instead by describing natural goals in the scenario and ways to achieve them.

In [17], Appelt and Pollack present an approach to abductive reasoning called weighted abduction, which they use to ascribe mental states to an agent based on observed actions. Their approach makes use of inference weights to compare competing explanations. The set of assumptions that leads to the lowest cost proof is selected as explanation. Apart from the obvious similarity in the use of abduction to infer mental states, their approach and ours differ significantly. Our starting point is the inference of mental states of characters in games or simulations, which are implemented in some variety of BDI agent programming language. Appelt and Pollack's work is not grounded in agent programming, and they take a more conceptual approach. Moreover, using inference weights to compare explanations does not readily apply in our case, as such weights would depend on the procedural mechanism underpinning agents' reasoning, and the context in which observed actions took place. Nevertheless, weighted abduction is an elegant way to choose among competing explanations, and the idea of using (dynamic) weights on explanations is something we consider for future work.

One other approach that is grounded in an agent programming methodology is that of Goultiaeva and Lespérance [18], which is based on the situation calculus, and ConGolog specifically. It details a technique for recognizing the plan an agent is performing, and corresponds to our approach insofar that inference is based on observed actions and a library of known plans. It also supports incremental recognition: each new perception narrows the space of possible explanations. However, it is limited in the respect that it requires all actions to be observed, and is therefore not robust in the terms of [15] mentioned earlier. Moreover, it stops with *recognizing* plans, whereas we provide explanations (in terms of mental states) that represent the *reasons why* an agent performed its actions. This allows for more flexibility, because the observing party can actively intervene, either cooperatively or obtrusively.

3 Basic Notions and Terminology

In this section, we present the syntax of a simplified logic-based agent programming language, with programming constructs for implementing rational BDI agents. Also, we present the syntax of a language for programming an abstract

observer entity, which serves as a proof of concept for the inferential mechanism we introduce. This separation of observer and agent serves to clarify, for ourselves as well as the reader, the interaction between those two distinct roles. Ultimately, though, we envision the possibility of integrating the abductive faculties of the observer into the general reasoning mechanism of agents.

3.1 Plans and Behavioral Rules

Let $\mathcal{L}$, with typical element ϕ, be a propositional language with negation $\neg$ and the standard conjunctional connective $\wedge$. Belief formulas β and goal formulas κ are elements from $\mathcal{L}$. Also, assume a set $\mathcal{A}$ of basic actions, with typical element α. Agents' plans are then represented as a sequence of basic actions α, tests on propositional formulas, and an operator for non-deterministic choice.

Definition 2 (Plans). *Plan expressions $\mathcal{L}_\Pi$ with typical element π are made up of elementary actions $\alpha \in \mathcal{A}$, tests on propositional formulas ϕ, non-deterministic choice between plans, or a sequential composition of plan elements.*

$$\pi ::= \alpha \mid \phi? \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2$$

In this work we are mainly concerned with the syntactical form of those plans, and do not define the semantics of the agent's operation. We do, however, make some assumptions about how an agent executes plans. Only external actions can be observed, and as we define all basic actions $\alpha \in \mathcal{A}$ to be external, we therefore deem them observable. Internal test actions $\phi?$ on propositional formulas of type $\phi \in \mathcal{L}$ are not observable. The choice operator $+$ is a non-deterministic choice operator, which has scope over two elements $\pi \in \mathcal{L}_\Pi$. It can also be used to represent deterministic instructions of the form `if` ϕ `then` π_1 `else` π_2 as $(\phi?; \pi_1) + (\neg\phi?; \pi_2)$. For completeness we introduce an empty or `null` action ϵ, such that $(\pi; \epsilon) \equiv (\epsilon; \pi) \equiv \pi$.

Agents' behavioral (planning) rules, of the form $\kappa \leftarrow \beta \mid \pi$, state that a plan π is appropriate for achieving a specific goal κ, given that some condition β holds. The selection of these rules is done by the agent's deliberation process.

Definition 3 (Rules). *Rules $\mathcal{L}_\mathrm{R}$ with typical element r are defined as follows, where $\beta, \kappa \in \mathcal{L}$, and $\pi \in \mathcal{L}_\Pi$.*

$$\mathsf{r} ::= \kappa \leftarrow \beta \mid \pi$$

We do not explicitly define the semantics of the agent's deliberation process, but instead adopt some assumptions about how this process operates. We refer to [2,3] for a more precise treatment of those assumptions. We assume that an agent selects a plan only if it is not actively dealing with some other plan. This means that at no point an agent is executing more than one plan. Furthermore, we assume that an agent only executes a plan to achieve some goal it explicitly has. Also, plans may be discarded, either because the goal for which they were selected has been achieved, or has been dropped.

The reasons for postulating these assumptions is to ensure some properties we require for the technique of mental state abduction to work. They make sure that an agent always performs its actions to achieve the single goal it is pursuing at that moment; in other words, an agent does not execute plans concurrently. The requirement of the agent only performing actions to achieve some goal — therefore not responding to external events — is to ensure that there is always *some* explanation for an action, since every action is produced by some rule from $\mathcal{L}_{\mathrm{R}}$ which has a mental state as precondition. We explicitly do allow for an agent discarding its plan for whatever reason, as we see the ability to deal with such a circumstance as a requirement for a plan recognition approach to be considered robust.

3.2 Observables, Abducibles, and Rule Descriptions

The observer is defined as an abstract entity — insofar that it does not act upon the environment — which perceives the actions of the agent, and attempts to come up with a plausible explanation. The observer's perception is a sequence of actions as performed by the agent. Perception might be complete in the sense that all the actions which the agent has performed are perceived, or some actions might not be perceived and thus perception might be incomplete.

Definition 4 (Percepts). *With α as the typical element of a set of basic actions $\mathcal{A}$, the perception language $\mathcal{L}_\Delta$ with typical element δ defines the perception of the observer.*

$$\delta ::= \alpha \mid \delta_1; \delta_2$$

As mentioned in Section 2.1, in computational abduction candidate explanations are constrained to a set of explicitly defined abducibles. In the present context, because we want to abduce the mental states that underlie an agent's actions, these abducibles are the agent's beliefs and goals. Abducible belief and goal formulas are predicated by belief and goal, respectively. Because conjunction between propositions is allowed in belief as well as goal formulas, this distinction is needed to discern between beliefs and goals in the process of abduction.

Definition 5 (Abducibles). *If $\beta, \kappa \in \mathcal{L}$, let λ be the typical element of the formulae of abducibles $\mathcal{L}_\Lambda$, and be defined as follows.*

$$\lambda ::= \mathsf{belief}(\beta) \mid \mathsf{goal}(\kappa) \mid \mathsf{belief}(\beta) \wedge \mathsf{goal}(\kappa)$$

To reason about an agent's mental state based on its observed behavior, the observer needs a description of the agent's rules of operation. Since the rules as such have meaning only in their original agent programming context, and we wish to use them in logic, a translation is needed. We present the translated rules as implications which state that a plan is implied by its (mental) preconditions. Note that preconditions are abducibles of type $\lambda \in \mathcal{L}_\Lambda$, where we allow for them to consist of a conjuncted belief and goal expression, or single belief and goal expressions. The latter would be the case if either the belief or goal condition

would be empty (or `true`), which does not apply in the present case, where we assume an agent always has a goal for which it performs its actions.

Definition 6 (Rule Descriptions). *If* $\lambda \in \mathcal{L}_\Lambda$ *and* $\pi \in \mathcal{L}_\Pi$*, then* $\mathsf{rd} \in \mathcal{L}_{\mathrm{RD}}$ *is a rule description, and is defined as follows.*

$$\mathsf{rd} ::= \lambda \rightarrow \mathsf{plan}(\pi)$$

One might argue that such a description is imperfect with respect to the actual execution of those rules by the agent, as it does not take the deliberation process into account. This is absolutely true, because if some precondition would be satisfied for multiple plans, then logically all these plans would have to be applied simultaneously. It goes without saying that in actual agent execution this would be disastrous! Moreover, a belief precondition β might cease to hold after the plan has been selected and execution started.

However, in the context of *explaining* observed behavior, such a description is sufficiently accurate. Better still, the generality which would be undesirable with respect to execution, actually serves to benefit in the case of explanation. This is because in the latter case the reasoning goes from observation to precondition, and if multiple explanations (mental preconditions) can account for some (partially) observed plan then this is all the more desirable, considering that these preconditions *could* indeed apply from the viewpoint of the observer.

This is not to say that taking into account procedural specifics would not aid mental state inference. If such specifics — which might for example entail that some rule is more likely to have been applied because of the way the agent's underlying reasoning mechanism works — are available, then they can, and probably should, be used. However, since our goal is to provide a general account, we omit such specifics here.

4 Mental State Abduction

In this section, we provide the details of our framework for mental state abduction. Before moving on to more high-level concepts, we introduce some prerequisite functions and relations which will be used later on to define the primary notions of our approach.

4.1 Functions and Relations

First, we introduce a trace generator function τ, which takes a single plan expression π, as defined in Definition 2, as its input, and maps to a set of perception expressions δ. These perception expressions can be thought of as the possible *observable traces* of the execution of the plan π. Internal test actions on propositional formulas are discarded in the translation process (mapped to the empty action ϵ) and do not appear in the trace. Where choice between plans occurs, two traces are generated; one for each plan in the scope of the choice operator. The operator $\cup$ is set union, and $\circ : \wp(\mathcal{L}_\Delta) \times \wp(\mathcal{L}_\Delta) \rightarrow \wp(\mathcal{L}_\Delta)$ is a non-commutative composition operator defined as $\Phi \circ \Psi = \{\phi; \psi \mid \phi \in \Phi \,\&\, \psi \in \Psi\}$, and $\Phi, \Psi \subseteq \mathcal{L}_\Delta$.

Definition 7 (Trace Generator Function). *The function* $\tau : \mathcal{L}_\Pi \rightarrow \wp(\mathcal{L}_\Delta)$ *is a trace generator function that translates plan expressions* $\pi \in \mathcal{L}_\Pi$ *to sets of perception expressions* $\delta \in \mathcal{L}_\Delta$*, and is defined as follows.*

$$\begin{aligned}
\tau(\alpha) &= \{\alpha\} \\
\tau(\phi?) &= \{\epsilon\} \\
\tau(\pi_1 + \pi_2) &= \tau(\pi_1) \cup \tau(\pi_2) \\
\tau(\pi_1; \pi_2) &= \tau(\pi_1) \circ \tau(\pi_2)
\end{aligned}$$

Not only are we interested in the observable traces of a plan, we are also interested in partial traces that are structurally related to the original trace. To identify these we specify the prefix, suffix, subsequence and dilution relations in Def. 8 and 9 — $\preccurlyeq$, $\succcurlyeq$, $\sqsubseteq$, and $\lesssim$, respectively — which are all partial orders (reflexive, antisymmetric, transitive relations) on $\mathcal{L}_\Delta$. Note that the subsequence relation $\sqsubseteq$ subsumes the prefix relation $\preccurlyeq$ and the suffix relation $\succcurlyeq$, and that dilution $\lesssim$ subsumes subsequence $\sqsubseteq$.

Definition 8 (Prefix, Suffix, Subsequence Relations). *The prefix relation* $\preccurlyeq$*, suffix relation* $\succcurlyeq$*, and subsequence relation* $\sqsubseteq$ *are partial orders on the domain* $\mathcal{L}_\Delta$*, and are defined as follows.*

$$\begin{aligned}
\preccurlyeq &= \{\ (\delta_1\ ,\ \delta_1; \delta_2) \quad \mid \delta_1, \delta_2 \in \mathcal{L}_\Delta \quad \} \\
\succcurlyeq &= \{\ (\delta_1\ ,\ \delta_2; \delta_1) \quad \mid \delta_1, \delta_2 \in \mathcal{L}_\Delta \quad \} \\
\sqsubseteq &= \{\ (\delta_2\ ,\ \delta_1; \delta_2; \delta_3) \mid \delta_1, \delta_2, \delta_3 \in \mathcal{L}_\Delta \} \quad \cup \quad \preccurlyeq \quad \cup \quad \succcurlyeq
\end{aligned}$$

The dilution relation $\lesssim$ is defined as the closure of the subsequence relation under a special 'dilution concatenation' operator dc, which maps two tuples (δ_1, δ_2) and (δ_3, δ_4) to the tuple $(\delta_1; \delta_3, \delta_2; \delta_4)$.

Definition 9 (Dilution Relation). *The dilution relation* $\lesssim$ *is a partial order on the domain* $\mathcal{L}_\Delta$*, defined as the closure of* $\sqsubseteq$ *under the operator* $\mathsf{dc} : (\mathcal{L}_\Delta \times \mathcal{L}_\Delta) \times (\mathcal{L}_\Delta \times \mathcal{L}_\Delta) \rightarrow \mathcal{L}_\Delta \times \mathcal{L}_\Delta$*, such that* $\lesssim = \bigcup_{n \in \mathbb{N}} \mathsf{cl}_n(\sqsubseteq)$*.*

$$\begin{aligned}
&\mathsf{dc}((\delta_1, \delta_2), (\delta_3, \delta_4)) = (\delta_1; \delta_3, \delta_2; \delta_4) \\
&\mathsf{cl}_0(\sqsubseteq) = \sqsubseteq \\
&\mathsf{cl}_n(\sqsubseteq) = \ \mathsf{cl}_{n-1}(\sqsubseteq) \cup \{\ \mathsf{dc}((\delta_1, \delta_2), (\delta_3, \delta_4)) \mid (\delta_1, \delta_2), (\delta_3, \delta_4) \in \mathsf{cl}_{n-1}(\sqsubseteq)\ \}
\end{aligned}$$

The dilution relation might be somewhat difficult to understand. What it (informally) states is that a dilution of some expression can be obtained by randomly removing any number of elements from the original perception expression, whilst preserving the order. For example, $\alpha_1; \alpha_3; \alpha_5$ is a dilution of the expression $\alpha_1; \alpha_2; \alpha_3; \alpha_4; \alpha_5$ with α_2 and α_4 removed and the order preserved, but $\alpha_1; \alpha_5; \alpha_3$ is not a dilution of this expression, nor is $\alpha_1; \alpha_3; \alpha_5; \alpha_6$.

In Definition 10 we use the structural relations to define so-called relational functions, which map an expression that is given as input to the set of perception

expressions related to it under one of the three structural relations. The output is the set of all prefixes / subsequences / dilutions of the expression that was given as input. Note that the suffix relation $\preccurlyeq$ is not used here explicitly, but only serves as part of the definition of the subsequence relation.

Definition 10 (Relational Functions). *The relational functions* $\mathsf{rf}_{\preccurlyeq} : \mathcal{L}_\Delta \to \wp(\mathcal{L}_\Delta)$, $\mathsf{rf}_{\sqsubseteq} : \mathcal{L}_\Delta \to \wp(\mathcal{L}_\Delta)$, *and* $\mathsf{rf}_{\lesssim} : \mathcal{L}_\Delta \to \wp(\mathcal{L}_\Delta)$ *map perception expressions* $\delta \in \mathcal{L}_\Delta$ *to sets of perception expressions, and are defined as follows.*

$$\mathsf{rf}_{\preccurlyeq}(\delta) = \{\delta' \mid \delta' \preccurlyeq \delta\}$$
$$\mathsf{rf}_{\sqsubseteq}(\delta) = \{\delta' \mid \delta' \sqsubseteq \delta\}$$
$$\mathsf{rf}_{\lesssim}(\delta) = \{\delta' \mid \delta' \lesssim \delta\}$$

4.2 Partial Traces for Explanatory Strategies

Now that the basic notions have been defined, it is time to put them to use in our approach to finding explanations for observed behavior. First of all, though, three perceptory conditions are considered, which the observer can assume to reflect its beliefs about the nature of the environment in which the agent operates, or to reflect its beliefs about its own status with respect to perception of the agent's actions. This will be explained in more detail later on.

Definition 11 (Perceptory Conditions). *The following perceptory conditions are distinguished, reflecting either the observer's assumption about the nature of the environment, or its status regarding perception of the agent's actions.*

- ***Complete observation:*** *If complete observation is assumed, then the observer expects to see every action the agent performs.*
- ***Late observation:*** *Late observation reflects the observer's assumption that it has possibly failed in seeing one or more of the initial actions the agent performed. From the moment it starts observing the observer expects to see every future action of the agent.*
- ***Partial observation:*** *In the case of partial observation, the observer assumes that it might fail to see some of the actions the agent performs. Such failure might occur due to some environmental circumstance, or due to the agent deliberately obscuring its actions.*

The perceptory conditions can be used to model the observer's expectations with respect to what it can and will perceive. Late observation, for instance, reflects the fact that the observer believes it has arrived 'late', and has missed some actions the agent has performed.[1] Partial observation can reflect the fact that the observer has to divide its attention among several tasks, on which it bases its assumption that it will miss out on some perceptions.

But perceptory conditions can also reflect beliefs about properties outside of the observer's control. Partial observation, again, might reflect the observer's

[1] Note that 'late' refers to the observer possibly observing the plan after execution has started, and not to some temporal delay in perception.

belief that the agent is deliberately hiding its actions from sight, or it might reflect incompleteness of perception due to some property of the environment. It should be noted that complete and late observation have in common that the observer expects to see every action of the agent.

The descriptions of the agent's operational rules capture a logical connection between the agent's mental state and some plan. To abduce the mental state based on those descriptions, the observer somehow has to compare the sequence of actions it has perceived to some plan it believes the agent might be executing. This is where the trace generator function and the relational functions come together. Combined, they allow for generating partial traces which are structurally related, under one of the three relations $\preccurlyeq$, $\sqsubseteq$, and $\lesssim$, to the set of complete traces of a plan as produced by the plan translation function τ.

Those partial traces fit in nicely with the perceptory conditions. In the case of complete observation, the observer expects to see every action the agent performs. This means the first action the observer has perceived should be the first action of one or more of the traces of the plan the agent is executing, and of which the observer should have information in its description of the agent's rules. Following the same line of reasoning, any sequence of actions the observer perceives should be the prefix of one of those traces. Late observation presumes the observer might have missed the initial actions of some plan, but has seen any actions performed since it started observing. Therefore, any observed sequence must be a subsequence of some plan trace. Partial observation presumes the agent has seen some actions of a plan — in the same order as they occurred in the plan — but might have missed others. It will be no surprise that in this case the observed sequence is a dilution of some plan trace.

Putting all this together, we define the functions in Definition 12, which take as input a plan expression, and generate the set of partial traces which capture the previously mentioned perceptory conditions. For this reason, the functions are indexed with c in the case of complete observation, l for late observation, and p for partial observation.

Definition 12 (Partial Trace Generator Functions). *The partial trace generator functions* $\chi_c : \mathcal{L}_\Pi \to \wp(\mathcal{L}_\Delta)$, $\chi_l : \mathcal{L}_\Pi \to \wp(\mathcal{L}_\Delta)$, *and* $\chi_p : \mathcal{L}_\Pi \to \wp(\mathcal{L}_\Delta)$ *translate plan expressions* $\pi \in \mathcal{L}_\Pi$ *to sets of perception expressions, and are defined as follows.*

$$\chi_c(\pi) = \bigcup \{\ \mathsf{rf}_{\preccurlyeq}(\delta) \mid \delta \in \tau(\pi)\ \}$$
$$\chi_l(\pi) = \bigcup \{\ \mathsf{rf}_{\sqsubseteq}(\delta) \mid \delta \in \tau(\pi)\ \}$$
$$\chi_p(\pi) = \bigcup \{\ \mathsf{rf}_{\lesssim}(\delta) \mid \delta \in \tau(\pi)\ \}$$

The output of $\chi_c(\pi)$ is the set containing every prefix of each trace of the plan π. The output of $\chi_l(\pi)$ is the set of subsequences of each trace of π, and $\chi_p(\pi)$ maps the plan expression π to the set of dilutions of each of its traces.

4.3 Agent and Observer

Here we define the configurations of the agent and the observer. We intentionally don't specify the semantics of the agent program formally, but present the agent configuration using similar terminology as [2,3], where possible semantics of agent operation are defined.

Definition 13 (Agent Configuration). *$\langle \sigma, \gamma, \Pi, \mathcal{R} \rangle$ is an agent configuration, defined as a tuple of a belief base $\sigma \subseteq \mathcal{L}$, a goal base $\gamma \subseteq \mathcal{L}$, the agent's plan base $\Pi \subseteq \mathcal{L} \times \mathcal{L}_\Pi$, and a set of operational rules $\mathcal{R} \subseteq \mathcal{L}_\mathrm{R}$.*

The configuration of an agent consists of three dynamic structures: the belief base σ, consisting of propositional facts expressing the agents beliefs, the goal base γ which consists of propositional facts that express the state of affairs the agent wishes to realise, and a plan base Π consisting of a tuple of goals and plans, which links a plan to the goal for which it has been generated. The only static structure (meaning it does not change during agent execution), and the one we are most interested in, is the agent's set of operational rules $\mathcal{R}$, as described in Section 3.1. From this set $\mathcal{R}$ of the agent's rules it is that the configuration of the observer is generated.

Definition 14 (Observer Configuration). *The configuration of the observer, is a tuple $\langle \Delta, \mathcal{RD}, \Lambda, \Xi_\mathsf{c}, \Xi_\mathsf{l}, \Xi_\mathsf{p} \rangle$ of a perceptory base $\Delta \in \mathcal{L}_\Delta \times \mathcal{L}_\Delta$, rule descriptions $\mathcal{RD} \subseteq \mathcal{L}_\mathrm{RD}$, pre-specified abducibles $\Lambda \subseteq \mathcal{L}_\Lambda$, and sets of partial plan traces $\Xi_\mathsf{c}, \Xi_\mathsf{l}, \Xi_\mathsf{p} \subseteq \mathcal{L}_\Delta \times \mathcal{L}_\Pi$.*

$$
\begin{aligned}
\mathcal{RD} &= \{\ \mathsf{goal}(\kappa) \wedge \mathsf{belief}(\beta) \rightarrow \mathsf{plan}(\pi) \mid (\kappa \leftarrow \beta \mid \pi) \in \mathcal{R}\ \} \\
\Lambda &= \{\ \lambda \mid (\lambda \rightarrow \mathsf{plan}(\pi)) \in \mathcal{RD}\ \} \\
\Xi_\mathsf{c} &= \{\ (\delta, \pi) \mid (\lambda \rightarrow \mathsf{plan}(\pi)) \in \mathcal{RD} \ \&\ \delta \in \chi_c(\pi)\ \} \\
\Xi_\mathsf{l} &= \{\ (\delta, \pi) \mid (\lambda \rightarrow \mathsf{plan}(\pi)) \in \mathcal{RD} \ \&\ \delta \in \chi_l(\pi)\ \} \\
\Xi_\mathsf{p} &= \{\ (\delta, \pi) \mid (\lambda \rightarrow \mathsf{plan}(\pi)) \in \mathcal{RD} \ \&\ \delta \in \chi_p(\pi)\ \}
\end{aligned}
$$

We present the sets Ξ as static precomputed entities for the sake of theoretic analysis, although in practice an algorithm which computes on-the-fly which plans match the perceived sequence of actions would be preferred for the sake of computability and resources. The only dynamic structure in the observer configuration is the base of incoming perceptory events Δ, of which we will speak more in the next section. The static structures can all be generated from the agent's set of rules $\mathcal{R}$, because these rules contain the necessary information for explaining the agent's behavior.

4.4 Explanation of Observed Actions

In Definition 14, we mentioned the observer's perceptory base Δ, without defining its actual content. Assume $\Delta = (\delta, \alpha)$, by which we mean that Δ is tuple of some sequence of actions δ the observer has remembered, and the last perceived

action α, which the observer receives as an event. We assume that memory can hold a sequence of arbitrary length.

The occurrence of a perception event, signifying that action α has been observed, is what triggers explanation. We now compare mental state abduction to logical abduction as presented in Section 2.1, and consider the following schema in analogy to the one in Definition 1.

Classical abduction		**Mental state abduction**
$\Theta \cup \phi \models \psi$	(1)	$\mathcal{RD} \cup \lambda \models \delta$
$\Theta \cup \phi \not\models \bot$	(2)	$\mathcal{RD} \cup \lambda \not\models \bot$
$\Theta \not\models \psi$ and $\phi \not\models \psi$	(3)	$\mathcal{RD} \not\models \delta$ and $\lambda \not\models \delta$

Most of the conditions that apply to classical abduction do not apply to the specific case of mental state abduction. There is no danger of having (logically) contradictory explanations, nor can the sequence δ of actions which have been perceived directly follow from the rule descriptions or a lone abducible, as is the case in (2) and (3), respectively. But — and this is more worrying — the perceived action sequence can't even be explained from the set of rule descriptions $\mathcal{RD}$ and some abducible λ put together, as follows from (1). Fortunately, shown in (1′), it *is* possible to explain something of the form $\mathsf{plan}(\pi)$.

$$\Theta \cup \phi \models \psi \qquad (1') \qquad \mathcal{RD} \cup \lambda \models \mathsf{plan}(\pi)$$

To explain an observation, the only thing left to do now is to relate observed actions of the form δ to a plan descriptor of the form $\mathsf{plan}(\pi)$. The mechanism for this is already in place; we can use the tuples (δ, π) of partial plan traces and plans in Ξ, as defined in Definition 14. The careful reader will have noticed that the sets of partial plan traces are indexed with c, l, and p, and indeed these correspond to the partial traces suited for relating observed sequences to plans, under the conditions of complete, late, and partial observation, respectively.

To actually generate explanations based on some observation, we define three explanatory functions expl_c, expl_l, and expl_p, which generate the set of explanations that account for the observed sequence of actions, under the perceptory conditions of complete, late, and partial observation. Thus, if some observed sequence of actions δ is a subsequence but *not* a prefix of one of the traces of plan π, then the explanation $\lambda \in \Lambda$ for which $\mathcal{RD} \cup \lambda \models \mathsf{plan}(\pi)$ applies will be in the set of explanations given by expl_l and expl_p, but not in the set given by expl_c (or only as explanation based on another plan π' of which δ *is* a prefix). In Def. 15 the formal definition of these explanatory functions is presented.

Definition 15 (Explanatory Functions). *Explanatory functions* $\mathsf{expl}_\mathsf{c} : \mathcal{L}_\Delta \times \wp(\mathcal{L}_{\mathrm{RD}}) \times \wp(\mathcal{L}_\Lambda) \times \wp(\mathcal{L}_\Delta \times \mathcal{L}_\Pi) \to \wp(\mathcal{L}_\Lambda)$, $\mathsf{expl}_\mathsf{l} : \mathcal{L}_\Delta \times \wp(\mathcal{L}_{\mathrm{RD}}) \times \wp(\mathcal{L}_\Lambda) \times \wp(\mathcal{L}_\Delta \times \mathcal{L}_\Pi) \to \wp(\mathcal{L}_\Lambda)$, *and* $\mathsf{expl}_\mathsf{p} : \mathcal{L}_\Delta \times \wp(\mathcal{L}_{\mathrm{RD}}) \times \wp(\mathcal{L}_\Lambda) \times \wp(\mathcal{L}_\Delta \times \mathcal{L}_\Pi) \to \wp(\mathcal{L}_\Lambda)$, *map*

to a finite set of abducibles $\lambda \in \Lambda$, given a finite perception $\delta \in \mathcal{L}_\Delta$, a finite set of rule descriptions $\mathcal{RD} \subseteq \mathcal{L}_{\mathrm{RD}}$, and a finite set of abducibles $\Lambda \subseteq \mathcal{L}_\Lambda$.

$$\begin{aligned}
\mathsf{expl_c}(\delta, \mathcal{RD}, \Lambda, \Xi_\mathsf{c}) &= \{\ \lambda \in \Lambda \mid (\mathcal{RD} \cup \lambda) \models \mathsf{plan}(\pi)\ \&\ (\delta, \pi) \in \Xi_\mathsf{c}\ \} \\
\mathsf{expl_l}(\delta, \mathcal{RD}, \Lambda, \Xi_\mathsf{l}) &= \{\ \lambda \in \Lambda \mid (\mathcal{RD} \cup \lambda) \models \mathsf{plan}(\pi)\ \&\ (\delta, \pi) \in \Xi_\mathsf{l}\ \} \\
\mathsf{expl_p}(\delta, \mathcal{RD}, \Lambda, \Xi_\mathsf{p}) &= \{\ \lambda \in \Lambda \mid (\mathcal{RD} \cup \lambda) \models \mathsf{plan}(\pi)\ \&\ (\delta, \pi) \in \Xi_\mathsf{p}\ \}
\end{aligned}$$

These explanatory functions can be used as part of different explanatory strategies, which reflect the conditions defined in Def. 11. For example, if an observer believes it will perceive all actions of an agent which it believes to have started execution of some plan, then the observer might consider only those explanations provided by $\mathsf{expl_l}$, and omit those provided by $\mathsf{expl_c}$ from the set of possible explanations.

An observer might also try using strategies based on different perceptory conditions, to see what works best. For example, it might be the case that the presumption of complete observation yields no explanation for the observed action sequence. The observer then might try explanation under some other perceptory condition, and success or failure of this process can shape its ideas about the behavior of the agent and/or the nature of the environment.

4.5 Propositions and Proofs

In this section we state some important properties of our technique, and provide formal proof of their veracity. In Proposition 1 we claim that some explanatory functions are guaranteed to yield more explanations than others, and in Proposition 2 we claim that the number of explanations decreases monotonically, as more actions are perceived. We also claim that a single action can always be explained, a proposition of which we do not give formal proof but which we justify by referring to previously made assumptions.

Proposition 1. *The number of explanations generated by the function $\mathsf{expl_c}$ is less or equal to that of the function $\mathsf{expl_l}$, which is in turn less or equal to that of the function $\mathsf{expl_p}$.*

Proof. The explanatory functions $\mathsf{expl_c}$, $\mathsf{expl_l}$, and $\mathsf{expl_p}$, as defined in Def. 15, yield a finite set of elements $\lambda \in \mathcal{L}_\Lambda$, given some finite input. They differ only in the fact that they are based on three different sets of tuples of perception expressions and plan expressions; Ξ_c for $\mathsf{expl_c}$, Ξ_l for $\mathsf{expl_l}$, and Ξ_p for $\mathsf{expl_p}$. As follows from Def. 10, 12, and 14, the defining characteristic of these tuples are the relations presented in Def. 8 & 9. From these definitions it follows that $(\preccurlyeq) \subseteq (\sqsubseteq) \subseteq (\lesssim)$, and because these relations define the sets of partial plan traces Ξ_c, Ξ_l, and Ξ_p, respectively, it follows that $\Xi_\mathsf{c} \subseteq \Xi_\mathsf{l} \subseteq \Xi_\mathsf{p}$ and $|\Xi_\mathsf{c}| \leq |\Xi_\mathsf{l}| \leq |\Xi_\mathsf{p}|$. Given any finite perception expression δ, finite set of rule descriptions $\mathcal{RD}$, and finite set of abducibles Λ, let $\mathsf{expl_c}(\delta, \mathcal{RD}, \Lambda, \Xi_\mathsf{c}) = C$, $\mathsf{expl_l}(\delta, \mathcal{RD}, \Lambda, \Xi_\mathsf{l}) = L$, and $\mathsf{expl_p}(\delta, \mathcal{RD}, \Lambda, \Xi_\mathsf{p}) = P$. It then follows that $|C| \leq |L| \leq |P|$. □

Lemma 1. *For any relation $R \in \{\preccurlyeq, \sqsubseteq, \lesssim\}$, if $(\delta'; \delta'', \delta) \in R$, then $(\delta', \delta) \in R$.*

Proof. It follows from the definition of the relations that $(\delta', \delta'; \delta'') \in R$*, because every relation subsumes* $\preccurlyeq$*. If this is the case, then it follows by transitivity that if* $(\delta', \delta'; \delta'') \in R$ *and* $(\delta'; \delta'', \delta) \in R$*, then* $(\delta', \delta) \in R$*.* □

Proposition 2. *The number of explanations under any single explanatory relation decreases monotonically (either decreases or stays the same) when* α *is explained as the next action of an already explained sequence of actions* δ*.*

Proof. Take some explanatory function expl*. Assume some perception* δ *has been observed and explained, yielding a finite set of explanations* X*. The number of explanations in* X *is* $|X|$*, the cardinality of* X*. An incoming observation* α *is added to* δ*, yielding the new perception* $\delta; \alpha$*. The only way the cardinality of the explanation set could increase with an incoming perception* α*, would be if* $(\delta, \pi) \notin \Xi$ *and* $(\delta; \alpha, \pi) \in \Xi$*, for some plan* π *and some set of partial traces and plan tuples* Ξ*. We now consider* Ξ_{c}*,* Ξ_{l}*, and* Ξ_{p}*, specifically. By Lemma 1, for any defining relation* R*, if* $(\delta; \alpha, \delta') \in R$*, then* $(\delta, \delta') \in R$*. If* $(\delta; \alpha, \pi) \in \Xi_{\mathsf{c}}$*, then* $\delta; \alpha \preccurlyeq \delta'$ *for some* $\delta' \in \chi_c(\pi)$*. If* $(\delta; \alpha, \pi) \in \Xi_{\mathsf{l}}$*, then* $\delta; \alpha \sqsubseteq \delta'$*, for some* $\delta' \in \chi_l(\pi)$*. If* $(\delta; \alpha, \pi) \in \Xi_{\mathsf{p}}$*, then* $\delta; \alpha \lesssim \delta'$ *for some* $\delta' \in \chi_p(\pi)$*. In all cases, if* $(\delta; \alpha, \pi) \in \Xi$ *then* $(\delta, \pi) \in \Xi$*. Therefore* $|X|$ *cannot increase with incoming perceptions, and* $|X|$ *must decrease monotonically.* □

One desirable property which we state but do not prove formally is the fact that a single perceived action can always be explained. This follows from the assumptions in Section 3.1, which can be summarized to the fact that any action an agent executes is part of some plan belonging to a known rule. If $\Delta = (\delta, \alpha)$ and $\delta; \alpha$ cannot be explained (the explanatory function gives an empty set), perhaps because the agent started working on another plan, then it is always possible to explain the last perceived action using $\Delta = (\epsilon, \alpha)$ using $\mathsf{expl}_{\mathsf{p}}$, since α *must* be part of a known rule, and is the dilution of some plan trace.

5 An Example

In this section, we present a brief example of the technique of mental state abduction, situated in a role-playing game (RPG) setting. In RPGs, a player is often accompanied by one or more non-player characters (NPCs), which act autonomously and (most of the time) are not under the player's direct control. In a utopian future where all characters are implemented as BDI agents, the player and his or her NPC companions, on a quest to deliver an important letter, are under attack by a band of brigands.[2] One of the brigands somehow knows the whereabouts of this letter, and has a plan for having it in his possession.

[2] The setting of a fight might seem rather violent, but it serves nicely to illustrate various points which deserve attention. There can be multiple friendly and enemy agents involved, as well as a human player. Moreover, the chaos of a fight warrants the assumption of partial observation.

$$\overbrace{\mathsf{have(letter)}}^{\kappa} \leftarrow \overbrace{\mathsf{in(chest, letter)} \wedge \neg\mathsf{guarded(chest)}}^{\beta} \mid \left.\begin{array}{l} {}^{\alpha_1}\mathsf{goto(chest)};\ {}^{\alpha_2}\mathsf{inspect(chest)}; \\ \quad ({}^{\phi_1}\mathsf{locked(chest)?}; \\ \quad\quad ({}^{\phi_2}\mathsf{sturdy(lock)?};\ {}^{\alpha_3}\mathsf{sheath(sword)};{}^{\alpha_4}\mathsf{pick(lock)})+ \\ \quad\quad ({}^{\neg\phi_2}\neg\mathsf{sturdy(lock)?};\ {}^{\alpha_5}\mathsf{smash(lock)}))+ \\ \quad ({}^{\neg\phi_1}\neg\mathsf{locked(chest)?})); \\ {}^{\alpha_6}\mathsf{open(chest)};{}^{\alpha_7}\mathsf{take(letter)} \end{array}\right\}\pi$$

This specific plan for having the letter has as a condition that the letter is believed to be in the chest, and the chest to be unguarded. The plan consists of inspecting the chest, picking or smashing the lock (depending on its quality) in case the chest is locked, then opening it and taking the letter. Notice that actions and tests have been annotated with α and ϕ, respectively.

The rule can also be represented as follows, using the superscripted annotations instead of the actual formulas.

$$\kappa \leftarrow \beta \mid \alpha_1;\alpha_2;(\phi_1?;(\phi_2?;\alpha_3;\alpha_4)+(\neg\phi_2?;\alpha_5))+(\neg\phi_1?);\alpha_6;\alpha_7$$

In the chaos of the fight, nobody is guarding the chest in which the letter has been stored. The brigand with the plan for stealing the letter takes notice of this. Fortunately, one of the NPCs sees the brigand inspecting the chest, and also has knowledge of the rule the brigand might use to achieve the goal of having the letter in its possession.

Assume the NPC is abducing the brigand's mental state using our technique of mental state abduction. The set of possible plan traces $\tau(\pi)$ then looks as follows, with overbraced conditions indicating a trace.

$$\tau(\pi) = \{\overbrace{\alpha_1;\alpha_2;\alpha_3;\alpha_4;\alpha_6;\alpha_7}^{\phi_1\wedge\phi_2},\ \overbrace{\alpha_1;\alpha_2;\alpha_5;\alpha_6;\alpha_7}^{\phi_1\wedge\neg\phi_2},\ \overbrace{\alpha_1;\alpha_2;\alpha_6;\alpha_7}^{\neg\phi_1}\}$$

The NPC observing the brigand perceived that the brigand was inspecting the chest. Assuming it didn't perceive anything prior to this point means the perception base was empty, and now $\Delta = (\epsilon, \alpha_2)$. Because of the tumultuous nature of the fight, the NPC assumes it might miss perceptions and therefore uses $\mathsf{expl_p}$. In this case, if we take for granted that $\mathsf{expl_p}(\alpha_2, \mathcal{RD}, \Lambda, \Xi_\mathsf{p}) = X$, then $\{\mathsf{goal(have(letter))} \wedge \mathsf{belief(in(chest, letter)} \wedge \neg\mathsf{guarded(chest))}\} \subseteq X$. Note that if the explanatory function $\mathsf{expl_c}$ had been used, this mental state could not have been abduced, as α_2 is not a prefix of a trace of π. Of course, other plans might involve opening the chest, and their mental state preconditions might also be part of the set of explanations produced by any of the explanatory functions.

The NPC then diverts its attention because of a brigand charge, and the next time it looks it sees the agent picking the lock, and since $\mathsf{pick(lock)} = \alpha_4$, the new perception is $\Delta = (\alpha_2, \alpha_4)$. Since $(\alpha_2;\alpha_4,\pi) \notin \Xi_\mathsf{c}$ and $(\alpha_2;\alpha_4,\pi) \notin \Xi_\mathsf{l}$, but

$(\alpha_2; \alpha_4, \pi) \in \Xi_{\mathsf{p}}$, only the explanatory function for partial observation $\mathsf{expl}_{\mathsf{p}}$ can explain the perceived action sequence $\alpha_2; \alpha_4$ in relation to the plan π.

Depending on its rules, the NPC can respond to the brigand trying to take the letter. It might inform others, or actively intervene itself. Note that the plan which is represented here is fictional and only shown for illustrative purposes, although nothing prevents it from being the part of the rule of a BDI-based virtual character in a game. Furthermore, it should be noted that in the observation of actions there is no indicator of which agent performed the action. This is due to our presentation of an observer as separated from the agent and the assumption of a single agent; in a setting with multiple agent such an indicator would be required to distinguish between the actions of different agents.

6 Conclusion and Future Work

In this paper we presented a technique called mental state abduction, which allows for inferring the mental states of BDI agents based on their observed behavior, using an approach inspired by abduction in propositional logic. Three explanatory strategies were presented, based on perceptory conditions reflecting either properties under the observer's influence, or properties of the environment beyond the observer's control. Our technique was shown to work for agents programmed in a simplified BDI-based programming language, as demonstrated by formal proofs of some interesting properties, and a game-like example.

In future research we mean to extend mental state abduction to deal with the dynamics of a current agent programming language such as 2APL. Specifically, we intend to make use of the internal actions inside plans, such as conditional (belief) tests, as a guideline for better abduction. The observer itself can then execute the test actions the agent executes, to select among plans involving identical actions. This also opens the opportunity for the observer to have a theory of mind of the agent, meaning the observer has a mental model of the possible mental state of the agent. Such a model would have specific requirements, such as a revision procedure to deal with inconsistent information, and possibly constructs for combining queries to the model of the agent's mental state with queries to the observer's own mental state, in order to represent reasoning in the line of "What would I have done if I were in this agent's situation?". Of course, such a model of the mental state of other agents is required in the first place to represent the mental states inferred by mental state abduction, and for those explanations to be used by the observer in its capacity of agent.

Furthermore, we see the possibility to incorporate institutional notions such as roles to provide restrictions on specific clusters of rules an agent might be expected to have. In line with this, we intend to relax the requirement of the observing party having to know all of the agent's rules.

References

1. Norling, E., Sonenberg, L.: Creating interactive characters with BDI agents. In: Proc. of the Australian Workshop on Interactive Entertainment, pp. 69–76 (2004)
2. Dastani, M., Meyer, J.-J.C.: A practical agent programming language. In: Dastani, M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS, vol. 4908, pp. 107–123. Springer, Heidelberg (2008)
3. Dastani, M.: 2APL: A practical agent programming language. Autonomous Agents and Multi-Agent Systems 16, 214–248 (2008)
4. Winikoff, M.: JACK^{TM}Intelligent Agents: An industrial strength platform. In: Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) Multi-Agent Programming, pp. 175–193. Springer, Heidelberg (2005)
5. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A BDI reasoning engine. In: Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) Multi-Agent Programming, pp. 149–174. Springer, Heidelberg (2005)
6. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley-Interscience, Hoboken (2007)
7. Doyle, P.: Believability through context. In: Proceedings of AAMAS 2002, pp. 342–349. ACM Press, New York (2002)
8. Flach, P.: Conjectures: An Inquiry Concerning the Logic of Induction. PhD thesis, University of Tilburg (1995)
9. Hartshorne, C., Weiss, P. (eds.): Collected Papers of C.S. Peirce. Harvard University Press (1931-1958)
10. Aliseda, A.: A unified framework for abductive and inductive reasoning in philosophy and AI. In: Proceedings of the ECAI 1996 Workshop on Abductive and Inductive Reasoning (1996)
11. Kakas, A., Kowalski, R., Toni, F.: The Role of Abduction in Logic Programming. In: Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 5, pp. 235–324. Oxford University Press, Oxford (1998)
12. Aliseda-Llera, A.: Seeking Explanations: Abduction in Logic, Philosophy of Science, and Artificial Intelligence. PhD thesis, University of Amsterdam (1997)
13. Harman, G.: Inference to the best explanation. The Philosophical Review 74, 88–95 (1965)
14. Lipton, P.: Inference to the Best Explanation. In: Inference to the Best Explanation, 2nd edn., Routledge (2004)
15. Carberry, S.: Techniques for plan recognition. User Modeling and User-Adapted Interaction 11(1-2), 31–48 (2001)
16. Albrecht, D.W., Zukerman, I., Nicholson, A.E.: Bayesian models for keyhole plan recognition in an adventure game. User Modeling and User-Adapted Interaction 8(1-2), 5–47 (1998)
17. Appelt, D.E., Pollack, M.E.: Weighted abduction for plan ascription. User Modeling and User-Adapted Interaction 2(1-2), 1–25 (1991)
18. Goultiaeva, A., Lespérance, Y.: Incremental plan recognition in an agent programming framework. In: Working Notes of the AAAI Workshop on Plan, Activity, and Intention Recognition (PAIR) (2007)

Iterated Belief Revision in the Face of Uncertain Communication

Yoshitaka Suzuki[1], Satoshi Tojo[1], and Stijn De Saeger[2]

[1] Japan Advanced Institute of Science and Technology
{syoshita,tojo}@jaist.ac.jp
[2] National Institute of Information and Communications Technology
stijn@nict.go.jp

Abstract. This paper offers a formalization of iterated belief revision for multiagent communication using the logic of communication graphs introduced in [15]. In this study we consider an agent (i.e., information source) capable of sending two types of message. In the first type, he tells that he *knows* a proposition, but in the second type, he tells that he *believes* a proposition. Consequently, iterated belief revision is brought about through a sequence of communication events (i.e., a *history*), and we propose a variation of the AGM rational postulates for history based belief revision. As we will show, a representation theorem is verified only for a class of restricted communication graphs. We consider this result to be a weak point in the application of the AGM postulates to multiagent communication, and propose a viable alternative.

1 Introduction

Information processing in uncertain communication environments is one of the important problems for the study of multiagent systems. When agents communicate, often less than certain information passes between them. In such a case, we technically cannot say that they acquire *knowledge*, even if the information in question eventually turns out to be true. Here, we distinguish between two types of uncertain information in multiagent communication. On the one hand, an agent may consider a given piece of information correct even though in fact he obtains it from an unreliable information source (via some insecure communication channel), and therefore his information may in fact be incorrect. On the other hand, the agent may not be convinced that this information is correct, but nonetheless consider it plausible. While he does not recognize it in the first case, he is aware of his own uncertainty in the second case. Thus, making an inquiry to some other agent, one agent finds two fallible answers. That is to say, the other agent is firmly convinced of the information or not[1]. These two types of epistemic attitudes, related to knowledge and belief, require a classification of rational processing for uncertain information. Traditional study of belief revision ([1,7]) proposes rational

[1] For the purpose of this paper, we take 'conviction' to be a technical notion that is epistemically stronger than belief, excluding the possibility of error in the agent's own mind. Thus, whereas introspectively an agent does not distinguish between his convictions and knowledge, he would consider his beliefs to be likely though not certain.

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 179–196, 2009.

postulates for the change of an agent's knowledge base called *minimal change*, and prove a representation theorem that tells us that these postulates correspond to certain revision operators. However, Gärdenfors ([7]) stated that the relation between knowledge base and external environment is not essential for the study of belief revision, and he did not distinguish knowledge from belief. Therefore, when a message comes from the other agent, whether the other agent knows something or not is not important for the general study of belief revision.

As an exception to the above characterization, Friedman and Halpern ([5,6]) defined a belief operator in terms of both a knowledge operator and a conditional operator. Their study proposed an expansion of the logic of knowledge for multiagent systems [4], and therefore it can be applied to multiagent systems, although they were interested in iterated belief revision for single agents. However, they assumed that an agent acquires an infallible information by the observation upon the external environment. As a result, any given sequence of external inputs must be consistent, and possible worlds that are inconsistent with the inputs already accepted are systematically eliminated. However, as indicated by Rott's study about conservatism in iterated belief revision [16], such an idea is *radical* for fallible information. Moreover, since they were not interested in communication, their model must not be related with the study of multiagent system. Because multiple agents' communication may give rise to uncertain information, such a formalization is suitable for single agents, but not multiple agents. In fact, they did not describe the application of the framework to multiagent system. Thus, the traditional study of belief revision usually assumed that epistemic input came from external environment, and was generally not interested in multiagent communication.

Example 1. If Alice observes Dean putting on a white shirt, she knows he did so. However, if Alice hears from Bob that he believes Dean put on a white shirt, she does not know for a fact that he did, although she may believe it regardless. Furthermore, when Alice later learns from Charlie that he knows Dean did not put on a white shirt, she will eliminate the previous belief in favor of this newly acquired knowledge.

Studying belief revision in multiagent communication, we utilize the logic of communication graphs from [15]. This logic expresses that information travels via communication channels that ensure its reliability. Only if an agent i can directly receive some piece of information from another agent j and j has acquired this knowledge in a similar fashion, i acquires this piece of knowledge from j. Nevertheless, this logic cannot represent the situation where an agent believes some proposition but does not *know* it. In order to solve the problem, we will not directly use the modal operator K_i for knowledge. Instead, we introduce the operator C_i for *conviction* and $\rightarrow_i$ for conditional, and define knowledge K_i and belief B_i in terms of these two operators. Furthermore, we will define some postulates for belief revision operations that accepts another agent's respective knowledge and beliefs as external inputs, and show that some revision operators are equivalent with these postulates when the communication graph satisfies some condition.

In section 2, we introduce formal preliminaries and semantics for our logic of communication graphs that pays attention to the other agent's knowledge and belief. In section 3, consulting moderate and radical belief revision in Rott [16], we define iterated revision operators for knowledge and belief. In section 4, rational postulates are

proposed, and we show the a representation theorem between the operators and the postulates for those cases where the communication graph satisfies some particular constraints. In section 5, we discuss some remaining problems of our study and its perceived advantages over related works, as well as indicate subsequent work to be done.

2 Preliminaries

In this section, we describe the semantics for our logic of communication graphs with conditionals $\mathcal{CC}(\mathcal{G})$. This is based on the logic of communication graphs from [15], expanded with a conditional operator like the study of belief revision by the logic of knowledge for multiagents by [5,6]. Thus our logic not only allows reasoning about the underlying communication channel, but also represents the plausibility of a given proposition as far as the agents are concerned.

Let $\mathcal{A}$ be the set of agents. Then, a *communication graph* is described by $\mathcal{G}_{\mathcal{A}} = \langle \mathcal{A}, E \rangle$, where $E \subseteq (\mathcal{A} \times \mathcal{A}) \setminus \{(i,i) \mid i \in \mathcal{A}\}$. The edges in E represent whether an agent can directly receive information from another agent or not, i.e., $(i,j) \in E$ means that agent i learns information from agent j directly.

We define that E is *fully connected* iff for all $i, j \in \mathcal{A}$, if $i \neq j$, then $(i,j) \in E$, and E is *existentially connected* iff for all $i \in \mathcal{A}$, for some $j \in \mathcal{A}$, $(i,j) \in E$. In the former case, any agent can communicate with any agent. In the latter case, for any agent, there is an information source.

Suppose that all agents share the (finite) set of atomic propositions At and a special propositional variable L. Intuitively, L means that any information exchange have emerged via reliable communication channel. As already discussed, we introduce the following modal operators; intuitively, $C_i\phi$ means that i is convinced that ϕ; $\phi \rightarrow_i \psi$ means that, given ϕ, ψ is plausible for i; $\Diamond\phi$ means that, after some communications, ϕ becomes true. Precise semantics of these symbols are given later. The set of well-formed formulae consists of

$$\phi := p \mid \neg\phi \mid \phi \wedge \psi \mid C_i\phi \mid \phi \rightarrow_i \psi \mid \Diamond\phi.$$

We use the standard abbreviations for connectives $\vee, \Rightarrow, \Leftrightarrow$, and propositional constants $true$ and $false$. Besides these we introduce the modal operators K_i and B_i as abbreviations, where as usual $K_i\phi$ means that i knows ϕ, and $B_i\phi$ means that i believes ϕ.

$$K_i\phi \stackrel{\text{def}}{=} L \wedge C_i\phi$$

$$B_i\phi \stackrel{\text{def}}{=} C_i(true \rightarrow_i \phi)$$

Thus, agent i knows ϕ iff any information exchange have emerged via reliable communication channel and agent i is convinced that ϕ. Agent i believes ϕ iff agent i is convinced that ϕ is plausible.

The definition of B_i is almost the same as Friedman and Halpern's in [5,6], but in their work belief is defined instead as

$$B_i\phi \stackrel{\text{def}}{=} K_i(true \rightarrow_i \phi).$$

That is to say, agent i believes ϕ iff agent i knows that ϕ is plausible. Their definition of knowledge is not based on communication channel, but our definition depends on the reliability of the communication channel, which is decided by an agent's external environment (See the definition of the semantics of L below)[2]. Thus, whether he knows something or not is not decided by his internal state without external environment. However, belief does not seem to depend on the external environment. Therefore, we introduced modal operator C_i that does not depend on the real external environment, but on the environments that cannot be distinguished from the real environment for the agent i. For details, see the definition fo the semantics of C_i.

Let $At_i \subseteq At$ be the set of atomic propositions of which agent i initially knows the truth value (we do not assume all At_i ($i \in \mathcal{A}$) to be disjoint). Furthermore, this fact about i's knowledge is *common knowledge* among all members of $\mathcal{A}$. So another agent knows that i knows the truth value of elements of At_i, though not what these values actually are. Thus we introduce an information vector $\boldsymbol{At} = \langle At_1, \ldots, At_n \rangle$ representing the initial knowledge of the agents. Given a set W, we say that $v \in W$ is a *world*, where v is a function $v : At \to \{0, 1\}$.

When agent i learns some information ϕ from agent j's knowledge or beliefs, we represent this situation by (i, j, ϕ, e) and call it a *communication event*, where $e = K$ or $e = B$. Technically ϕ is restricted to formulae in disjunctive normal form (DNF), i.e., ϕ is of the form $\bigvee_{i=1,\ldots,k} \bigwedge C_i$, where each C_i is a consistent finite set of literals in At. That is, (i, j, ϕ, K) is an event that agent i learns from agent j that j knows ϕ. Given the set of all communication events $\Sigma_{\mathcal{G}}$, $H \in \Sigma_{\mathcal{G}}^*$ is a finite sequence of events called a *history*, where the empty history is denoted ϵ. We define the temporal ordering over all histories as follows: $H \sqsubseteq H'$ iff $H' = H \cdot H''$ for some H'', where $\cdot$ is a concatenation of communication events.

A history can be considered as a God's eye record of past agent communications, but an agent may only witness part of it. When the second agent j and the third agent k communicate with each other, the first agent i has no access to their communication. Therefore, we introduce i's local history $\lambda_i(H)$, in which i can only recognize those events that concern himself, i.e., in which i receives some information from the other agent. It is defined as follows.

$$\lambda_i(H \cdot (m, j, \phi, e)) = \begin{cases} \lambda_i(H) \cdot (m, j, \phi, e) & \text{if } m = i \\ \lambda_i(H) & \text{otherwise} \end{cases}$$

A pair (v, H) consisting of a system state v and a history H is called a *point*. Given two points (w, H) and (v, H'), we can imagine an agent i unable to distinguish between the two. We can describe this situation in terms of an accessibility relation $\sim_i$ as follows, where $w|_{At_i}$ is a restriction of w whose domain is At_i.

$$(w, H) \sim_i (v, H') \text{ iff } w|_{At_i} = v|_{At_i} \text{ and } \lambda_i(H) = \lambda_i(H')$$

[2] In the following discussion, predicate L is defined by communication channel. Whether an agent knows something or not is justified by L. In other words, the source of the knowledge lies outside of himself. Thus, in our formalization, an agent has no access to the ground of the justification directly. Philosophically, such a view is called *externalism about knowledge* [2].

We use a total preorder $\leq$[3] over W for the definition of conditional operator and we call it a *preference relation*. We denote $v \leq w$ when v is at least as plausible as w. $v < w$ is the strict case of $\leq$, i.e., $v \leq w$ and $w \not\leq v$. We assume that sets of states are also comparable with $\leq$. Thus, $A \leq B$ iff for all $v \in B$, for some $w \in A$, $w \leq v$. The maximal worlds w.r.t. the preference relation are called *implausible worlds*. The set of implausible worlds $Im \subseteq W$ satisfies the following condition: for any $w \in Im$, $v \in W$, $v \leq w$, and if $v \notin Im$, then $w \not\leq v$. Let a *preference assignment function* $\mathcal{P}_i$ for agent i be a mapping from (v, H) to $(\leq, Im)$, i.e., $\mathcal{P}(v, H) = (\leq, Im)$, where $Im = \{w \in W \mid \text{for any } H', (w, H') \not\prec_i (v, H) \text{ or } w, H' \not\models_{\mathcal{M}} L\}$ ($\models_{\mathcal{M}}$ is defined in the following discussion in this page). That is to say, implausible worlds can be distinguished from the actual world in which agent i finds himself. In the following discussion, we define $\mathcal{P}_i(v, H) = (\leq_{i,v,H}, Im_{i,v,H})$. That is, $w \leq_{i,v,H} w'$ means that w is at least as plausible as w' for agent i at world v and history H. Thus, we will define a *communication graph model* as a tuple $\mathcal{M} = \langle \mathcal{G}, \boldsymbol{At}, \boldsymbol{\mathcal{P}}, W \rangle$.

We will introduce the satisfiability relation of legality and truth as follows.

(Legality)

- $w, \epsilon \models_{\mathcal{M}} L$
- $w, H \cdot (i, j, \phi, K) \models_{\mathcal{M}} L$ iff $w, H \models_{\mathcal{M}} L$, $(i, j) \in E$, and $w, H \models_{\mathcal{M}} K_j\phi$
- $w, H \cdot (i, j, \phi, B) \models_{\mathcal{M}} L$ iff $w, H \models_{\mathcal{M}} L$ and $(i, j) \in E$

(Truth)

- $w, H \models_{\mathcal{M}} p$ iff $w(p) = 1$, where $p \in At$.
- $w, H \models_{\mathcal{M}} \neg\phi$ iff $w, H \not\models_{\mathcal{M}} \phi$
- $w, H \models_{\mathcal{M}} \phi \wedge \psi$ iff $w, H \models_{\mathcal{M}} \phi$ and $w, H \models_{\mathcal{M}} \psi$
- $w, H \models_{\mathcal{M}} \Diamond\phi$ iff $\exists H'$ such that $H \sqsubseteq H'$, $w, H' \models_{\mathcal{M}} L$ and $w, H' \models_{\mathcal{M}} \phi$
- $w, H \models_{\mathcal{M}} C_i\phi$ iff $\exists (v, H'), (w, H) \sim_i (v, H')$ and $v, H' \models_{\mathcal{M}} L$, and $\forall (v, H')$, if $(w, H) \sim_i (v, H')$ and $v, H' \models_{\mathcal{M}} L$, then $v, H' \models_{\mathcal{M}} \phi$
- $w, H \models_{\mathcal{M}} \phi \rightarrow_i \psi$ iff $\models_{\mathcal{M}} \phi \Leftrightarrow false$ or $[\![\phi \wedge \psi]\!]^{\mathcal{M}}_{i,w,H} <_{i,w,H} [\![\phi \wedge \neg\psi]\!]^{\mathcal{M}}_{i,w,H}$, where $[\![\phi]\!]^{\mathcal{M}}_{i,w,H} = \{v \in W | \exists H', (w, H) \sim_i (v, H'),$ and $v, H' \models_{\mathcal{M}} L,$ and $v, H' \models_{\mathcal{M}} \phi\}$.

We will explain the meaning of the above definition. Legality is defined as follows. At first, when a history is empty, the point is legal. Next, If there is some legal point, there is some communication graph from agent j to agent i, and j knows ϕ, the pair of the world and the concatenation of the history and the event (i, j, ϕ, K) is also legal. Finally, if there is some legal point, and there is some communication channel from agent j to agent i, then the pair of the world and the concatenation of the history and the event (i, j, ϕ, B) is also legal. So legality means that there is some reliable communication channel.

Note that agent j's knowledge is related with the definition of legality, but not j's belief, since legality is a condition that ensures the truth of knowledge (see the definition

[3] A total preorder $\leq$ is a total and transitive relation. $\leq$ is total iff $w \leq w'$ or $w' \leq w$ for any $w, w' \in W$. $\leq$ is transitive iff for any $w, w' \in W$, if $w \leq w'$ and $w' \leq w''$, then $w \leq w''$.

of $K_i\phi$). While knowledge must be justified by a reliable information source, whether belief needs such a source or not is a difficult epistemological problem for our current study. Therefore, our current stance is that we do not require belief to be justified by legality.

The truth condition of $\neg$ and $\wedge$ is as usual. The formula $\Diamond\phi$ is satisfied when there will be some future point that satisfies ϕ. The formula $C_i\phi$ is true when there is some legal point that is indistinguishable with the real point for agent i, and for any indistinguishable legal point, ϕ is true. The formula $\phi \rightarrow_i \psi$ means that $\phi \wedge \psi$ is more plausible for agent i than $\phi \wedge \neg\psi$ at the point.

The difference in definition between our C_i and Pacuit and Parikh's K_i in [15] is important. Their definition is as follows.

$$w, H \models_{\mathcal{M}} K_i\phi \text{ iff } \forall(v, H'), \text{if } (w, H) \sim_i (v, H') \text{ and } v, H' \models_{\mathcal{M}} L, \text{then } v, H' \models_{\mathcal{M}} \phi$$

That is, their definition of knowledge does not assume that there is some legal point that is not distinguished from the real points for agent i. This definition does not satisfy the modal axiom D, which characterizes an essential property of belief the property of belief (i.e., nobody believes a contradiction.), while the axiom T is known from the logic of knowledge (i.e., veridicality of knowledge). Perhaps the real point may not be legal, and we may not find any legal point that is not distinguished from the real points for agent i. In such a case, K_i does not satisfy D. Therefore, we define C_i as already mentioned.

Thus, the derived modal operators K_i and B_i, defined by L, C_i, and $\rightarrow_i$, satisfy the following traditional axioms.

K: $(K_i\phi \wedge K_i(\phi \Rightarrow \psi)) \Rightarrow K_i\psi$
T: $K_i\phi \Rightarrow \phi$.
4: $K_i\phi \Rightarrow K_iK_i\phi$.
5: $\neg K_i\phi \Rightarrow K_i\neg K_i\phi$.
G: From ϕ infer $K_i\phi$.

K: $(B_i\phi \wedge B_i(\phi \Rightarrow \psi)) \Rightarrow B_i\psi$.
D: $\neg B_i false$.
4: $B_i\phi \Rightarrow B_iB_i\phi$.
5: $\neg B_i\phi \Rightarrow B_i\neg B_i\phi$.
G: From ϕ infer $B_i\phi$.

Note that C_i also satisfies KD45 like B_i.

K: $(C_i\phi \wedge C_i(\phi \Rightarrow \psi)) \Rightarrow C_i\psi$.
D: $\neg C_i false$.
4: $C_i\phi \Rightarrow C_iC_i\phi$.
5: $\neg C_i\phi \Rightarrow C_i\neg C_i\phi$.
G: From ϕ infer $B_i\phi$.

Furthermore, the conditional operator $\rightarrow_i$ satisfies the following axioms of nonmonotonic reasoning in [12,13].

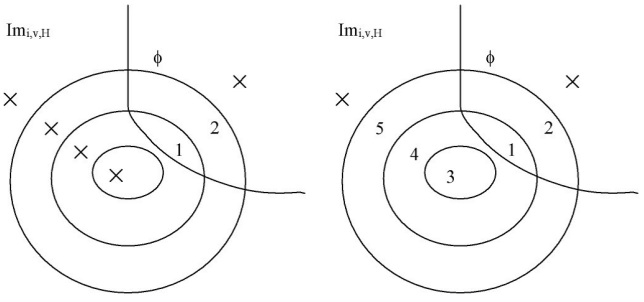

Fig. 1. Rewriting sphere by knowledge and belief

LLE: From $\phi \Leftrightarrow \phi'$ and $\phi \rightarrow_i \psi$ infer $\phi' \rightarrow_i \psi$.
RW: From $\psi \Rightarrow \psi'$ and $\phi \rightarrow_i \psi$ infer $\phi \rightarrow_i \psi'$.
REF: $\phi \rightarrow_i \phi$.
AND: From $\phi \rightarrow_i \psi_1$ and $\phi \rightarrow_i \psi_2$ infer $\phi \rightarrow_i \psi_1 \wedge \psi_2$.
OR: From $\phi_1 \rightarrow_i \psi$ and $\phi_2 \rightarrow_i \psi$ infer $\phi_1 \vee \phi_2 \rightarrow_i \psi$.
CM: From $\phi \rightarrow_i \psi_1$ and $\phi \rightarrow_i \psi_2$ infer $\phi \wedge \psi_1 \rightarrow_i \psi_2$.
RM: From $\phi \rightarrow_i \psi_1$ and not $\phi \rightarrow_i \neg\psi_2$ infer $\phi \wedge \psi_2 \rightarrow_i \psi_1$.

Finally, our formalization satisfies the following interaction between knowledge and beliefs characterized by [11].

KB1: $B_i\phi \Rightarrow K_iB_i\phi$.
KB2: $K_i\phi \Rightarrow B_i\phi$.

In the following discussion, we will suppose that if $w|_{At_i} = v|_{At_i}$, then $\leq_{i,w,\epsilon}=\leq_{i,v,\epsilon}$, although the above axioms are satisfied without the supposition.

3 Rewriting Rules for the Preference Relation

In this section we introduce our iterated belief revision operation by incorporating the other agent's knowledge (or correctly the information that the agent regards as knowledge) and belief. Although this is basically identical to traditional sphere semantics by [8,10], much of our idea depends on the radical/moderate approach to iterated belief revision by [16][4], and therefore, we can not only revise one agent's belief by one input, but also by a sequence of inputs from the other agent's knowledge and belief. Namely, we revise an agent's belief not only by an input but rather by a sequence of inputs. The change of belief corresponds to that of preference relation. Thus, we will define *rewriting rules for the preference relation*.

[4] Rott studied the conservative approach to iterated belief revision in [16]. In this approach, when $\neg\psi \in K\dot{+}\phi$, $(K\dot{+}\phi)\dot{+}\psi = K\dot{+}\psi$, where $\dot{+}$ is a revision function for a set K of formulae and a formula ϕ. The radical/moderate approach does not satisfy such a property. For details, see [16].

When we incorporate the information that states that j knows ϕ into i's belief, we consider that the worlds which satisfy $\neg\phi$ become implausible for agent i, because knowledge must be true. Perhaps, this information may be false due to j's misjudgment, when his information was acquired from an unreliable information source. However, we will consider that i relies on j's information if i can assume that he finds himself at some legal point and j has acquired the information in question from a reliable source. Thus, we use the radical approach for the definition of revision by another agent's knowledge. This approach is also presupposed by the study of belief revision in modal logic [5,6], where worlds that contradict the input are systematically deleted. Such a method is suitable for information that is known (i.e. guaranteed to be correct), because any situation that contradicts it should be eliminated from the set of worlds still considered possible.

The basic idea can be intuitively described by systems of spheres like the left side of figure 1. Spheres represent the various equivalence classes imposed on W by the preference relation $\leq_{i,v,H}$. Therefore, implausible worlds are at the outside of all the spheres. Thus, change of preference relation by knowledge is explained as follows: the most plausible worlds that satisfy ϕ (worlds of part 1 in the figure) become the most plausible worlds after accepting the knowledge that ϕ, the second plausible worlds that satisfy ϕ (worlds of part 2 in the figure) become the second-most plausible worlds in the next step, but implausible worlds that satisfy ϕ and any worlds that do not satisfy ϕ (worlds with black marks) become implausible worlds in the next step.

More formally, $w \leq_{i,v,H\cdot(m,j,\phi,K)} w'$ iff

I. $v, H \not\models_{\mathcal{M}} L$ or $(m,j) \notin E$ or $v, H \models_{\mathcal{M}} \neg K_j\phi$ or
II. $m \neq i$ and $w \leq_{i,v,H} w'$ or
III. $m = i$ and
  a. $w' \in Im_{i,v,H\cdot(i,j,\phi,K)}$ or
  b. $w \notin Im_{i,v,H\cdot(i,j,\phi,K)}$ and $w \leq_{i,v,H} w'$.

The meaning of the definition is as follows: w is at least as plausible as w' for i at (v,H) when the event (m,j,ϕ,K) occurs iff (I.) $(v, H\cdot(m,j,\phi,K))$ is already illegal (and therefore, all worlds are implausible from the point of view at $(v, H\cdot(m,j,\phi,K))$) or (II.) i cannot distinguish $(v, H\cdot(m,j,\phi,K))$ with (v,H) because of $m \neq i$ or (III.a.) w' is implausible at $(v, H\cdot(m,j,\phi,K))$ or (III.b.) w is at least as plausible as w' for i at the previous point (v,H). Thus, in the case of (I.) and (III.a.), any worlds become implausible, and in the case of (II.), the preference relation is not changed, and therefore only the case of (III.b.) requires belief revision by the knowledge.

We defined a revision operation for the preference relation by another agent's knowledge. We regard worlds that contradict the knowledge as implausible, when we incorporate the information that j believes ϕ into i's belief. However, nothing prevents worlds that satisfy $\neg\phi$ from becoming more plausible again upon learning new information as a result of future communications. Thus, we employ the moderate approach for the definition of revision by another agent's belief, because belief may be false and can not eliminate the information that is not known to be impossible.

The basic idea can also be characterized by systems of spheres like the right side of figure 1. Change of preference relation by belief is explained as follows: the most plausible worlds that satisfy ϕ (worlds of part 1 in the figure) become the most plausible

world in the next step of accepting the information that tells that ϕ is known, the second-most plausible worlds that satisfy ϕ (worlds of part 2 in the figure) become the second-most plausible worlds in the next step, the most plausible worlds that do not satisfy ϕ (worlds of part 3 in the figure) become the third-most plausible worlds in the next step, the second-most plausible worlds that do not satisfy ϕ (worlds of part 4) become the fourth-most plausible worlds, and the third-most plausible worlds that do not satisfy ϕ (worlds of part 5) become the fifth-most plausible worlds, but implausible worlds (marked black) remain implausible.

Again, $w \leq_{i,v,H\cdot(m,j,\phi,B)} w'$ iff

I. $v, H \not\models_{\mathcal{M}} L$ or $(m,j) \notin E$ or
II. $m \neq i$ and $w \leq_{i,v,H} w'$ or
III. $m = i$ and
  a. $w' \in Im_{i,v,H}$ or
  b. $w \notin Im_{i,v,H}$ and
    1. $w, H \models_{\mathcal{M}} \phi$, $w', H \models_{\mathcal{M}} \phi$, and $w \leq_{i,v,H} w'$, or
    2. $w, H \models_{\mathcal{M}} \phi$, $w', H \not\models_{\mathcal{M}} \phi$, or
    3. $w, H \not\models_{\mathcal{M}} \phi$, $w', H \not\models_{\mathcal{M}} \phi$, and $w \leq_{i,v,H} w'$.

The meaning of the definition is as follows: w is at least as plausible as w' for i at (v, H) when the event (m, j, ϕ, B) occurs iff (I.) $(v, H \cdot (m, j, \phi, B))$ is already not legal (and therefore, all worlds are implausible from the point of view at $(v, H \cdot (m, j, \phi, B)))$ or (II.) i cannot distinguish $(v, H \cdot (m, j, \phi, B))$ from (v, H) because of $m \neq i$ or (III.a.) w' is implausible at $(v, H \cdot (m, j, \phi, B))$ (i.e., at (v, H)) or (III.b.1.) both w and w' satisfy ϕ and w is at least as plausible as w' for i at the previous point (v, H) or (III.b.2.) w satisfies ϕ but w' does not satisfy ϕ or (III.b.3.) neither w nor w' satisfy ϕ and w is at least as plausible as w' for i at the previous point (v, H). Like the revision for knowledge, in the case of (I.) and (III.a.), any worlds become implausible, and in the case of (II.), the preference relation is not changed, and therefore, the case of (III.b.1.) - (III.b.3) is a standard condition for belief revision by knowledge.

While we assumed that $\leq_{i,w,\epsilon} = \leq_{i,v,\epsilon}$ when $w|_{At_i} = v|_{At_i}$, this assumption now can be generalized by the following theorem.

Theorem 1. *If* $w, H \models_{\mathcal{M}} L$ *and* $v, H' \models_{\mathcal{M}} L$ *and* $(w, H) \sim_i (v, H')$*, then* $\leq_{i,w,H} = \leq_{i,v,H'}$.

Proof. The proof is separated into two cases: (i) $w, H \cdot (m, j, \phi, e) \models_{\mathcal{M}} L$ and $m \neq i$ and $v, H' \models_{\mathcal{M}} L$ and $(w, H \cdot (m, j, \phi, e)) \sim (v, H')$, and (ii) $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} L$ and $v, H' \cdot (i, j, \phi, e) \models_{\mathcal{M}} L$ and $(w, H \cdot (i, j, \phi, e)) \sim (v, H' \cdot (i, j, \phi, e))$. Then, $w, H \models_{\mathcal{M}} L$ and $v, H' \models_{\mathcal{M}} L$ and $(w, H) \sim (v, H')$. In the case of (i), we can derive $w, H \models_{\mathcal{M}} L$ and $(w, H) \sim (v, H')$, and the conclusion that we want to show is proved from $\leq_{i,w,H} = \leq_{i,v,H'}$ by induction. In the case of (ii), after deriving $w, H \models_{\mathcal{M}} L$ and $v, H' \models_{\mathcal{M}} L$ and $(w, H) \sim (v, H')$, we will conclude the theorem from $\leq_{i,w,H} = \leq_{i,v,H'}$ and the definition of the rewriting rules for the preference relation.

Example 2. Suppose that $\mathcal{M} = \langle\{\{alice, bob, charlie\}, \{(alice, bob), (alice, charlie)\}\}, \langle\emptyset, \{p\}, \{q\}\rangle, \boldsymbol{\mathcal{P}}, W\rangle$, where $\boldsymbol{\mathcal{P}}$ is arbitrary except for some point (v, ϵ) for agent i. p

means that Dean puts on the red cap, and q means that Dean puts on the white shirt. We define the preference relation in (v, ϵ) for agent i as follows: let $v, w, x, y \in W$ such that $v(p) = 0$ and $v(q) = 0$, $w(p) = 0$ and $w(q) = 1$, $x(p) = 1$ and $x(q) = 0$, and $y(p) = 1$ and $y(q) = 1$, and suppose that v, w, x, and y are equivalent(i.e., $v \leq_{i,v,\epsilon} w$, $w \leq_{i,v,\epsilon} v$, $v \leq_{i,v,\epsilon} x$, $x \leq_{i,v,\epsilon} v$, and so on). Then, $v, (alice, bob, p \wedge q, B) \cdot (alice, charlie, \neg q, K) \models_{\mathcal{M}} \neg B_{alice} p$, but $v, (alice, bob, p, B) \cdot (alice, bob, q, B) \cdot (alice, charlie, \neg q, K) \models_{\mathcal{M}} B_{alice} p$.

4 Postulates

In this section, we will construct rational postulates for our revision operation that follow AGM's paradigm [1] and show the representation theorem in the restricted communication graph, which states that the above operation is essentially equivalent with the postulates. Although some of them are changed from AGM's original definition, we will discuss the reason why our result is restricted, and we conclude that the result is due to the fact that AGM's definition neglects the case that an external input comes from the other's knowledge.

First, we revisit AGM rational postulates for a belief revision operator $\dot{+}$ that accepts a belief set and a formula, and returns a belief set. Given some consequence relation $\vdash$, belief set K is a set of sentences such that $K = \{\phi | K \vdash \phi\}$, i.e., logically closed set. Moreover, suppose that $K + \phi = \{\psi | K \cup \{\phi\} \vdash \psi\}$.

AGM1. For any sentence ϕ and any belief set K, $K \dot{+} \phi$ is a belief set.
AGM2. $\phi \in K \dot{+} \phi$.
AGM3. $K \dot{+} \phi \subseteq K + \phi$.
AGM4. If $\neg\phi \notin K$, then $K + \phi \subseteq K \dot{+} \phi$.
AGM5. $K \dot{+} \phi = K_{\perp}$ iff $\vdash \neg\phi$.
AGM6. If $\vdash \phi \Leftrightarrow \psi$, then $K \dot{+} \phi = K \dot{+} \psi$.
AGM7. $K \dot{+} (\phi \wedge \psi) \subseteq (K \dot{+} \phi) + \psi$.
AGM8. If $\neg\psi \notin K \dot{+} \phi$, then $(K \dot{+} \phi) + \psi \subseteq K \dot{+} (\phi \wedge \psi)$.

As the details of AGM postulates are well explained in [1], we will not recapitulate them here. Instead, we will indicate a problem that arises when applying the AGM postulates to our formalism. That is, the beliefs in K are treated uniformly, with no distinction between truths and mere beliefs. Thus, an agent can not compare the knowledge that ϕ is implausible with the belief that ϕ is implausible. He cannot accept the information that ϕ is correct in the former case, while he can do so in the latter case. However, when we utilize the framework of belief set K, such a difference is neglected. In other words, an agent can not reject an input ϕ, even though he knows $\neg\phi$. For this reason we introduce a variation of the AGM postulates that addresses this problem.

1. If $(w, H) \sim_i (v, H')$, then $w, H \models_{\mathcal{M}} B_i\phi$ iff $v, H' \models_{\mathcal{M}} B_i\phi$.
2. If $w, H \models_{\mathcal{M}} \neg C_i \neg\phi$, then $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} B_i\phi$.
3. If $w, H \models_{\mathcal{M}} C_i \neg\phi$, then $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} B_i\psi$ iff $w, H \models_{\mathcal{M}} B_i\psi$.
4. If $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} B_i\psi$, then $v, H' \models_{\mathcal{M}} B_i(\phi \Rightarrow \psi)$.
5. If $w, H \models_{\mathcal{M}} \neg B_i \neg\phi$ and $w, H \models_{\mathcal{M}} B_i(\phi \Rightarrow \psi)$, then $v, H' \cdot (i, j, \phi, e) \models_{\mathcal{M}} B_i\psi$.

6. $w, H \models_{\mathcal{M}} \neg B_i L$ iff $w, H \models_{\mathcal{M}} \neg B_i \phi$ for any ϕ.
7. If $\models_{\mathcal{M}} \phi \Leftrightarrow \psi$, then $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} B_i \chi$ iff $w, H \cdot (i, j, \psi, e) \models_{\mathcal{M}} B_i \chi$.
8. If $w, H \cdot (i, j, \phi \wedge \psi, e) \models_{\mathcal{M}} B_i \chi$, then $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} B_i(\psi \Rightarrow \chi)$.
9. If $w, H \models_{\mathcal{M}} \neg C_i \neg \phi$, $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} \neg B_i \neg \psi$ and $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} B_i(\psi \Rightarrow \chi)$, then $w, H \cdot (i, j, \phi \wedge \psi, e) \models_{\mathcal{M}} B_i \chi$.
10.
    - 10.a. If $w, H \models_{\mathcal{M}} \neg C_i \neg \phi$, $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} \neg C_i \neg \psi$ and $\models_{\mathcal{M}} \neg(\phi \wedge \psi)$, then $w, H \cdot (i, j, \phi, e) \cdot (i, k, \psi, e') \models_{\mathcal{M}} B_i \chi$ iff $w, H \cdot (i, k, \psi, B) \models_{\mathcal{M}} B_i \chi$.
    - 10.b. Else if $w, H \models_{\mathcal{M}} \neg C_i \neg \phi$, $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} \neg C_i \neg \psi$ and $\not\models_{\mathcal{M}} \neg(\phi \wedge \psi)$, then $w, H \cdot (i, j, \phi, e) \cdot (i, k, \psi, e') \models_{\mathcal{M}} B_i \chi$ iff $w, H \cdot (i, k, \phi \wedge \psi, B) \models_{\mathcal{M}} B_i \chi$.
    - 10.c. Else if $w, H \models_{\mathcal{M}} C_i \neg \phi$ and $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} \neg C_i \neg \psi$, then $w, H \cdot (i, j, \phi, e) \cdot (i, k, \psi, e') \models_{\mathcal{M}} B_i \chi$ iff $w, H \cdot (i, k, \psi, B) \models_{\mathcal{M}} B_i \chi$.
    - 10.d. Else if $w, H \models_{\mathcal{M}} \neg C_i \neg \phi$ and $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} C_i \neg \psi$, then $w, H \cdot (i, j, \phi, e) \cdot (i, k, \psi, e') \models_{\mathcal{M}} B_i \chi$ iff $w, H \cdot (i, k, \phi, B) \models_{\mathcal{M}} B_i \chi$.
    - 10.e. Else if $w, H \models_{\mathcal{M}} C_i \neg \phi$ and $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} C_i \neg \psi$, $w, H \cdot (i, j, \phi, e) \cdot (i, k, \psi, e) \models_{\mathcal{M}} B_i \chi$ iff $w, H \models_{\mathcal{M}} B_i \chi$.

The following mapping acts as the translation between our postulates and the AGM ones.

$$\phi \in K \text{ iff } w, H \models_{\mathcal{M}} B_i \phi.$$

$$\psi \in K \dot{+} \phi \text{ iff } w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} B_i \psi.$$

Some of them (4., 5., 7. and 8.) are equivalent with AGM postulates (AGM3., 4., 6. and 7.)[5]. Postulates 2. and 9. are almost the same as AGM2. and AGM8., but they presuppose that i is not convinced that input ϕ is false, i.e., he considers that ϕ is possible. Otherwise, he neglects input ϕ by 3. Generally, belief revision that can reject external input is called *nonprioritized belief change* [9]. In our study, the reason for eliminating external input is the conviction that it is impossible. When histories cannot be distinguished by agent i, i's belief is also not distinguished in postulate 1.

Since our approach is the combination of radical and moderate approach, we introduce Postulate 10 for iterated belief revision. At first, suppose that information ϕ arrives at the agent i. When ϕ is inconsistent with i's knowledge, we should neglect the information ϕ from the result of the iterated belief revision (10.c and 10.e). After ϕ arrives at the agent i, suppose that ψ arrives at the agent i. When ψ is inconsistent with i's current knowledge, we should neglect this information ψ from the result of the iterated belief revision (10.d). Suppose that ψ is inconsistent with i's current knowledge. If the new information ψ is inconsistent with the old information ϕ, the new information eliminates the old information, and the iterated belief revision is equal to the belief revision by the new information (10.a). Otherwise, the iterated belief revision is equal to the belief revision by the conjunction of the new information and the old information (10.b).

Now we show that the following theorem holds when communication graph is restricted. The proof is shown in Appendix A.

[5] For 4., 5. and 8., note that $\psi \in K + \phi$ iff $\phi \Rightarrow \psi \in K$ by the deduction theorem.

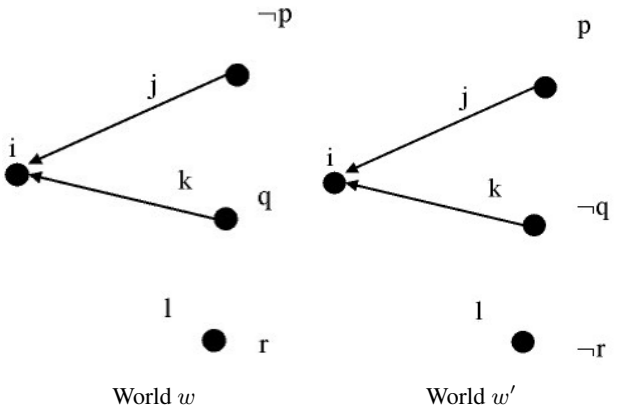

Fig. 2. Example that does not satisfy postulate 5

Theorem 2. *Suppose that E is fully connected. If a communication graph model $\mathcal{M}$ satisfies the rewriting rules for the preference relation, then it also satisfies all the postulates.*

Inconveniently, this theorem cannot be generalized to the unrestricted communication graph. Suppose that $\mathcal{M} = \langle\{\{i,j,k,l\},\{(i,j),(i,k)\}\},\langle\emptyset,\{p\},\{q\}\{r\}\rangle,\boldsymbol{\mathcal{P}},W\rangle$, where $\boldsymbol{\mathcal{P}}$ is arbitrary except for some point (v,ϵ) for agent i. We define the preference relation in (v,ϵ) for agent i as follows: let $w,w' \in W$ such that $w(p) = 0$, $w(q) = 1$, and $w(r) = 1$ (see the left side of figure 2), and $w'(p) = 1$, $w'(q) = 0$, and $w(r) = 0$ (see the right side of figure 2), and suppose that w is more plausible than w' (i.e., $w <_{i,v,\epsilon} w'$) and for any $w'' \in W$, if w'' is equal to neither w or w', then w' is more plausible than w'' (i.e., $w' <_{i,v,\epsilon} w''$). Then, $v,\epsilon \models_{\mathcal{M}} \neg B_i \neg(p \vee q)$ and $v,\epsilon \models_{\mathcal{M}} B_i((p \vee q) \Rightarrow r)$, but $v,(i,j,p \vee q,K) \models_{\mathcal{M}} \neg B_i r$. The communication graph in this example is not fully connected, and therefore, postulate 5 is violated.

We can show the converse of the above theorem when the communication graph is restricted. The proof is shown in Appendix B.

Theorem 3. *Suppose that E is existentially connected. If a communication graph model $\mathcal{M}$ satisfies all the postulates, then there is some model $\mathcal{M}'$ that satisfies the rewriting rules for the preference relation, such that for any w, H and ϕ, $w,H \models_{\mathcal{M}} \phi$ iff $w,H \models_{\mathcal{M}'} \phi$.*

That is, we showed the following representation theorem in the restricted communication graph.

Theorem 4. *Suppose that $\mathcal{A}$ has at least two agents and E is fully connected. There is some communication graph model $\mathcal{M}$ satisfies all the postulates iff there is some communication graph model $\mathcal{M}'$, which satisfies the rewriting rules for the preference relation, such that for any w, H and ϕ, $w,H \models_{\mathcal{M}} \phi$ iff $w,H \models_{\mathcal{M}'} \phi$.*

5 Conclusion

The traditional AGM paradigm of belief revision does not distinguish belief from knowledge in the way epistemic logic does. Even though AGM postulates are directly translated into the possible world semantics, each agent still cannot see if he lives in the actual world or not [8,10], i.e., there is no difference between belief and knowledge. The modal logic approach to belief revision had the same problem, because modal operator for knowledge was not introduced, which is distinguished from modal operator for belief [3].

As already discussed, Friedman and Halpern [5,6] defined a belief operator with the epistemic logic of multiagent systems in Fagin et al. [4]. They considered that an agent i believed ϕ iff i knew that ϕ was plausible. Thus, their approach to iterated belief revision can distinguish belief from knowledge.

Our study obviously depends on their result, because Pacuit and Parikh's history is equivalent with systems of runs in Fagin et al. [4] (See [14]). However, there is an important difference w.r.t. the acquisition of knowledge. They assumed that an agent acquires knowledge by observations from the external environment, and the information cannot be false. Our approach considers that an agent acquires knowledge from another agent by communication, and the information may be false. Therefore, our approach requires legality of history.

In this paper, we proposed a belief revision model for two types of uncertain communication. At first, when an agent as information source considers that he knows the information, and we can consider that there is reliable communication, we tend to accept this information. Secondly, when the agent that is the source of the information is himself less than certain, we are inclined to believe him, but at the same time we are less than fully convinced. Thus, postulates for the model was introduced, and the restricted representation theorem was shown.

However, there are four problems remaining with our study. Firstly, when we do not assume a fully connected communication graph, it may not satisfy the postulates. In the above example, we showed that an agent i may not believe ψ, when he believes that $\phi \Rightarrow \psi$, and j informs that he knows ϕ, although i believes that ϕ is not known to j, but can be true in fact. In the AGM postulates, such a case is forbidden. AGM postulates do not refer to the information source of external inputs. So when the first agent accepts a piece of information ϕ that comes from the second agent's knowledge, he will not distinguish it from other information ϕ received from a third agent's knowledge. However, these two situations can be regarded as different in our logic. Therefore, we do not agree with this idea, and consider that their postulates should be rethought for the case of multiagent communication, e.g., instead of our postulate 5., we use the following postulate.

- If $w, H \models_{\mathcal{M}} \neg B_i \neg K_j \phi$ and $w, H \models_{\mathcal{M}} B_i(\phi \Rightarrow \psi)$, then $v, H' \cdot (i, j, \phi, K) \models_{\mathcal{M}} B_i \psi$.
- If $w, H \models_{\mathcal{M}} \neg B_i \neg \phi$ and $w, H \models_{\mathcal{M}} B_i(\phi \Rightarrow \psi)$, then $v, H' \cdot (i, j, \phi, B) \models_{\mathcal{M}} B_i \psi$.

Our model satisfies this postulate generally. Construction of rational postulates for belief revision in multiagent communication will be our future subject.

Secondly, we do not consider legality of belief. Therefore, a communication event (i, j, ϕ, B) is not related with whether j believes ϕ actually or not. Since we consider that i can believe it even if it is j's lie, while i cannot know ϕ when j informs that j knows ϕ but it is false in fact, we did not define the legality of belief. In future work, we will study the problem of the representation of legality.

Thirdly, our definition of various concepts (e.g., the rewriting rules for the preference relation) is complicated. We suspect that this problem is due to the introduction of a legality predicate in the object language. Perhaps we may need to change the object language and lift the concept of history legality to the meta-language.

Finally, we depend on many presuppositions, e.g., whether an atomic proposition is known by someone or not is common knowledge, we use the moderate approach for the belief and the radical approach for the knowledge, etc. We do not reject such assumptions, since we are afraid of more complex formalization than this paper. We should resolve the problem for the simplicity.

References

1. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the logic of theory change: partial meet contraction and revision functions. Journal of Symbolic Logic 50, 510–530 (1985)
2. Audi, R.: Epistemology, 2nd edn. Routledge (2003)
3. Boutilier, C.: Unifying default reasoning and belief revision in a modal framework. Artificial Intelligence 68(1), 33–85 (1994)
4. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about knowledge. The MIT Press, Cambridge (1995)
5. Friedman, N., Halpern, J.Y.: Modeling belief in dynamic systems, part i: Foundations. Artificial Intelligence 95(2), 257–316 (1997)
6. Friedman, N., Halpern, J.Y.: Modeling belief in dynamic systems, part ii: Revision and update. Journal of Artificial Intelligence Research 10, 117–167 (1999)
7. Gärdenfors, P.: Knowledge in flux: Modeling the dynamics of epistemic states. The MIT Press, Cambridge (1988)
8. Grove, A.: Two modelings for theory change. Journal of Philosophical Logic 17, 157–170 (1988)
9. Hansson, S.O.: A textbook of belief dynamics. Kluwer Academic Publishers, Dordrecht (1999)
10. Katsuno, H., Mendelzon, A.: Propositional knowledge base revision and minimal change. Artificial Intelligence 52, 263–294 (1991)
11. Kraus, S., Lehman, D.: Knowledge, belief, and time. Theoretical Computer Science 58, 155–174 (1988)
12. Kraus, S., Lehman, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence 44, 167–207 (1990)
13. Lehman, D., Magidor, M.: What does a conditional knowledge base entail? Artificial Intelligence 55, 1–60 (1992)
14. Pacuit, E.: Some comments on history based structures. Journal of Applied Logic 5, 613–624 (2007)
15. Pacuit, E., Parikh, R.: Reasoning about communication graphs. In: Augustus de Morgan Workshop: Interactive Logic: Games and Social Software (2006)
16. Rott, H.: Coherence and conservatism in the dynamics of belief ii: Iterated belief change without dispositional coherence. Journal of Logic and Computation 13(1), 111–145 (2003)

A The Proof of Theorem 2

Proof. 1., 2., 3., 4., 6., 7., and 8. are rather easy to show. The problematic cases are 5., 9., and 10.

We show the case of 5. Let a communication graph model $\mathcal{M} = \langle \mathcal{G}, \boldsymbol{At}, \boldsymbol{\mathcal{P}}, W \rangle$ satisfy the rewriting rules for the preference relation. Suppose that $v, H' \models_{\mathcal{M}} \neg B_i \neg \phi$ and $v, H' \models_{\mathcal{M}} B_i(\phi \Rightarrow \psi)$. Then, there is some (w, H) such that $(w, H) \sim_i (v, H')$ and $w, H \models_{\mathcal{M}} L$, and for some $w' \in [\![\phi \Rightarrow \psi]\!]^{\mathcal{M}}_{i,w,H}$, for all $w'' \in [\![\neg(\phi \Rightarrow \psi)]\!]^{\mathcal{M}}_{i,w,H}$, $w' <_{i,w,H} w''$. Besides, for all $w' \in [\![\neg\phi]\!]^{\mathcal{M}}_{i,w,H}$, for some $w'' \in [\![\phi]\!]^{\mathcal{M}}_{i,w,H}$, $w'' \leq_{i,w,H} w'$. Thus, for some $w' \in [\![\phi \wedge \psi]\!]^{\mathcal{M}}_{i,w,H}$, for all $w'' \in [\![\phi \wedge \neg\psi]\!]^{\mathcal{M}}_{i,w,H}$, $w' <_{i,w,H} w''$. Therefore, for some $w' \in [\![\psi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,e)}$, for all $w'' \in [\![\neg\psi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,e)}$, $w' <_{i,w,H\cdot(i,j,\phi,e)} w''$. (In the case of $e = K$, note that $[\![\phi \wedge \psi]\!]^{\mathcal{M}}_{i,w,H} = [\![K_j\phi \wedge \psi]\!]^{\mathcal{M}}_{i,w,H}$ and $[\![\phi \wedge \neg\psi]\!]^{\mathcal{M}}_{i,w,H} = [\![K_j\phi \wedge \neg\psi]\!]^{\mathcal{M}}_{i,w,H}$.) Thus, we showed $v, H' \cdot (i, j, \phi, e) \models_{\mathcal{M}} Bi\psi$.

We show the case of 9. Suppose that $w, H \models_{\mathcal{M}} \neg C_i \neg\phi$, $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} \neg B_i \neg\psi$, $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} B_i(\psi \Rightarrow \chi)$. Then, for some $w' \in [\![\psi \Rightarrow \chi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,e)}$, for all $w'' \in [\![\neg(\psi \Rightarrow \chi)]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,e)}$, $w' <_{i,w,H\cdot(i,j,\phi,e)} w''$. Besides it, for some $w' \in [\![\phi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,e)}$, for all $w'' \in [\![\neg\phi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,e)}$, $w' <_{i,w,H\cdot(i,j,\phi,e)} w''$. Therefore, for some $w' \in [\![\phi \wedge (\psi \Rightarrow \chi)]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,e)}$, for all $w'' \in [\![\phi \wedge \neg(\psi \Rightarrow \chi)]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,e)}$, $w' <_{i,w,H\cdot(i,j,\phi,e)} w''$. Besides it, for all $w'' \in [\![\neg\psi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,e)}$, for some $w' \in [\![\psi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,e)}$, $w' \leq_{i,w,H\cdot(i,j,\phi,e)} w''$. It follows that for some $w' \in [\![\phi \wedge \psi \wedge \chi)]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,e)}$, for all $w'' \in [\![\phi \wedge \psi \wedge \neg\chi)]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,e)}$, $w' <_{i,w,H\cdot(i,j,\phi,e)} w''$. Thus, for some $w' \in [\![\phi \wedge \psi \wedge \chi)]\!]^{\mathcal{M}}_{i,w,H}$, for all $w'' \in [\![\phi \wedge \psi \wedge \neg\chi)]\!]^{\mathcal{M}}_{i,w,H}$, $w' <_{i,w,H} w''$. Therefore, for some $w' \in [\![\chi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi\wedge\psi,e)}$, for all $w'' \in [\![\neg\chi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi\wedge\psi,e)}$, $w' <_{i,w,H\cdot(i,j,\phi\wedge\psi,e)} w''$[6]. We showed $w, H \cdot (i, j, \phi \wedge \psi, e) \models_{\mathcal{M}} B_i\chi$.

About 10., We only show the case of 10.a. Suppose that $w, H \models_{\mathcal{M}} \neg C_i \neg\phi$, $w, H \cdot (i, j, \phi, e) \models_{\mathcal{M}} \neg C_i \neg\psi$ and $\models_{\mathcal{M}} \neg(\phi \wedge \psi)$. We want to show that $w, H \cdot (i, j, \phi, e) \cdot (i, k, \psi, e') \models_{\mathcal{M}} B_i\chi$ iff $w, H \cdot (i, k, \psi, B) \models_{\mathcal{M}} B_i\chi$. It suffices to show that $w, H \cdot (i, j, \phi, B) \cdot (i, k, \psi, e) \models_{\mathcal{M}} B_i\chi$ iff $v, H' \cdot (i, k, \psi, B) \models_{\mathcal{M}} B_i\chi$. It is obvious from the following point.

for some $w' \in [\![\chi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,B)\cdot(i,k,\psi,e)}$, for all $w'' \in [\![\neg\chi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,B)\cdot(i,k,\psi,e)}$,
$w' <_{i,w,H\cdot(i,j,\phi,B)\cdot(i,k,\psi,e)} w''$.
$\Updownarrow$
for some $w' \in [\![\psi \wedge \chi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,B)\cdot(i,k,\psi,e)}$, for all
$w'' \in [\![\psi \wedge \neg\chi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,B)\cdot(i,k,\psi,e)}$, $w' <_{i,w,H\cdot(i,j,\phi,B)\cdot(i,k,\psi,e)} w''$.
$\Updownarrow$
for some $w' \in [\![\psi \wedge \chi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,B)}$, for all $w'' \in [\![\psi \wedge \neg\chi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,j,\phi,B)}$,
$w' <_{i,w,H\cdot(i,j,\phi,B)} w''$.
$\Updownarrow$
for some $w' \in [\![\psi \wedge \chi]\!]^{\mathcal{M}}_{i,w,H}$, for all $w'' \in [\![\psi \wedge \neg\chi]\!]^{\mathcal{M}}_{i,w,H}$, $w' <_{i,w,H} w''$.

[6] In the case of $e = K$, note that $[\![\phi \wedge \psi \wedge \chi]\!]^{\mathcal{M}}_{i,w,H} = [\![K_j(\phi \wedge \psi) \wedge \chi]\!]^{\mathcal{M}}_{i,w,H}$ and $[\![\phi \wedge \psi \wedge \neg\chi]\!]^{\mathcal{M}}_{i,w,H} = [\![K_j(\phi \wedge \psi) \wedge \neg\chi]\!]^{\mathcal{M}}_{i,w,H}$.

$\Updownarrow$
for some $w' \in [\![\chi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,k,\psi,B)}$, for all $w'' \in [\![\neg\chi]\!]^{\mathcal{M}}_{i,w,H\cdot(i,k,\psi,B)}$, $w' <_{i,w,H\cdot(i,k,\psi,B)} w''$.

We can show 10.b. to 10.e. in the same way.

B The Proof of Theorem 3

Proof. Suppose $form(w) = \bigwedge\{p \in At | w(p) = 1\} \wedge \bigwedge\{\neg p \in At | w(p) = 0\}$. Given $\mathcal{M} = \langle \mathcal{G}, \boldsymbol{At}, \boldsymbol{\mathcal{P}}, W\rangle$, suppose $\mathcal{M}' = \langle \mathcal{G}, \boldsymbol{At}, \boldsymbol{\mathcal{P}'}, W\rangle$, where $\mathcal{P}'_i(v, H) = \leq_{i,v,H}$ such that

$$w \leq_{i,v,H} w' \text{ iff } v, H \not\models_{\mathcal{M}} L \text{ or } w' \in Im_{i,v,H}$$

$$\text{or } v, H \models_{\mathcal{M}} C_i\neg(form(w) \vee form(w'))$$

$$\text{or for some } j,$$

$$v, H \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w).$$

We will show that for any w, H, and ϕ, $w, H \models_{\mathcal{M}} \phi$ iff $w, H \models_{\mathcal{M}'} \phi$. It suffices to show that for any w, H, ϕ, and ψ, $w, H \models_{\mathcal{M}} \phi \rightarrow_i \psi$ iff $w, H \models_{\mathcal{M}'} \phi \rightarrow_i \psi$, when $w, H \models_{\mathcal{M}} L$. It is obvious as follows.

$v, H \models_{\mathcal{M}'} \phi \rightarrow_i \psi$
$\Updownarrow$
for some $w \in [\![\phi \wedge \psi]\!]^{\mathcal{M}'}_{i,v,H}$, for all $w \in [\![\phi \wedge \neg\psi]\!]^{\mathcal{M}'}_{i,v,H}$, $w <_{i,v,H} w'$.
$\Updownarrow$ (Note that E is existentially connected.)
for some $w \in [\![\phi \wedge \psi]\!]^{\mathcal{M}}_{i,v,H}$, for all $w \in [\![\phi \wedge \neg\psi]\!]^{\mathcal{M}}_{i,v,H}$, for some j,
$v, H \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} B_i form(w)$.
$\Updownarrow$ (Note that E is existentially connected.)
for some $w \in [\![\phi \wedge \psi]\!]^{\mathcal{M}}_{i,v,H}$, for all $w \in [\![\phi \wedge \neg\psi]\!]^{\mathcal{M}}_{i,v,H}$, $w <_{i,v,H} w'$.
$\Updownarrow$
$v, H \models_{\mathcal{M}} \phi \rightarrow_i \psi$

Moreover, connectedness, transitivity, satisfiability of the rewriting rules for the preference relations can be shown in the same way of the proof of the semantic version for belief revision [10] as follows.

(Connectedness) Suppose $w \not\leq_{i,v,H} w'$. We want to show that $w' \leq_{i,v,H} w$. Then, $v, H \models_{\mathcal{M}} L$ and $w' \notin Im_{i,v,H}$ and $v, H \models_{\mathcal{M}} \neg C_i\neg(form(w) \vee form(w'))$ and $v, H \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} B_i\neg form(w)$. From Postulate 2, $v, H \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} B_i(form(w) \vee form(w'))$. Since B_i satisfies axiom K, $v, H \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} B_i form(w')$. Therefore, $v, H \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i\neg form(w')$.

(Transitivity) Suppose that $w \leq_{i,v,H} w'$ and $w' \leq_{i,v,H} w''$. We want to show that $w \leq_{i,v,H} w''$. From the supposition, $v, H \not\models_{\mathcal{M}} L$ or $w' \in Im_{i,v,H}$ or $v, H \models_{\mathcal{M}} C_i\neg(form(w)\vee form(w'))$ or $v, H\cdot(i, j, form(w)\vee form(w'), B) \models_{\mathcal{M}} \neg B_i\neg form(w)$. Besides it, $w'' \in Im_{i,v,H}$ or $v, H \models_{\mathcal{M}} C_i\neg(form(w') \vee form(w''))$ or $v, H \cdot (i, j, form(w') \vee form(w''), B) \models_{\mathcal{M}} \neg B_i\neg form(w')$. Suppose that $v, H \models_{\mathcal{M}} L$

and $w' \notin Im_{i,v,H}$ and $v, H \models_{\mathcal{M}} \neg K_i \neg(form(w) \vee form(w''))$. We want to show that $v, H \cdot (i, j, form(w') \vee form(w''), B) \models_{\mathcal{M}} \neg B_i \neg form(w')$. Since $v, H \models_{\mathcal{M}} \neg K_i \neg(form(w) \vee form(w''))$, we can conclude $v, H \models_{\mathcal{M}} \neg C_i \neg(form(w) \vee form(w') \vee form(w''))$. From Postulates 2, $v, H \cdot (i, j, form(w) \vee form(w') \vee form(w''), B) \models_{\mathcal{M}} B_i(form(w) \vee form(w') \vee form(w''))$.

We will show $v, H \not\models_{\mathcal{M}} C_i \neg(form(w) \vee form(w'))$. Suppose that $v, H \models_{\mathcal{M}} C_i \neg(form(w) \vee form(w'))$. Then, $v, H \cdot (i, j, form(w) \vee form(w') \vee form(w''), B) \models_{\mathcal{M}} B_i \neg(form(w) \vee form(w'))$. From the axiom K, $v, H \cdot (i, j, form(w) \vee form(w') \vee form(w''), B) \models_{\mathcal{M}} B_i form(w'')$. It follows that $v, H \models_{\mathcal{M}} \neg C_i \neg(form(w') \vee form(w''))$. From the suposition, $v, H \cdot (i, j, form(w') \vee form(w''), B) \models_{\mathcal{M}} \neg B_i \neg form(w')$. It contradicts the supposition. Therefore, $v, H \not\models_{\mathcal{M}} C_i \neg(form(w) \vee form(w'))$.

So, it suffices to show the result in the case of (i) $v, H \models_{\mathcal{M}} C_i \neg(form(w') \vee form(w''))$ or (ii) $v, H \models_{\mathcal{M}} \neg C_i \neg(form(w') \vee form(w''))$.

(i) Suppose that $v, H \models_{\mathcal{M}} C_i \neg(form(w') \vee form(w''))$. Then, $v, H \cdot (i, j, form(w) \vee form(w''), B) \models_{\mathcal{M}} B_i \neg form(w'')$. From the axiom K, $v, H \cdot (i, j, form(w) \vee form(w''), B) \models_{\mathcal{M}} B_i form(w)$. It follows that $v, H \cdot (i, j, form(w) \vee form(w''), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$.

(ii) Suppose that $v, H \models_{\mathcal{M}} \neg C_i \neg(form(w') \vee form(w''))$. Then, $v, H \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$ and $v, H \cdot (i, j, form(w') \vee form(w''), B) \models_{\mathcal{M}} \neg B_i \neg form(w')$. From Postulate 9, $v, H \cdot (i, j, form(w) \vee form(w') \vee form(w''), B) \models_{\mathcal{M}} \neg B_i((form(w) \vee form(w')) \Rightarrow \neg form(w))$. Therefore, $v, H \cdot (i, j, form(w) \vee form(w') \vee form(w''), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$. Since $v, H \cdot (i, j, form(w) \vee form(w') \vee form(w''), B) \models_{\mathcal{M}} \neg B_i((form(w) \vee form(w'')) \Rightarrow \neg form(w))$, it follows from Postualte 8 that $v, H \cdot (i, j, form(w) \vee form(w''), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$.

(Condition B $\Rightarrow$) Suppose that $w \leq_{i,v,H \cdot (m,j,\phi,B)} w'$. We want to show that

I. $v, H \not\models_{\mathcal{M}} L$ or $(m, j) \notin E$ or
II. $m \neq i$ and $w \leq_{i,v,H} w'$ or
III. $m = i$ and
  a. $w' \in Im_{i,v,H}$ or
  b. $w \notin Im_{i,v,H}$ and
    1. $w, H \models_{\mathcal{M}} \phi$, $w', H \models_{\mathcal{M}} \phi$, and $w \leq_{i,v,H} w'$, or
    2. $w, H \models_{\mathcal{M}} \phi$, $w', H \not\models_{\mathcal{M}} \phi$, or
    3. $w, H \not\models_{\mathcal{M}} \phi$, $w', H \not\models_{\mathcal{M}} \phi$, and $w \leq_{i,v,H} w'$,

We can conclude from $w \leq_{i,v,H \cdot (m,j,\phi,B)} w'$ that $v, H \cdot (m, j, \phi, K) \not\models_{\mathcal{M}} L$ or $w' \in Im_{i,v,H}$ or $v, H \cdot \models_{\mathcal{M}} C_i \neg(form(w) \vee form(w'))$ or $v, H \cdot (m, j, \phi, B) \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$. When $v, H \not\models_{\mathcal{M}} L$ or $w' \in Im_{i,v,H}$, I. or III.a. is obvious. When $m \neq i$, $w \leq_{i,v,H} w'$ is obvious from Postulate 1. Therefore, II. is satisfied. Suppose $v, H \models_{\mathcal{M}} L$ and $w' \notin Im_{i,v,H}$ and $m = i$. We can conclude $v, H \models_{\mathcal{M}} \neg C_i \neg(form(w) \vee form(w'))$ from $w' \notin Im_{i,v,H}$. Thus, $v, H \cdot (m, j, \phi, B) \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$. $w \notin Im_{i,v,H}$ is obvious.

(i) In the case of $v, H \models_{\mathcal{M}} C_i \neg\phi$, we can deduce $v, H \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$ from 10.c. Therefore, III.b.3. is satisfied.

(ii) In the case of $v, H \models_{\mathcal{M}} \neg C_i \neg \phi$ and $\models_{\mathcal{M}} \neg(\phi \wedge (form(w) \vee form(w')))$, we can deduce $v, H \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$ from 10.a. Therefore, III.b.3 is satisfied.

(iii) In the case of $v, H \models_{\mathcal{M}} \neg C_i \neg \phi$ and $\not\models_{\mathcal{M}} \neg(\phi \wedge (form(w) \vee form(w')))$, we can deduce $v, H \cdot (i, j, \phi \wedge (form(w) \vee form(w'), B)) \models_{\mathcal{M}} \neg B_i \neg form(w)$ from 10.b. When $w, H \models_{\mathcal{M}} \neg\phi$ and $w', H \models_{\mathcal{M}} \phi$, $v, H \cdot (i, j, form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$ from 7., but it contradicts $v, H \cdot (i, j, form(w'), B) \models_{\mathcal{M}} B_i \neg form(w)$. When $w, H \models_{\mathcal{M}} \phi$ and $w', H \models_{\mathcal{M}} \neg\phi$, III.b.2. is satisfied. When $w, H \models_{\mathcal{M}} \phi$ and $w', H \models_{\mathcal{M}} \phi$, $v, H \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$ from 7., and III.b.1. is satisfied.

(Condition B $\Rightarrow$) Suppose that

I. $v, H \not\models_{\mathcal{M}} L$ or $(m, j) \notin E$ or
II. $m \neq i$ and $w \leq_{i,v,H} w'$ or
III. $m = i$ and
  a. $w' \in Im_{i,v,H}$ or
  b. $w \notin Im_{i,v,H}$ and
    1. $w, H \models_{\mathcal{M}} \phi$, $w', H \models_{\mathcal{M}} \phi$, and $w \leq_{i,v,H} w'$, or
    2. $w, H \models_{\mathcal{M}} \phi$, $w', H \not\models_{\mathcal{M}} \phi$, or
    3. $w, H \not\models_{\mathcal{M}} \phi$, $w', H \not\models_{\mathcal{M}} \phi$, and $w \leq_{i,v,H} w'$,

We want to show that $w \leq_{i,v,H\cdot(m,j,\phi,B)} w'$. When I. or III.a. is satisfied, it is obvious. When II. is satisfied, it is followed from Postulate 1. Suppose that I.-III.a. is not satisfied.

(i) In the case of III.b.1., $w \notin Im_{i,v,H}$, $w, H \models_{\mathcal{M}} \phi$, $w', H \models_{\mathcal{M}} \phi$, and $v, H \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$. From $w \notin Im_{i,v,H}$, $v, H \models_{\mathcal{M}} \neg C_i \neg\phi$. It folows from Postulate 7. and 10.b. that $v, H \cdot (i, j, \phi, B) \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$. Therefore, $w \leq_{i,v,H\cdot(m,j,\phi,B)} w'$.

(ii) In the case of III.b.2., $w \notin Im_{i,v,H}$, $w, H \models_{\mathcal{M}} \phi$, $w', H \not\models_{\mathcal{M}} \phi$. From Postulate 2., $v, H \cdot (i, j, form(w), B) \models_{\mathcal{M}} B_i form(w)$. From Postulate 7., $v, H \cdot (i, j, \phi \wedge (form(w) \vee form(w')), B) \models_{\mathcal{M}} B_i form(w)$. From Postulate 10.a., $v, H \cdot (i, j, \phi, B) \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} B_i form(w)$. Thus, $v, H \cdot (i, j, \phi, B) \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$. Therefore, $w \leq_{i,v,H\cdot(m,j,\phi,B)} w'$.

(iii) In the case of III.b.3., $w \notin Im_{i,v,H}$, $w, H \models_{\mathcal{M}} \neg\phi$, $w', H \models_{\mathcal{M}} \neg\phi$, and $v, H \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$. From $w \notin Im_{i,v,H}$, $v, H \models_{\mathcal{M}} \neg K_i \neg\phi$. It folows from Postulate 10.a. that $v, H \cdot (i, j, \phi, B) \cdot (i, j, form(w) \vee form(w'), B) \models_{\mathcal{M}} \neg B_i \neg form(w)$. Therefore, $w \leq_{i,v,H\cdot(m,j,\phi,B)} w'$.

The proof of (Condition K $\Rightarrow$) and (Condition K $\Leftarrow$) is almost as same as the proof of (Condition K $\Rightarrow$) and (Condition K $\Leftarrow$).

Abstracting and Verifying Strategy-Proofness for Auction Mechanisms

Emmanuel M. Tadjouddine, Frank Guerin, and Wamberto Vasconcelos

Department of Computing Science, King's College,
University of Aberdeen, Aberdeen AB24 3UE, Scotland

Abstract. We are interested in finding algorithms which will allow an agent roaming between different electronic auction institutions to automatically verify the game-theoretic properties of a previously unseen auction protocol. A property may be that the protocol is robust to collusion or deception or that a given strategy is optimal. Model checking provides an automatic way of carrying out such proofs. However it may suffer from state space explosion for large models. To improve the performance of model checking, abstractions were used along with the SPIN model checker. We considered two case studies: the Vickrey auction and a tractable combinatorial auction. Numerical results showed the limits of relying solely on SPIN. To reduce the state space required by SPIN, two property-preserving abstraction methods were applied: the first is the classical program slicing technique, which removes irrelevant variables with respect to the property; the second replaces large data, possibly infinite values of variables with smaller abstract values. This enabled us to model check the strategy-proofness property of the Vickrey auction for unbounded bid range and number of agents.

1 Introduction

Trust is a major concern in agent-mediated eCommerce systems. To tackle this, much research has been carried out to develop game theory mechanisms which guarantee desirable properties for the system, even in the face of agents who are willing to lie or cheat; for example, there are mechanisms which can guarantee that a system is robust to agents bidding falsely or colluding. These mechanisms work perfectly well in a closed Multi-Agent System (MAS) when designers can program agents in full knowledge of the favourable properties of the mechanism. However, it is not clear how such mechanisms could be used in open systems where agents might have to interoperate between different institutions. A roaming agent arriving at an institution where a new, previously unseen, protocol is in use, will need to understand the rules of engagement in much the same way human agents can.

In this paper we assume that there is some standard language in which the rules of the auction can be written and published. By this we mean a low-level protocol specifying who can bid, in what order, and how the winners and prices are determined. We assume that institutions publish the specifications of

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 197–214, 2009.

their auction protocols in this way. A roaming agent who arrives at a foreign institution can download a protocol and analyse it in order to make a decision about whether or not to participate, and what strategy to use. The challenge now is for the roaming agent, with bounded computing resources, to be able to automatically check some of the game-theoretic properties of the protocol.

Auctions are usually designed to have certain desirable game theoretic properties such as *incentive compatibility*, encouraging agents to bid truthfully, or *collusion-proofness* meaning agents cannot collude to achieve a certain outcome, or *false-name bidding free* meaning agents cannot manipulate the outcome by using fictitious names. We focus on dominant strategy equilibrium, which means the game has the property of *strategy-proofness*; this gives agents an incentive to bid their true valuations. Game theoretic properties such as strategy-proofness rely on very strong assumptions; it is required that the property be common knowledge among the players. If the common knowledge of the equilibrium is not achieved, then agents cannot expect it to be played [1].

Model checking provides an automatic way of carrying out the verification of game-theoretic properties of a given auction mechanism. However it may suffer from state space explosion for large models [2,3]. This work extends that of [3] from model-checking single item auctions to combinatorial ones. In here, we have considered two case studies: the Vickrey auction and a tractable combinatorial auction termed Quantity-Restrained Multi-Object Auctions (QRMOA) in [4] and then checked their strategy-proofness using the SPIN model checker [5]. Numerical results showed the limits of relying solely on SPIN. To reduce the state space required by SPIN, two property-preserving abstraction methods were applied: the first is the classical program slicing technique [6], which removes irrelevant variables with respect to the property; the second replaces large data, possibly infinite values of variables with smaller abstract values. This enabled us to model check the strategy-proofness property of the Vickrey auction for an unbounded bid range and number of agents.

The remainder of this paper is organized as follows. In Section 2, we present some preliminaries with respect to mechanism design and combinatorial auctions. In Section 3, we use the Vickrey auction and the QRMOA to demonstrate the limits of relying solely on a model checker to verify for example the strategy-proofness property. In Section 4, we present two sound abstractions for improving the performance of model checking and illustrate their benefits in checking strategy-proofness for the Vickrey auction. In Section 5, we discuss the related work. Finally, Section 6 concludes and opens up with future work.

2 Preliminaries

This section provides the required background on combinatorial auctions and mechanism design.

Mechanism Design: Game theory *mechanism design* [7, Chap. 1 & 2], consists in finding decision procedures that determine the outcome for the game mechanism according to some desired objective. In open multi-agent systems where

auctions take place, an objective that is of interest is achieving *incentive compatibility* (no bidder can benefit from lying provided all other agents are truthful) or *strategy-proofness* (truth-telling is a dominant strategy). A Nash equilibrium can be used to implement an incentive compatible mechanism. A dominant strategy equilibrium can implement strategy-proofness, this is a stronger notion than Nash equilibrium; i.e. every dominant strategy equilibrium is a Nash equilibrium, but the converse is not true. A well known class of auction mechanisms that is efficient and strategy-proof is the Vickrey-Clarke-Groves (VCG), see for example [8,7,9]. The VCG mechanism is performed by finding (i) the allocation that maximises the social welfare and (ii) a pricing rule allowing each winner to benefit from a discount according to his contribution to the overall value for the auction. To formalise the VCG mechanism, let us introduce the following notations:

- $\mathcal{X}$ is the set of possible valid allocations
- $v_i(x)$ is the true valuation of $x \in \mathcal{X}$ for bidder i
- $b_i(x)$ is the bidding value of $x \in \mathcal{X}$ for bidder i
- $x^* \in \text{argmax}_{x \in \mathcal{X}} \sum_{i=1}^{n} b_i(x)$ is the optimal allocation for the submitted bids.
- $x^*_{-i} \in \text{argmax}_{x \in \mathcal{X}} \sum_{j \neq i}^{n} b_j(x)$ is the optimal allocation if agent i were not to bid.
- u_i is the utility function for bidder i.

The VCG payment p_i for bidder i is defined as

$$\begin{aligned} p_i(b_i, b_{-i}) &= b_i(x^*) - \left(\textstyle\sum_{j=1}^{n} b_j(x^*) - \sum_{j=1,\ j\neq i}^{n} b_j(x^*_{-i})\right) \\ &= \textstyle\sum_{j=1,\ j\neq i}^{n} b_j(x^*_{-i}) - \sum_{j=1,\ j\neq i}^{n} b_j(x^*). \end{aligned} \tag{1}$$

In this equation, the quantity $\sum_{j=1,\ j\neq i}^{n} b_j(x^*_{-i})$ is called the *Clarke tax*. The utility u_i for agent i is simply its valuation on the received bundle minus the price it has paid. The price paid is a function of its own bid, and the bids of opponents. Therefore we can define the utility u_i as a quasi-linear function of three variables: agent i's valuation v_i for the received bundle and its own bid b_i, and the bids of the opponents b_{-i}.

$$u_i(v_i, b_i, b_{-i}) = v_i - p_i(b_i, b_{-i}). \tag{2}$$

The mechanism is strategy-proof if and only if the following property holds:

$$\forall i,\ \forall v_i,\ \forall b_i,\ \forall b_{-i},\ u_i(v_i, v_i, b_{-i}) \geq u_i(v_i, b_i, b_{-i}). \tag{3}$$

This is a stronger property than Nash equilibrium. Nash equilibrium states that for some given strategy profile b^*

$$\forall i,\ \forall v_i,\ \forall b_i,\ u_i(v_i, b^*_i, b^*_{-i}) \geq u_i(v_i, b_i, b^*_{-i}) \tag{4}$$

Any strategy-proof mechanism has a Nash equilibrium with $b^*_i = v_i$ for all i.

Combinatorial Auctions: In a *combinatorial auction*, there is a set M of m items to be sold to a set N of n potential buyers. A bid is formulated as a pair $(B(x), b(x))$ in which $B(x) \subseteq M$ is a bundle of items and $b(x) \in \mathbb{R}^+$ is the price offer for the items in B. We are interested in the *XOR formulation* of the combinatorial auction problem (CAP). This means that each bidder submits a set of bids, and a maximum of one bid can be accepted from each bidder. Formally each bidder $k \in N$ submits a set $\mathcal{B}_k$ of l_k bids, $\mathcal{B}_k = \{(B(x_1), b(x_1)), (B(x_2), b(x_2)), \ldots, (B(x_{l_k}), b(x_{l_k}))\}$. The auctioneer's problem is to find a set $X_0 \subset \bigcup_{k \in N} \mathcal{B}_k$ of bids such that $\sum_{x \in X_0} b(x)$ is maximal, subject to the constraints that

1. For all $x_i, x_j \in X_0 :\ B(x_i) \cap B(x_j) = \emptyset$ meaning that an item can be found in at most one accepted bid.
2. For all $x_i, x_j \in X_0, k \in N :\ \{x_i, x_j\} \nsubseteq \mathcal{B}_k$ meaning that there are never two bids accepted from the same bidder (i.e. one is the maximum).

We assume that items may remain unallocated at the end of the auction. The CAP is NP-complete [10]. In here, we focus on a very simple type of combinatorial auction: the Quantity-Restricted Multi-Object Auction (QRMOA), which we describe in Section 3.2. The QRMOA is tractable [4] and therefore can be solved efficiently using graph matching algorithms [11]. This means the incentive compatibility or strategy-proofness of the QRMOA should be ensured since we have an exact winner determination algorithm instead of using approximations for the general CAP.

3 Verifying Properties by Model Checking

In previous work [12], we have detailed the use of the ALLOY [13] model checker based on first-order relational logic to verify strategy-proofness for auctions. The game mechanism and the property are expressed as first-order logic formula and the property is checked for a finite scope. This approach is elegant but does not scale up very well. In this work, we used SPIN [5] on the premiss that it can be combined with powerful abstractions in order to check large-scale models. We have considered a simple Vickrey auction, and a tractable instance of the CAP (the QRMOA) to which we applied the VCG mechanism.

We formally specified the auctions using the Promela process modelling language [5]; this required us to code both the optimal WDA (winner determination algorithm) and the pricing rule according to a VCG implementation. We then verified some game-theoretic properties of both auctions using the SPIN model checker. These properties are expressed as Promela assertions in a model parameterized by the range of the bids, the number of agents and the number of items. In other words, we consider a finite set A of actions (bidding values) that can be used by each agent, two numbers n of agents and m of items for sale; all these parameters can be varied in order to check small as well as large models. We have carried out the checking from the viewpoint of the participants. For a given agent i, we evaluate its utility u_i^* when bidding its true valuation and its

utility u_i otherwise; and check the assertion $u_i^* \geq u_i$ in all the possible game configurations specified by the property.

We illustrate this verification scheme by using the Vickrey auction as shown in Figure 1 wherein the strategy-proofness property is verified from the viewpoint of an agent, say Agent 1. In Figure 1, we are given the number of agents n, a bid range representing integer values between 0 and `upbound` and a vector v of the n agents' valuations of the item in sale. The procedure `vickrey` implements the winner determination algorithm for the Vickrey auction. It takes as inputs the number of agents `n`, a vector of n agents' valuations `v`, and a vector of n agents' bids `x`; then it determines the winner of the item and returns a vector of n utilities `u`. The verification procedure basically scans all possible configurations and for each one, it uses a Promela assertion to check that the utility of Agent 1 bidding its true valuation is greater or equal to that obtained by bidding any other value regardless to what its opponents bid.

```
wloop:
do
 :: !finished ->
   vickrey(n,v,x,ul); /* Agent 1 lies; it bids x[0] */
   tmp = x[0];
   x[0] = v[0];
   vickrey(n,v,x,ut); /* Agent 1 bids its true valuation */
   x[0] = tmp;
   assert(ul[0] <= ut[0]);
  floop: { j = 0; /* generates all bid profiles between 0..upbound */
  body: {
  if
  ::(x[j] < upbound) ->
   x[j] = x[j]+1;
   goto wloop; /* break the for loop */
  :: else -> if
     :: (j!=n-1) -> x[j] = 0;
      j=j+1; goto body;
     ::else -> finished = true;
     fi
   fi
  }
 }
 :: else -> printf("ul[0] = %d, ut[0] = %d", ul[0], ut[0]);
  break;
od
```

Fig. 1. Promela code verifying strategy-proofness for Vickrey auction

For the strategy-proofness property in equation (3), the number of possible configurations represents all possible strategy profiles of the participants. This is exponential in the number of agents and items. Consequently, the checking for the strategy-proofness property is computationally expensive. If the assertion $u_i^* \geq u_i$ is not violated, then the property holds for the specified model. Notice that this does not imply the property holds independently from the bid range or the numbers of agents and items. Moreover, if the property does not hold, then SPIN will display a counter-example.

Table 1. Statistics for the strategy-proofness property in a Vickrey Auction (bids from 0 to 1000)

	Number of players				
	100	300	500	600	700
Memory(Mb)	146.72	795.92	904.20	1083.30	–
CPU Time(s)	1.20	6.44	7.20	8.29	–

Tables 1 and 2 show runtime statistics for the checking of the strategy-proofness property. These results were obtained by compiling and running the models produced by SPIN on a PC Pentium Dual Processor 2.99 GHz with 2GB RAM, running Windows XP. We used the options `-DBITSTATE -DVETORSZ` $=m$ where m is the size of the state vector chosen according to the size of the model at hand. As explained in [5, p. 206], the option `-DBITSTATE` triggers the use of an algorithm allowing the checker to visit every reachable state of the transition system at most once. For efficiency reasons, the states are stored in a hashtable. This turns a rather exponential search into a linear one, thus enhancing the performance of the checking procedure.

3.1 The Vickrey Auction Example

In a Vickrey auction, n agents bid for a single item. Each agent has a private valuation v of the item. The highest bidder wins the item but pays the second highest bid p, getting the utility $u = v - p$. A losing bidder pays nothing and has a zero utility. Table 1 shows the results obtained by fixing the bid range and varying the number of agents. We observe that beyond a certain number of agents, the amount of the memory required by SPIN explodes provoking a 'ran out of memory' error. However, this mechanism is simple enough to carry out a full space checking for up to 600 agents.

3.2 The QRMOA Example

The combinatorial auction termed as QRMOA in [4] can be described as follows. Bidders place individual price offers for a number m of items, but will only accept a restricted quantity $q < m$. A buyer will pay nothing for any item assigned to him beyond the quantity q. This restricted type of combinatorial auction allows a concise representation for bids. In this setting, a bid is a tuple of the form $(i_1, p_1, i_2, p_2, \ldots, i_m, p_m, q)$ where each p_j is a price offer for object i_j and q is the maximum number of objects to be assigned to this bid.

This CAP is tractable in the following sense. An instance of the QRMOA can be transformed to an instance of the assignment problem, and solved using the Hungarian algorithm [14]. This is a 1-matching algorithm, which works for a square matrix. To solve the QRMOA problem using the assignment algorithm, we duplicate q times a given bid with a constraint q. Given n bids and m items for sale, we construct a cost matrix $\mathbf{C}$ with m columns and n rows. The entry $\mathbf{C}_{ij}$ is the value which the ith bid places for the jth item. Fictitious bids or

Table 2. Statistics for the strategy-proofness property in a QRMOA with two items and two items: Influence of the bid range (left) and number of players (right)

	Bid range			
	0..5	0..10	0..15	0..16
Memory (Mb)	48.10	158.90	1164.60	–
CPU Time (s)	40.60	42.70	67.40	–

	Number of players		
	2	3	4
Memory (Mb)	39.10	1157.10	–
CPU Time (s)	36.27	40.75	–

items may be added to render the matrix $\mathbf{C}$ square. If all q values are 1 then we can immediately solve for the optimal allocation via the assignment algorithm. If any row corresponds to a bid with a q-value greater than one then we simply duplicate that row so that it occurs q times. Now it can be assigned up to q items. After applying this procedure for all such bids, we have a matrix with $k \geq n$ rows; k is the sum of all constraints q in the submitted bids. The assignment algorithm computes an optimal allocation in $\mathcal{O}(k^3)$ time complexity. Different matching algorithms can be used to solve this problem, see [11] for an example with 'better' time complexity. In here, we rather focus on model-checking combinatorial auctions that have exact solutions and the QRMOA solved by the tractable assignment algorithm is a good case study.

For the pricing rule, we used the VCG mechanism. This implies using the above exact WDA to determine the prices and resulting utilities for the agents. The Promela implementation of the entire mechanism consists of over six hundred lines of code. The timing results obtained by SPIN are presented in Table 2. The results show the checking can be completed for very limited models of the QRMOA. For example, model-checking strategy-proofness of a QRMOA composed of four agents and two items failed to complete because of explosion of memory requirement.

Moreover, we observe that increasing the number of items has the same effect as increasing the number of players since the strategy space in the QRMOA is A^{nm} wherein n, m are the numbers of players and items respectively and A, the set of actions (bid range) for each player. These results show that the size of the model must be controlled in order to avoid an explosion of memory requirement. To this end, we use abstraction techniques to build a less complex model in which the property to be proved is preserved. We assume that in the combinatorial auction, the number of items is fixed but the number of players is unbounded. Then, we focus on building up an abstraction of the exponential data domain A^n since this will allow us to completely model check at least single item auctions such as the Vickrey auction.

4 Abstract Model Checking

In this section, we present a way of combining model checking with abstract interpretation, enabling us to reduce the complexity of model checking by using an appropriate choice of abstraction for our auctions.

The SPIN model checker allows us to show that for a given model a game-theoretic property holds. Unfortunately, the computational costs of this exercise grow exponentially, thus prohibiting us from giving a definite answer for large models as it is not possible to explore the entire search space. In such scenarios, model checking will never produce false negatives but it may produce false positives.

In order to reduce the costs involved in our checks, we propose to use *abstract interpretation* to simplify the problem. Abstract interpretation [15,16] provides a general theory for approximating the semantics of computer programs allowing the analysis of possible/potential computations without actually executing programs. We shall use abstract interpretation to find a property-preserving approximation of the concrete domain, our effort consisting of the following:

1. Defining a suitable abstraction that maps the concrete domain (data objects and associated operations) and properties onto their abstract counterparts,
2. Performing the checking using the abstract model, and
3. Deciding if the property holds in the concrete model.

It is worth noting that abstract interpretation never produces false positives but may raise false negatives due to the use of approximations.

4.1 Definitions

An abstraction provides a mapping of the original (concrete) domain (and associated search space) onto a less complex (abstract) domain, enabling us to eliminate irrelevant details. Since we represent games as computer programs, our search space is the *state domain*, that is, the execution state of the program containing the values of all its variables and the current point of the execution flow.

Definition 1. *A finite game is a transition system* $\Sigma = \langle N, A, S, \theta, \rho, u \rangle$ *where*

- N, A, S *are non-empty sets of agents, strategies, and states respectively;*
- $\theta : S \rightarrow$ *Boolean is true for at least one element of* S *(called initial state);*
- $\rho : S \times S \rightarrow$ *Boolean is a transition relation, and*
- $\eta : S \rightarrow$ *Boolean is true for at least one element of* S *(called final state)*

For each final state we associate a utility to each agent. A reachable state of Σ *is a state that can be reached following a finite sequence of transitions from an initial state. A reachable transition is a transition from a reachable state.*

We introduce the abstract version of the previous concept:

Definition 2. *An abstract finite game* $\widehat{\Sigma} = \langle \widehat{N}, \widehat{A}, \widehat{S}, \widehat{\theta}, \widehat{\rho}, \widehat{\eta} \rangle$ *is an abstraction of* $\Sigma = \langle N, A, S, \theta, \rho, \eta \rangle$ *if there exists a mapping* $\alpha : S \rightarrow \widehat{S}$ *such that*

- $\forall s \in S,\ \theta(s) \rightarrow \widehat{\theta}(\alpha(s))$
- $\forall s, s',\ \rho(s, s') \rightarrow \widehat{\rho}(\alpha(s), \alpha(s'))$

The mapping α is called an *abstraction map*. Its inverse γ, associating an abstract state $\widehat{s} \in \widehat{S}$ and transition $\widehat{\rho}$ to its corresponding concrete state $s \in S$ and transition ρ is called a *concretization map*.

Abstraction maps usually rely on over-approximations to produce, for every point of the program, an abstract state $\widehat{s}$ such that $\gamma(\widehat{s})$ contains all the concrete reachable states at that location. Traditionally, these approximations are defined over *lattices*.

Definition 3. *A lattice* $(L, \sqcup, \sqcap, \bot, \sqsubseteq)$ *is a complete partial order on set* L *by* $\sqsubseteq$ *in which any two elements* $x, y \in L$ *have a greatest lower bound* $(x \sqcap y) \in L$ *and a least upper bound* $(x \sqcup y) \in L$.

A lattice is *complete* if any two elements $x, y \in L$ have a greatest element $(x \sqcup y)$ and a least element $(x \sqcap y)$. An example of complete lattice is the power set domain with the usual set operators.

The game-theoretic properties we are interested in are first-order logic formulae (denoted as φ) that can be expressed in Σ and their abstract counterparts $\widehat{\varphi}$ in $\widehat{\Sigma}$. It is important to ensure that whenever a property φ is violated in the concrete domain Σ, its abstraction $\widehat{\varphi}$ is also violated in the abstract domain $\widehat{\Sigma}$.

Definition 4. *An abstraction* $\alpha : (\Sigma, \varphi) \rightarrow (\widehat{\Sigma}, \widehat{\varphi})$ *is sound if whenever* $\widehat{\varphi}$ *holds in* $\widehat{\Sigma}$*, then* φ *holds in* Σ*. An abstraction* $\alpha : (\Sigma, \varphi) \rightarrow (\widehat{\Sigma}, \widehat{\varphi})$ *is complete if whenever* φ *holds in* Σ*, then* $\widehat{\varphi}$ *holds in* $\widehat{\Sigma}$.

4.2 Building Abstractions for Auctions

Finding an abstraction map is not an easy task and depends on the property to be checked. In our work, abstraction is a way of minimising the explosion on the number of states of the concrete model as illustrated in Section 3. Bearing in mind that an abstraction map must be at least sound, we propose the following two abstractions.

Removal of Irrelevant Constructs. This abstraction amounts to the *program slicing* technique used to remove portions of code in program analysis which are not relevant to a given criterion [6]. A typical criterion is a line of the program – the slice contains those commands which affect the variables in that line. Another criterion is a set of variables – the slice contains those commands affecting these variables. Our slicing criterion is the property φ to be checked – the slice contains those portions which influence the variables that are being checked in the property.

The analysis of which programming constructs to include in (or exclude from) a slice is based on the semantics of the programming language. This is used to capture *dependency relationships* among variables: $v_1 \prec v_2$, for variables v_1, v_2, holds if the computation of v_1 depends on the value of v_2. For instance, the command `x := y + (z/2)` of a C-like language assigning to `x` the result of an expression which uses `y` and `z`, forges the relationships $\texttt{x} \prec \texttt{y}$ and $\texttt{x} \prec \texttt{z}$. This

relationship between variables is transitive and its transitive closure is denoted by $\prec^*$.

To perform program slicing, we first perform a dependency analysis on the mechanism description to determine the set of variables that influence the formula φ of the property to be checked. We then remove those constructs of the mechanism which do not make use of any of these variables. This is carried out as a *backward analysis* (from the program outputs to the program inputs) as follows:

1. Let V be the set of variables in φ;
2. For each variable $v_i \in V$, compute the set of variables V_i on which v_i depends and merge it with V, that is, $V \leftarrow V \cup V_i$ – this step should compute the transitive closure of variable dependency, that is, it must include all variables in the dependency paths from the inputs to the calculation of v_i.
3. Remove all constructs using only those variables that do not belong to V.

We regard a slice as an abstraction of the original program: a slice partially computes what the original program does. All those constructs making use of variables unrelated to the property are removed. Such slices do not alter the property φ to be checked but can reduce the size of the state space to be model checked, see for example [2]. Program slicing naturally provides a sound and complete abstraction map.

To illustrate this abstraction, let us consider the Vickrey auction example for two agents shown on the left-hand side of Figure 2. Let us suppose we want to

```
if
  :: (x1 >= x2) ->
    u1 = v1 - x2;
    u2 = 0;
  :: else ->
    u2 = v2 - x1;
    u1 = 0;
fi;
```

```
if
  :: (x1 >= x2) ->
    u1 = v1 - x2;

  :: else ->

    u1 = 0;
fi;
```

Fig. 2. Vickrey Auction (left) and its Slice (right)

check the assertion $\texttt{u1t} \geq \texttt{u1}$ where `u1t` and `u1` are, respectively, the utilities of agent 1 when it bids its valuation and any other number. The variable dependencies of the auction are $\texttt{x1} \prec \texttt{x2}, \texttt{x2} \prec \texttt{x1}, \texttt{u1} \prec \texttt{v1}, \texttt{u1} \prec \texttt{x2}, \texttt{u2} \prec \texttt{v2}, \texttt{u2} \prec \texttt{x1}$; they describe the flow of data among the variables and (in the case of the `if` test) how variables depend on one another to define the flow of execution. We can obtain a slice of the original auction in which commands not referring to any of the variables $y, \texttt{u1} \prec^* y$. We show on the right-hand side of Figure 2 a slice of the Vickrey auction, in which all commands referring to `u2` have been removed.

Redefining Strategy Space via Abstract Values. The principal cause of the explosion in the number of states observed in Section 3 is the exponential input data required by the strategy-proofness property. This input data describes all the strategy profiles for n players and m items in the combinatorial auction.

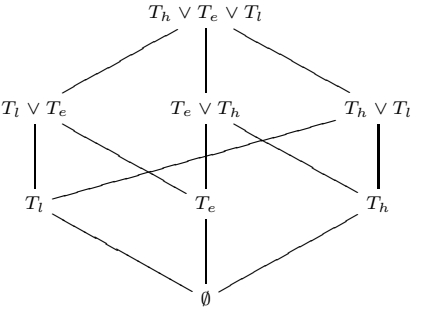

Fig. 3. Lattice of Abstract Values

This detailed search space can be replaced by a less complex one by transforming the concrete program into an abstract one. The variables in the abstract version of the program store abstract values, thus we need to redefine all concrete operations so as to manipulate the newly created values.

Given the valuation v_i of an agent i for a given single item, we can distinguish the following three strategies its opponents may adopt: (i) bid higher than v_i, (ii) bid exactly v_i, and (iii) bid lower than v_i. We define the following types for agents competing with agent i, corresponding to the strategies above:

$$T_h = \{x \in \mathbb{R} \mid x > v_i \geq 0\}$$
$$T_e = \{x \in \mathbb{R} \mid x = v_i\}$$
$$T_l = \{x \in \mathbb{R} \mid 0 \leq x < v_i\}$$

All the configurations of the game involving agent i with respect to its opponents can be described by agent i's bid against the bids of the typed agents in the set $T = \{T_h, T_e, T_l\}$. Consider the following three mappings with signature $A^{n-1} \rightarrow T$ and projecting each component of a vector $x_{-i} \in A^{n-1}$ to a type $t \in T$:

$$\text{proj}_h(x_{-i}) = \{x_j \in T_h \mid j \neq i\}$$
$$\text{proj}_e(x_{-i}) = \{x_j \in T_e \mid j \neq i\}$$
$$\text{proj}_l(x_{-i}) = \{x_j \in T_l \mid j \neq i\}$$

The mapping proj_h projects all components of the vector x_{-i} that are greater than v_i to the data type T_h. Similarly proj_e and proj_l are projections on T_e and T_l respectively. Let us consider $f : A^{n-1} \rightarrow 2^T$ mapping an element $x_{-i} \in A^{n-1}$ to an element $\widehat{x_{-i}} = f(x_{-i})$ of the powerset 2^T as follows:

$$\widehat{x_{-i}} = \begin{cases} T_h & \text{if } \text{proj}_e(x_{-i}) = \text{proj}_l(x_{-i}) = \emptyset \\ T_e & \text{if } \text{proj}_h(x_{-i}) = \text{proj}_l(x_{-i}) = \emptyset \\ T_l & \text{if } \text{proj}_e(x_{-i}) = \text{proj}_h(x_{-i}) = \emptyset \\ T_h \vee T_e & \text{if } \text{proj}_l(x_{-i}) = \emptyset \\ T_h \vee T_l & \text{if } \text{proj}_e(x_{-i}) = \emptyset \\ T_l \vee T_e & \text{if } \text{proj}_h(x_{-i}) = \emptyset \\ T_h \vee T_e \vee T_l & \text{otherwise} \end{cases}$$

Table 3. Signature of Abstract Subtraction $-_{\text{abs}}$

$-_{\text{abs}}$	x_n	x_l	x_e	x_h
x_n	$\{x_n, x_l, x_e, x_h\}$	x_n	x_n	x_n
x_l	$\{x_l, x_e, x_h\}$	$\{x_n, x_l\}$	x_n	x_n
x_e	x_h	x_l	x_l	x_n
x_h	x_h	$\{x_l, x_e, x_h\}$	$\{x_l, x_e, x_h\}$	$\{x_n, x_l, x_e, x_h\}$

By construction, f maps every vector of A^{n-1} to its equivalent type in the complete lattice $L = (2^T, \vee, \wedge, \emptyset, \sqsubseteq)$. The mapping f induces an equivalence relation whose equivalence classes represent state variables; these state variables correspond to elements of the powerset 2^T. In Figure 3 we show the lattice of abstract values. It is isomorphic to the complete lattice $L = (2^T, \vee, \wedge, \emptyset, \sqsubseteq)$.

Let x_h, x_e, x_l be the equivalent classes associated to the types T_h, T_e, T_l respectively. x_h, x_e, x_l are abstract variables that will be used in the transformed (abstract) program. Concrete arithmetic operations, e.g., +, −, ⋆, <, and >, must also be transformed so as to manipulate the abstract variables x_h, x_e, x_l. Moreover, the variables x_h, x_e, x_l cover real values that are greater than or equal to zero. However, the arithmetic operation "−" forces us to consider negative values as well, which we denote by x_n. We can therefore partition the set $\mathbb{R}$ into the subsets represented by the equivalence classes x_n, x_l, x_e, and x_h. The abstract variables x_n, x_l, x_e, and x_h represent the real-valued intervals $(-\infty, 0), [0, v_i), [v_i, v_i]$, and (v_i, ∞) respectively.

Table 3 shows the signature of the abstract operation $-_{\text{abs}}$ – it is the abstract counterpart of the subtraction operation. The first column of the table shows the values of the first parameter of the operation $-_{\text{abs}}$; the top row contains the values of the second parameter; the various outcomes of the operation are the table cells. If the result of the abstract operation belongs to a set of equivalence classes (as opposed to a single equivalence class) then this indicates a lack of knowledge about the abstract variables since they over-approximate concrete values in the original program. This inaccuracy is modelled by the model checker SPIN as a non-deterministic choice over the set of values in the set.

It follows that the mapping $f : \Sigma \rightarrow \widehat{\Sigma}$ enables us to transform concrete data and operations from the original program into corresponding abstract data and operations in the transformed program. The states and transitions in the abstract program are defined to be those induced by the states and transitions in the abstract program. An important issue is whether the abstraction f is sound for the game-theoretic property to be checked. For that purpose, we need to express the property in the resulting abstract domain.

4.3 Abstracting Properties

For a given valuation v_i of an agent i, the strategy-proofness property φ is defined in equation (3). In the resulting abstract model, the valuations v_i, bids $b_i \in A_i$, $b_{-i} \in A_{-i}$, payments p_i and utilities u_i of the agents become abstract variables $\widehat{v_i}$, $\widehat{b_i} \in \widehat{A_i}$, $\widehat{b_{-i}} \in \widehat{A_{-i}}$, $\widehat{p_i}$ and $\widehat{u_i}$ respectively. All original operations are also

transformed into abstract operations manipulating the defined abstract types. The strategy-proofness property φ in equation (3) becomes $\widehat{\varphi}$ as follows:

$$\forall i,\ \forall \widehat{v_i},\ \forall \widehat{b_i},\ \forall \widehat{b_{-i}},\ \widehat{u_i}(\widehat{v_i}, \widehat{v_i}, \widehat{b_{-i}}) \geq_{\text{abs}} \widehat{u_i}(\widehat{v_i}, \widehat{b_i}, \widehat{b_{-i}}). \tag{5}$$

Using our abstraction f, we have $\widehat{A_i} = \{x_h, x_e, x_l\}$, which is a partition of A_i and $\widehat{A_{-i}}$ is isomorphic to 2^T (which we denote by $\widehat{A_{-i}} \equiv 2^T$). We now need to prove that our abstraction f is sound. This is established by the following:

Lemma 1. *The abstraction map $f : (\Sigma, \varphi) \rightarrow (\widehat{\Sigma}, \widehat{\varphi})$ is sound.*

Proof: We need to prove that if the strategy-proofness property $\widehat{\varphi}$ holds in $\widehat{\Sigma}$, then its equivalent version φ holds in Σ. Following Definition 2, an abstraction f produces for every point of the program, an abstract state $\widehat{s}$ such that $f^{-1}(\widehat{s})$ contains all the concrete reachable states at that location. We need to show that for every configuration c of the abstract domain wherein $\widehat{\Sigma}$ holds, $f^{-1}(c)$ contains all possible corresponding configurations in the concrete domain and that we have $\Sigma \subseteq \cup_{c \in \widehat{\Sigma}} f^{-1}(c)$. From the viewpoint of agent i, strategy-proofness in the abstract domain means:

- If i has opponents of one type T_h, T_e, or T_l, then the inequality (5) must be true for all $\widehat{b_i} \in \widehat{A_i}$ and $\widehat{b_{-i}}$ taking the values representing the equivalence classes x_h, x_e, or x_l of the abstraction f respectively.
- If i has opponents of two types $\{T_h, T_e\}, \{T_h, T_l\}$, or $\{T_e, T_l\}$, then the inequality (5) must be true for all $\widehat{b_i} \in \widehat{A_i}$ and $\widehat{b_{-i}}$ taking the values x_h and x_e, x_h and x_l, or x_e and x_l of the abstraction f respectively.
- If i has opponents of three types $\{T_h, T_e, T_l\}$, then the inequality (5) must be true for all $\widehat{b_i} \in \widehat{A_i}$ and $\widehat{b_{-i}}$ taking the values x_h, x_e, x_l.

By construction, the inverse f^{-1} of the mapping f associates each element of $\widehat{A_{-i}} \equiv 2^T$ to a subset of A_{-i} and clearly

$$\cup_{c \in \widehat{A_{-i}}} f^{-1}(c) = A_{-i}.$$

Furthermore, $\widehat{A_i}$ is a partition of A_i. It follows that f is sound. □

4.4 An Abstract Model-Checking Algorithm

To check the strategy-proofness property for a given player amounts to checking bidding x_e gives the maximum utility in all the following settings:

1. Its opponents of single type can bid a single value x_h, x_e, or x_l.
2. Its opponents of two types can bid the tuples (x_h, x_e), (x_h, x_l), or (x_e, x_l).
3. Its opponents of three types can bid the tuple (x_h, x_e, x_l).

This reduces the strategy space from the size $|A|^{nm}$ initially to $(3 \times 7)^m$, three for agent i and seven for its opponents. If the number of items $m = 1$ as, for example, in the Vickrey auction, this is easy to check. Notice that our abstraction

is only sound – this means that if the property is true in the abstract domain, then, it is true is the concrete domain. If, however, the property does not hold in the abstract model, then SPIN will generate a counter-example. The generated counter-example may be due to spurious behaviour caused by approximations in the abstract model or it may be genuine. Techniques have been developed to cope with such scenarios, see for example [17,18].

We have implemented this algorithm for checking strategy-proofness in the Vickrey auction. For this simple single item auction, we have designed and implemented the abstract variables and related operations, thus building up an abstract program modelling the auction. Then, we checked the abstracted strategy-proofness property using our algorithm. In the results shown in Table 4, AMCA stands for the abstract model checking algorithm hereby outlined and Slicing stands for the application of the program slicing optimisation. These results show that for the Vickrey auction, the number of players and the bid range cease to be a factor of state space explosion and that strategy-proofness can be checked using a small amount of computer resources. Moreover, the program slicing technique improved slightly the checking as expected in this case.

Table 4. Statistics for the strategy-proofness property in a Vickrey Auction with an unbounded number of players using the proposed two abstractions

	AMCA	AMCA & Slicing
Memory (Mb)	3.65	3.02
CPU Time (s)	0.31	0.25

For the QRMOA, which is a tractable combinatorial auction, the winner determination algorithm represents around 400 lines of Promela code containing data structures for matrices. Furthermore, the integration of the VCG mechanism and the testing procedure represent an extra 200 lines of code. This is too complex to hand-code an abstract program from. As a first attempt to model-check this rather challenging system, we kept the Promela code for the QRMOA unchanged and applied our abstraction only to the VCG procedure. Since QRMOA's code works with concrete values, we have used a mapping from abstract to concrete values and vice versa. In the forward mapping we converted abstract data from the abstracted VCG procedure using a non determistic selection over the corresponding concrete values. This enables us to run the WDA. In the reverse mapping we fed the results (concrete values) of the WDA back to the abstracted VCG procedure. Running our abstract verification algorithm for this case led to a "run out of memory" on the PC we have used. However, one can assume the optimality of WDA is a priori established and therefore the code that implements it can be trusted. In this case, we are left to checking the pricing rule gives rise to a strategy-proof mechanism. This should reduce the state space required by the model-checker. We set out our ideas on how to model-check this combinatorial auction using this assumption and our abstraction in the concluding section.

5 Related Work

The idea of agents automatically checking game-theoretic properties of a protocol is relatively new, so there are not many results in this area as yet. However earlier work explored similar ideas. Related work includes verification of MASs and model checking using abstraction techniques.

5.1 Verification of MASs

Verification of MASs has been explored in [19]. The approach consists on automatically translating MAS programmed in AgentSpeak, a BDI language proposed in [20] into Promela or Java. After this translation the approach uses the SPIN or JPF (Java PathFinder) [21] model checkers, respectively, to verify whether a property (in linear temporal logic) holds. In the context of verifying game equilibria, the work reported in [22] dealt with the verification of mechanisms that are specified in a WHILE programming language. Game-theoretic properties for 2-player games with complete and 'almost perfect' information using correctness assertions, via an extension of Hoare's calculus were proved. The main advantage of this approach is that it offers the possibility of verifying large games without needing to explore every state that the system can reach; the advantage of our model checking approach is that it offers the possibility of agents checking mechanisms automatically.

In [23] a model checking approach was adopted in verifying mechanisms, and the authors set forth their vision for how this approach can contribute to the mechanism design problem of game theorists. The paper shows, as an example, how a voting mechanism can be formalised and checked to see if a coalition of agents can force a deadlock indefinitely. The main motivation was to provide computing tools which can make mechanism specifications unambiguous, reveal hidden assumptions and automate the verification of desirable properties. Our work aims not to design new mechanisms, but to make existing ones useful to agents in open systems, by giving agents an efficient way of checking the properties of a previously unseen game specification.

In [24] the same issues as ours were explored, but from the perspective of building trust in an agent system. To this end, the paper looks at verifying the reputation of an agent as well as verifying that a protocol enforces truth-telling. However it does not go as far as analyzing the potential utilities which participants could gain, as we have; our approach is more computationally expensive, but this seems necessary to provide guarantees about truth-telling.

5.2 Model Checking and Abstraction

The use of abstraction in model checking derives from the seminal work on abstract interpretation published in [16] three decades ago. Abstract interpretation concerns static determination of dynamic properties of a system. Its principles are based on a central paradigm common to many engineering activities. This paradigm consists in modelling the system by a set of equations, solving those equations and using the solutions to predict the runtime behavior of the system.

In model checking, abstract interpretation is a tool to cope with the state space explosions suffered by model checkers.

In [2] the authors present an approach to slicing a multiagent system before it is model-checked, (possibly) reducing its state-space complexity. That work aims at slicing abstraction to agent protocols specified via a restricted form of AgentSpeak(L) [20] in which only propositions are considered. They proposed an algorithm which takes as input a set of agent programs, a property in a restricted (propositional) BDI logic, and the environment abstracted as rules updating the state of affairs. The algorithm builds a representation of the dependencies among the agents' programs and environment rules, then uses this representation to find out plans which were not used. Although the example presented in that paper illustrates well the benefits of program slicing, the authors fail to warn that, in the worst case, their slicing algorithm may return the very same input MAS – this happens if the property to be checked requires that all original parts be preserved.

In [12], the ALLOY model checker based on first order relational logic [13] was used to verify strategy-proofness for auctions. This provided an elegant way of modelling the auction and express the game-theoretic property but it does not scale up very well since the ALLOY analysis is intractable asymptotically. ALLOY's analysis relies on the assumption that "negative answers tend to occur in small models already, boosting the confidence we may have in a positive answer" [25, p. 143]. Tractability is therefore achieved by restricting the analysis to a finite universe.

This work on abstraction in model checking is closely related to that of [26] on detection of safety violations in software requirements specifications and that of [17] on model checking Java source codes. In [26], program slicing and tailored data abstractions were used along with the SPIN model checker. In [17], an integrated approach between the Bandera tool [27] and the Java Path Finder [21] model checker is used to search for counter examples in the abstract model that are feasible in the concrete model. Bandera implements a program slicer and provides a set of abstract definitions of program data types. However, abstraction is not a 'one size fits all'. Our work uses program slicing and presents a new property-preserving abstraction for verifying game-theoretic property of auction mechanisms. A shorter version of this work looking at only the case of single item auctions is published in [3].

6 Conclusions and Future Work

In this paper, we have considered auction mechanisms expressed in a formal language and automatically checked desirable properties such as strategy-proofness. We have presented numerical results showing the computational limits of using a plain (exhaustive) model checking approach. These limits are due to the state space explosion problem. To enhance this approach, we have combined model checking and abstract interpretation. We have proposed two property-preserving abstractions. The first is the classical program slicing technique; the second is

novel and tailored to the problem of verifying game equilibria. This allowed us to verify the Vickrey auction regardless of the number of bidders and their bids range with a small amount of computer resources. This was not feasible by exhaustive model checking, see Table 1. Note that although the abstraction requires some creativity from the human designer, once the appropriate abstraction is found, it can be published as a trusted procedure to facilitate automatic checking of the auction mechanism by agents.

For future work, we plan to apply our abstraction framework to the combinatorial auction QRMOA as follows. We will use the winner determination algorithm as a *trusted black box* using SPIN's `c_code`. This means implementing the winner determination algorithm in the C programming language and embedding it within SPIN so that it is not checked by SPIN but it behaves like a simple state transformer [28]. This will further demonstrate the benefits of our abstraction techniques and will enable us to assess the impact of increasing the number of items for our checking algorithm. We will also investigate the use of this abstraction in verifying Bayesian Nash equilibria. Computationally verifiable mechanisms are useful for agent scenarios wherein trust in the system must be guaranteed and entry-deterrence be tackled in order to attract more participants.

Acknowledgement. We thank the UK EPSRC for funding this project under grant EP/D02949X/1.

References

1. Guerin, F., Tadjouddine, E.M.: Realising common knowledge assumptions in agent auctions. In: The IEEE/WIC/ACM Int'l Conf. on Intelligent Agent Technology, Hong Kong, China, pp. 579–586 (2006)
2. Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: State-space Reduction Techniques in Agent Verification. In: AAMAS 2004, pp. 896–903. ACM Press, New York (2004)
3. Tadjouddine, E.M., Guerin, F., Vasconcelos, W.: Abstractions for model checking game-theoretic properties in auctions. In: AAMAS (accepted, 2008)
4. Tennenholtz, M.: Some tractable combinatorial auctions. In: AAAI, pp. 98–103 (2000)
5. Holzmann, G.J.: The SPIN Model checker: Primer and Reference Manual. Addison, Boston (2004)
6. Tip, F.: A Survey of Program Slicing Techniques. Journal of Progr. Lang. 3, 121–189 (1995)
7. Cramton, P., Shoham, Y., Steinberg, R.: Combinatorial Auctions. MIT Press, Cambridge (2006)
8. Cavallo, R.: Optimal decision-making with minimal waste: strategyproof redistribution of VCG payments. In: AAMAS, pp. 882–889 (2006)
9. Nisan, N., Ronen, A.: Computationally feasible VCG mechanisms. In: ACM Conference on Electronic Commerce, pp. 242–252 (2000)
10. Rothkopf, M.H., Pekec, A., Harstad, R.M.: Computationally manageable combinatorial auctions. Management Science 44, 1131–1147 (1998)

11. Gabow, H.N., Tarjan, R.E.: Faster scaling algorithms for network problems. SIAM J. Comput. 18, 1013–1036 (1989)
12. Tadjouddine, E.M., Guerin, F.: Verifying dominant strategy equilibria in auctions. In: Burkhard, H.-D., Lindemann, G., Verbrugge, R., Varga, L.Z. (eds.) CEEMAS 2007. LNCS, vol. 4696, pp. 288–297. Springer, Heidelberg (2007)
13. Jackson, D.: Automating first-order relational logic. In: SIGSOFT FSE, pp. 130–139 (2000)
14. Taha, H.A.: Operations Research: An Introduction, 6th edn. Prentice-Hall, Englewood Cliffs (1997)
15. Cousot, P.: Program Analysis: The Abstract Interpretation Perspective. ACM Computing Surveys 28, 165 (1996)
16. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: The Fourth Annual ACM SIGPLAN-SIGACT Symposium on POPL, Los Angeles, California, pp. 238–252. ACM Press, New York (1977)
17. Pasareanu, C.S., Dwyer, M.B., Visser, W.: Finding feasible abstract counterexamples. Soft. Tools for Tech. Transfer 5, 34–48 (2003)
18. Saïdi, H.: Model checking guided abstraction and analysis. In: Palsberg, J. (ed.) SAS 2000. LNCS, vol. 1824, pp. 377–396. Springer, Heidelberg (2000)
19. Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: Verifying Multi-Agent Programs by Model Checking. Autonomous Agents and Multi-Agent Systems 12, 239–256 (2006)
20. Rao, A.S.: AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
21. Visser, W., Havelund, K., Brat, G., Park, S.J.: Model checking programs. In: Proc. of the 15th IEEE International Conf. on Automated Software Engineering (2000)
22. Pauly, M.: Programming and verifying subgame-perfect mechanisms. J. Log. Comput. 15, 295–316 (2005)
23. Pauly, M., Wooldridge, M.: Logic for mechanism design—a manifesto. In: GTDT 2003 workshop, Hakodate, Japan, AAMAS 2003 (2003)
24. Osman, N., Robertson, D.: Dynamic verification of trust in distributed open systems. In: IJCAI, pp. 1440–1445 (2007)
25. Huth, M.R.A., Ryan, M.D.: Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press, Cambridge (2000)
26. Heitmeyer, C., Kirby, J., Labaw, B., Archer, M., Bharadwaj, R.: Using abstraction and model checking to detect safety violations in requirements specifications. IEEE Transactions on Software Engineering 24, 927–948 (1998)
27. Corbett, J.C., Dwyer, M.B., Hatcliff, J., Laubach, S., Păsăreanu, C.S., Zheng, H.: Bandera. In: Proceedings of the 22nd International Conference on Software Engineering, pp. 439–448. ACM Press, New York (2000)
28. Holzmann, G.J.: Personal communication (2008)

Using Temporal Logic to Integrate Goals and Qualitative Preferences into Agent Programming*

Koen V. Hindriks[1] and M. Birna van Riemsdijk[2]

[1] EEMCS, Delft University of Technology, Delft, The Netherlands
[2] LMU, Munich, Germany

Abstract. The core capability of a rational agent is to choose its next action in a rational fashion, a capability that can be put to good use by a designer to satisfy the design objectives of an agent system. In agent programming languages for rational agents, such choices are derived from the agent's beliefs and goals. At any one time, an agent can typically choose from multiple actions, which may all lead to goal achievement. Existing approaches usually select one of those actions non-deterministically. In this paper, we propose the use of goals as *hard constraints* and qualitative preferences as *soft constraints* for choosing a most preferred action among the available ones. We use temporal logic for the representation of various kinds of goals and preferences, leading to a uniform framework for integrating goals and preferences into agent programming.

1 Introduction

The core component of a rational agent is its capability to make rational choices of action, in order to satisfy its design objectives. In agent programming languages for rational agents, such choices are derived from the agent's beliefs and goals. These goals are often achievement goals, i.e., goals that define states that are to be achieved. Typically, agent programs consist of a number of rules (usually called action selection or plan rules) that provide the agent with the means to choose actions that achieve its goals. Such rule sets, however, do not necessarily completely determine the choice of action, and, at any one time, an agent can typically choose from *multiple* actions, which may all lead to goal achievement. Existing approaches usually select one of those actions non-deterministically.

Recent research in both the planning and agent programming field [1,2,10,12,13,15] has shown that it can be useful to have additional mechanisms for further *constraining* the choice of action in an agent program and thus to reduce the non-determinism that would be present in an agent program without such mechanisms. In [13], for example, we introduced an operational semantics for so-called *maintenance goals* which express conditions that must remain true throughout (some part of) the agent's lifetime. Adding maintenance goals to

* This work has been sponsored by the project SENSORIA, IST-2005-016004.

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 215–232, 2009.

agent programs provides a programmer with a tool to restrict the options for action selection that an agent has, since the agent is programmed to avoid selecting actions that would violate maintenance goals.

In this paper, we investigate such mechanisms for further constraining an agent's choice of action in agent programming languages in a more general and uniform setting. We distinguish between *hard constraints* which *must* be satisfied, and *soft constraints* or preferences which allow an agent to distinguish preferred courses of action from those that are less preferred. The main contribution of this paper consists of the layered rational action selection architecture (*RASA* for short) that we propose as a conceptual framework for rational action selection in agent programming. The RASA architecture introduces a uniform framework based on temporal logic that integrates rule-based action selection (based on beliefs and goals), hard constraints such as maintenance goals, and soft constraints such as preferences.

We show how the proposed architecture can be operationalized using the GOAL agent programming language [7]. For this purpose, we introduce an extension of the GOAL language that incorporates temporal logic operators used to represent both goals as well as preferences. A key motivation for integrating temporal logic into agent programming languages is the potential that the additional expressiveness thus introduced provides for defining a uniform framework that naturally allows for the integration of hard and soft constraints, including such concepts as *achievement goals*, *maintenance goals*, as well as *temporally extended preferences*. The work reported is inspired among others by work in planning where temporal logic is used to "guide" planners through the search space [1] and to select preferred plans [2,10,15].

In Section 2, we discuss rational action selection in agent programming, and informally present the rational action selection architecture. In Section 3, we present the extension of GOAL with temporal logic, which forms the first layer of our action selection architecture. Section 4 defines some technical preliminaries with respect to temporal logic, which are needed in Sections 5 and 6 in which we introduce the second and third layer, respectively, of the action selection architecture. In Section 7 we conclude the paper.

2 Rational Action Selection Architecture

An agent program typically does not completely determine the behaviour of the agent in each state as it allows multiple actions to be executed in such a state. Agent programs thus typically *underspecify* an agent's behaviour.[1] For example, consider a carrier agent that needs to bring parcels to two different locations A and B. An agent program for such a carrier agent may instruct the agent to $goto(A)$ and $goto(B)$ but leave unspecified in which order this should be done.

[1] It may be the case that interpreters for agent programs are implemented such that they produce the same sequence of actions each time the agent program is executed. However, when read as specifications they typically do not dictate such a unique course of action.

The underspecification of agent behaviour can have benefits from a design point of view. If it does not matter whether the agent executes one action or another for reaching a goal, one can argue that it is more natural to let the agent program reflect this by leaving these choices open. However, recent research in both the planning and agent programming field [1,2,10,12,13,15] has shown that it can be useful to have additional mechanisms for finding courses of actions to further constrain the choice of action in an agent program. The idea is that *additional selection mechanisms can be introduced on top of an existing agent programming semantics* which can be used by the agent to further restrict the selection among actions. For example, one might wish to specify that going to location A first before going to location B is to be preferred by the carrier agent. Preferences such as these are difficult to code into an agent program, and, we argue, are more naturally introduced as an additional constraint in the agent program. In this particular example, an agent then should check whether a selected (course of) action satisfies such constraints to optimize its performance. In this section, we present a layered rational action selection architecture (RASA) for adding such additional selection mechanisms on top of agent programs.

2.1 An Architecture for Rational Action Selection

Regarding the kinds of additional constraints that may be applied to select among the possible courses of action, we distinguish between *hard constraints* which *must* be satisfied, and *soft constraints* or *preferences*, by means of which one can distinguish more preferred courses of action from less preferred ones (see also [15,10,13]).

What one takes as hard constraints varies across approaches. In [15,10], and more generally in planning, the (achievement) goals themselves are considered to be hard constraints.[2] In [13], maintenance goals are taken as hard constraints, i.e., an agent may never violate a maintenance goal. Preferences can be used to distinguish between optimal and suboptimal courses of action, e.g., in terms of costs, but may also be used to distinguish courses of action with respect to which goals are reached (if goals are not considered as hard constraints and not all goals can be reached).

These considerations lead to the following layered rational action selection architecture.

- **Layer 1:** The RASA *generates* options for courses of action, typically on the basis of an agent's beliefs and (achievement) goals.
- **Layer 2:** The RASA verifies whether the courses of action from layer one satisfy *hard constraints* and discards those that do not.
- **Layer 3:** The RASA selects from the options remaining after the application of layer two those courses of action that maximize satisfaction of *soft constraints*.

[2] That is, in that approach the initial goals are the basis for plan generation, while at the same time being considered as hard constraints. The initial goals are thus not used as an *additional* action selection mechanism.

In this paper, we show how this architecture can be made concrete in the context of the GOAL agent programming language.

2.2 Rational Action Selection in Agent Programming

This rational action selection architecture is inspired among others by research on planning with preferences [10,15,4,2]. The classical AI planning problem is to search for a plan (a sequence of actions) to get from the current state to a goal state, given a set of action specifications [9,11]. In agent programming, on the other hand, the behaviour of the agent is specified by means of a program [5]. One of the main differences between planning and programming approaches, is that in planning one seeks a *complete plan*, the execution of which will result in the agent achieving its goal (given certain assumptions on the environment). An agent program, on the other hand, is executed *step by step*, typically without first checking whether the executed actions will eventually lead to the agent reaching its goal.[3] That is, in our approach we want the agent to execute an action at each step, even though it does not know for sure that the action is the best one according to the hard and soft constraints. Nevertheless, we *do* want the agent to take into account the constraints to select an action that is at least likely to be a good one.

The way in which we propose to do this, is partly based on our previous work on the incorporation of maintenance goals in agent programming languages [13]. The idea is that the agent has a fixed, usually finite, *lookahead horizon*, i.e., the agent can lookahead a certain number of execution steps. The agent then evaluates the possible courses of action or paths it can take (up to the given horizon) using its constraints, in order to determine which paths are the best. It then takes one step along one of these paths, and the process is repeated.

It is important to note that, when the agent takes a step in a particular direction, it does not have complete knowledge of the outcome of following that path. It can only look forward until its lookahead horizon, but does not know what happens beyond this horizon. This means that the agent may take a step that is suboptimal, i.e., the agent uses its constraints as a *heuristic* for action selection. Constraints are also used as a heuristic in some planning approaches [1,2], in which they are used to guide the search for an optimal plan.

The technical tool we use for the representation of goals and preferences is linear temporal logic (LTL) [8]. The idea is that if the agent has a temporal formula as a goal, it should try to produce execution traces on which this temporal formula holds, and similarly for preferences. This idea is inspired by work on the representation of goals and qualitative preferences in the context of planning [1,10,15,4,2]. While LTL formulas are typically evaluated on infinite traces, we need to evaluate LTL formulas on finite traces, since we use a finite lookahead

[3] If the programmer has written a program that is correct (with respect to some specification), of course, the execution will indeed lead to goal achievement (cf. [7] for a verification framework of the agent programming language GOAL.). Verifying agent programs that are executed in dynamic, unpredictable environments, however, is an unsolved problem and a non-trivial undertaking in practice.

horizon. It turns out that the 3-valued semantics of LTL as proposed in [3] provides a natural solution to this problem, and we use it to incorporate temporal goals and preferences in GOAL.

3 RASA Layer 1: Temporalized GOAL

In this section, we define the first layer of the RASA architecture in the context of the GOAL agent programming language [7,14]. We extend the original GOAL language by allowing temporal formulas as goals, rather than only propositional formulas.

3.1 The General Idea

In the GOAL language, an agent selects actions on the basis of its beliefs and goals. A program consists of (1) a set of beliefs, collectively called the *belief base* of the agent, (2) a set of goals, called the *goal base*, (3) an *action specification* which consists of a specification of the pre- and post-conditions of *basic actions* of the agent, and (4) a program section which consists of a set of *actions rules*.

In the original GOAL language, the belief base and goal base are sets of propositional formulas. The goals are interpreted as achievement goals. That is, if a propositional formula ϕ is in the goal base, this informally means that the agent wants to reach a situation in which ϕ is (believed to be) the case. If a basic action is executed, the agent's beliefs change as specified in the pre- and postconditions of the action, and the achievement goals that are believed to be reached through the execution of the action are removed from the goal base.[4] As an example, an action $goto(A)$ would update the belief base to include $at(A)$. An action rule consists of a basic action and a condition on the agent's beliefs and goals. Such a rule expresses that the basic action may be executed, if the condition holds. An action rule might specify that the agent only goes to a location x if a parcel needs to be delivered to x (a goal) and the agent does not believe it is currently at x. During execution, a GOAL agent selects non-deterministically any of its enabled action rules, i.e., an action rule of which the condition holds, and then executes its corresponding basic action. At any one time, typically *multiple* action rules are enabled, i.e., a GOAL program underspecifies an agent's behaviour.

In [13], we have extended the GOAL language with maintenance goals. Just like achievement goals, maintenance goals were also expressed by propositional formulas. A maintenance goal ϕ expresses that the agent wants that ϕ holds continuously throughout the execution of the agent. Both achievement goals

[4] The idea is that the agent's beliefs also represent the environment, and that basic actions change this environment. However, the environment is not modeled in the formal specification of GOAL, and consequently basic actions update only the belief base.

and maintenance goals express particular desired properties of the behaviour of the agent. These properties can be expressed conveniently in LTL. LTL has computation traces (sequences of states) as models. An achievement goal for ϕ can be represented by the LTL formula $\Diamond\phi$, specifying that ϕ should hold eventually, i.e., in some state on the computation trace. A maintenance goal for ϕ can be represented by $\Box\phi$, specifying that ϕ should always hold. This idea can be generalized by realizing that in fact any LTL formula can be used for expressing goals. As an example, $\Diamond at(A)$ may be used to represent the achievement goal of being at location A and $\Box fuel(x) \wedge x > 30$ may be used to represent a maintenance goal of having always at least 30 units of fuel in the tank.

In this paper, we make this idea concrete in the context of the GOAL language by using LTL for the representation of goals. This increases the expressiveness of the language by allowing the representation of all kinds of goals, and allows for the representation of goals in a uniform way. For example, the goal $\phi U \phi'$, which expresses that the agent wants to ensure that ϕ while it is trying to achieve ϕ', can now be expressed easily.

3.2 Formalization

First, we define the LTL language that we use for representing goals. For reasons of simplicity we do not extend the beliefs of an agent in the programming language to temporal formulas. For example, it would be more involved to establish when an agent should drop one of its goals.

The states of the traces on which the LTL formulas are evaluated are belief bases, i.e., goals express how the belief base of the agent should evolve.[5] We assume a language $\mathcal{L}_0$ of propositional logic with typical element ϕ, and the standard entailment relation $\models$. A belief base $\Sigma \subseteq \mathcal{L}_0$ is a set of propositional formulas. Our LTL language contains the standard (temporal) operators of LTL. The difference between our LTL language and standard LTL is that the states of the traces are belief bases, rather than worlds as valuations of propositional atoms. The semantics of non-temporal propositional formulas is thus defined on belief bases. We specify that a propositional formula ϕ holds in a belief base Σ if $\Sigma \models \phi$.

Definition 1 *(linear temporal logic (LTL))*
Let $\phi \in \mathcal{L}_0$. The set of LTL formulas $\mathcal{L}_{LTL}$ with typical element χ is defined as follows.

$$\chi ::= \top \mid \phi \mid \neg\chi \mid \chi_1 \wedge \chi_2 \mid \Diamond\chi \mid \bigcirc\chi_1 \mid \chi U \chi_2$$

Let $t^b = \Sigma_0, \Sigma_1, \ldots$ be an infinite trace of belief bases and let $i \in \mathbb{N}$ be a position in a trace. The semantics of LTL formulas is defined on infinite traces t^b as follows.

[5] As the idea is that the beliefs also represent the environment of the agent, goals also express desired properties of the environment.

$$
\begin{array}{ll}
t^b, i \models_{LTL} \top & \\
t^b, i \models_{LTL} \phi & \Leftrightarrow \Sigma_i \models \phi \\
t^b, i \models_{LTL} \neg\chi & \Leftrightarrow t^b, i \not\models_{LTL} \chi \\
t^b, i \models_{LTL} \chi_1 \wedge \chi_2 & \Leftrightarrow t^b, i \models_{LTL} \chi_1 \text{ and } t^b, i \models_{LTL} \chi_1 \\
t^b, i \models_{LTL} \Diamond\chi & \Leftrightarrow \exists k \geq i : t^b, k \models_{LTL} \chi \\
t^b, i \models_{LTL} \bigcirc\chi & \Leftrightarrow t^b, i+1 \models_{LTL} \chi \\
t^b, i \models_{LTL} \chi_1 U \chi_2 & \Leftrightarrow \exists k \geq i : t^b, k \models_{LTL} \chi_2 \text{ and } \forall i \leq l < k : t^b, l \models_{LTL} \chi_1
\end{array}
$$

As usual, the always operator is defined in terms of the eventually operator by: $\Box\chi \equiv \neg\Diamond\neg\chi$.

The mental state of a GOAL agent consists of those components that change during execution. That is, a mental state consists of a belief base and a goal base. The goal base is typically denoted by Γ and in temporalized GOAL this is a set of LTL formulas. Mental states should satisfy a number of rationality constraints.

Definition 2 *(Mental States)*
A mental state of a GOAL agent, typically denoted by m, is a pair $\langle \Sigma, \Gamma \rangle$ with $\Sigma \subseteq \mathcal{L}_0$ and $\Gamma \subseteq \mathcal{L}_{LTL}$ where Σ is the belief base, and Γ with typical element χ is the goal base. Additionally, mental states need to satisfy the following *rationality constraints*:

(i) The belief base is consistent: $\Sigma \not\models \bot$,
(ii) The goal base is consistent: $\Gamma \not\models_{LTL} \bot$,
(iii) The goal base does not contain goals that have already been achieved.

The third rationality constraint is implemented by means of the progression operator which will be introduced below (Definition 4).

A GOAL agent derives its choice of action from its beliefs and goals. In order to do so, a GOAL agent inspects its mental state by evaluating so-called *mental state conditions*. The syntax and semantics of these conditions is defined next.

Definition 3 *(Mental State Conditions)*
Let $\phi \in \mathcal{L}_0$, $\chi \in \mathcal{L}_{LTL}$. The language $\mathcal{L}_M$ of mental state conditions, typically denoted by ψ, is defined as follows.

$$\psi ::= \mathbf{B}\phi \mid \mathbf{G}\chi \mid \neg\psi \mid \psi_1 \wedge \psi_2$$

The truth conditions of mental state conditions ψ, relative to a mental state $m = \langle \Sigma, \Gamma \rangle$, are defined as follows.

$$
\begin{array}{lll}
m \models_m \mathbf{B}\phi & \text{iff} & \Sigma \models \phi \\
m \models_m \mathbf{G}\chi & \text{iff} & \Gamma \models_{LTL} \chi \\
m \models_m \neg\psi & \text{iff} & m \not\models_m \psi \\
m \models_m \psi_1 \wedge \psi_2 & \text{iff} & m \models_m \psi_1 \text{ and } m \models_m \psi_2
\end{array}
$$

The semantics of $\mathbf{B}\phi$ is defined relative to a given belief base: $\mathbf{B}\phi$ holds iff ϕ follows from the belief base under a standard propositional logic entailment relation. The

semantics of the **G** operator is different from original GOAL, where the goal base is a set of propositional formulas. Here, we use the LTL entailment relation to generalize the operator to be able to express arbitrary types of goals. $\neg\mathbf{B}(at(x)) \wedge \mathbf{G}(\Diamond at(x))$ is a simple example of a mental state condition that expresses that the agent does not belief it is at location x although it wants to be. Such a condition can be used to determine whether to goto a location x (see below).

Before we can move on to defining how the execution of a basic action changes the agent's mental state, we need to explain how the goal base is updated. In the original GOAL language, achievement goals that are believed to be achieved after the execution of a basic action are removed from the goal base. Checking whether an achievement goal is achieved is simple if these are represented as propositional formulas: an achievement goal ϕ is achieved in a mental state if it follows from the belief base in that mental state. In temporalized GOAL, goals are temporal formulas. In order to be able to evaluate the achievement of temporal formulas in a mental state, we use a technique from [1] for "progressing" LTL formulas.

Progression of an LTL formula is a transformation of this formula, which should be performed at each execution step. The idea is that the transformation yields a new formula in which those "parts" of the formula which have already been achieved are set to $\top$, leaving an LTL formula which expresses what still has to be satisfied.

For example, if the agent has a goal $\Diamond\phi$ ($\phi \in \mathcal{L}_0$) in a particular mental state and ϕ is achieved in that mental state, i.e., follows from the belief base, then the progression of this formula is "$\top$", as the agent has produced an execution trace on which the formula holds. If ϕ does not hold, then the progression is the formula "$\Diamond\phi$" itself, since it still needs to be satisfied. If a formula has progressed to a formula equivalent to $\top$, it means the agent has produced an execution trace on which the formula holds. Note that formulas of the form $\Box\chi$ can never progress to $\top$, since it needs to be checked continuously whether χ holds.

The progression operator can be defined inductively for general LTL formulas, as specified in the next definition. We adapt the definition of [1] slightly, as we want a formula which is reached in a mental state to be true already in that state, rather than one mental state later. For this, we define the progression of $\bigcirc\chi'$ as $Progress(\chi')$, rather than as χ'.

Definition 4 *(progression of LTL formulas)*
Let Σ be a belief base, let $\chi \in \mathcal{L}_{LTL}$ be an LTL formula, and let $\phi \in \mathcal{L}_0$. The progression of χ in Σ, $Progress(\chi, \Sigma)$ is then defined as follows.

form of χ is	$Progress(\chi, \Sigma) =$
$\top$	$\top$
ϕ	$\top$ if $\Sigma \models \phi$, $\bot$ otherwise
$\neg\chi'$	$\neg Progress(\chi', \Sigma)$
$\chi_1 \wedge \chi_2$	$Progress(\chi_1, \Sigma) \wedge Progress(\chi_2, \Sigma)$
$\Diamond\chi'$	$Progress(\chi', \Sigma) \vee \chi$
$\bigcirc\chi'$	$Progress(\chi')$
$\chi_1 U \chi_2$	$Progress(\chi_2, \Sigma) \vee (Progress(\chi_1, \Sigma) \wedge \chi)$
$\Box\chi'$	$Progress(\chi', \Sigma) \wedge \chi$

We lift the progression function of Definition 4 to sets of LTL formulas $\Gamma \subseteq \mathcal{L}_{LTL}$ as follows: $Progress(\Gamma, \Sigma) = \bigcup_{\chi \in \Gamma} Progress(\chi, \Sigma)$.

The next definition specifies how the execution of a basic action changes an agent's mental state. In the formal definition of GOAL, we use a *transition function* $\mathcal{T}$ to model the effects of basic actions for technical convenience, rather than a specification of pre- and postconditions. The function $\mathcal{T}$ maps a basic action $\mathtt{a}$ and a belief base Σ to an updated belief base $\mathcal{T}(\mathtt{a}, \Sigma) = \Sigma'$. The transition function is undefined if an action is not enabled in a mental state. For example, the transition function may specify that an atom $at(A)$ is added to the belief base, if a basic action $goto(A)$ is executed, while the action is undefined if the agent is already at location A. The GOAL language also includes special actions for adding and removing goals from the goal base, but we do not discuss these actions here (see e.g. [7]). The change of the goal base is defined by means of the progression operator. At each step, all goals of the goal base are progressed.

Definition 5 *(Mental State Transformer $\mathcal{M}$)*
Let $\mathtt{a}$ be a basic action, $\phi \in \mathcal{L}_0$ and $\mathcal{T}$ be a transition function for basic actions. Then the *mental state transformer function* $\mathcal{M}$ is defined as a mapping from actions and mental states to updated mental states as follows:

$$\mathcal{M}(\mathtt{a}, \langle \Sigma, \Gamma \rangle) = \begin{cases} \langle \Sigma', Progress(\Gamma, \Sigma') \rangle & \text{if } \mathcal{T}(\mathtt{a}, \Sigma) = \Sigma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

As is noted in [1], the progression operator has to a certain extent the ability to model check formulas used here to express maintenance goals such as $\Box\chi$. In particular it is able to detect that a finite prefix of an execution path falsifies such goals. The progression operator however is not complete and will not always be able to detect for a particular goal that it can never be satisfied on extensions of all finite prefixes of all execution paths generated by the agent program. Similarly, it cannot be detected that an achievement goal $\Diamond\chi$ might never be achieved on an extension of a course of actions taken so far, nor can we use the operator to detect that a formula is unsatisfiable. The advantage, however, of giving up this component of completeness is computational efficiency; the progression of a formula can be computed in time linear in the size of the formula [1]. In addition, in the context of agent programming the expressivity gained by allowing temporal formulas arguably also provides for a more natural means of designing declarative agent programs. We see the investigation of the properties of the progression operator as an important issue for future research.

The specification of when a basic action may be executed, is done by means of action rules. An action rule c has the form **if** ψ **then** $\mathtt{a}$, with $\mathtt{a}$ a basic action. This action rule specifies that $\mathtt{a}$ may be performed if the mental state condition ψ holds and the transition function is defined for $\mathtt{a}$. In that case we say that action c is *enabled*. As an example, we can now formally write the action rule to goto a location as: **if** $\neg\mathbf{B}(at(x)) \wedge \mathbf{G}(\Diamond at(x))$ **then** $goto(x)$. Given that the agent believes it is still at its base location $at(Base)$ which implies $\neg at(A)$ and has a goal $\Diamond at(A)$ the agent can derive from this rule that $goto(A)$ is an action

it might perform (is enabled). During execution, a GOAL agent selects non-deterministically any of its enabled actions. This is expressed in the following transition rule, describing how an agent gets from one mental state to another.

Definition 6 *(Action Semantics)*
Let m be a mental state, and $c = \textbf{if } \psi \textbf{ then } \mathtt{a}$ be an action. The transition relation $\xrightarrow{c}$ is the smallest relation induced by the following transition rule.

$$\frac{m \models \psi \quad \mathcal{M}(\mathtt{a}, m) \text{ is defined}}{m \xrightarrow{c} \mathcal{M}(\mathtt{a}, m)}$$

The execution of a GOAL agent results in a *computation trace*. We define a trace as a sequence of mental states, such that each mental state can be obtained from the previous by applying the transition rule of Definition 6. As GOAL agents are non-deterministic, the semantics of a GOAL agent is defined as the *set* of possible computations of the GOAL agent, where all computations start in the initial mental state of the agent.

Definition 7 *(Agent Computation).* A computation trace, typically denoted by t, is an infinite sequence of mental states $m_0, m_1, m_2, \ldots$ such that for each i there is an action c_i and $m_i \xrightarrow{c_i} m_{i+1}$ can be derived using the transition rule of Definition 6, or $m_i \not\xrightarrow{c_i}$ and for all $j > i$, $m_j = m_i$. The meaning $R_{\mathcal{A}}(m_0)$ of a GOAL agent named $\mathcal{A}$ with initial mental state m_0 is the set of all computations starting in that state.

Observe that a computation is infinite by definition, even if the agent is not able to perform any action anymore from some point in time on. Also note that the concept of a computation trace is a general notion in program semantics that is not particular to GOAL. The notion of a computation trace can be defined for any agent programming language that is provided with a well-defined operational semantics. The semantics $R_{\mathcal{A}}(m_0)$ thus consists of all traces that may be generated by the agent program. These traces form the first layer of the RASA architecture.

4 Evaluating Temporal Formulas on Prefixes of Traces

In Sections 5 and 6, we will show how to define the second layer (hard constraints) and third layer (soft constraints) on top of the first layer as defined in the previous section. Both hard constraints and soft constraints will be represented using LTL. In standard LTL, formulas are evaluated on *infinite* traces. As explained in Section 2.2, however, in our setting an agent can lookahead a *finite* number of steps and we want to evaluate LTL formulas on such finite traces. We thus need to define the semantics of LTL formulas on such finite traces. This introduces several issues, one of which is the definition of the semantics of the next operator $\bigcirc$. It is not immediately clear what the semantics of a formula $\bigcirc\phi$ should be if it is evaluated in the last state of a finite trace.

For explaining our approach, it is important to realize that the finite trace on which we evaluate LTL formulas is only a *prefix* of a trace which will be continued beyond the lookahead horizon. Intuitively, the truth of a formula $\bigcirc\phi$ evaluated in the last state of such a finite prefix cannot be established. Its truth depends on how the trace continues beyond the finite prefix under consideration. A formula $\Diamond\phi$, on the other hand, clearly holds on a finite prefix if ϕ holds on some state of this prefix. Similarly, a formula $\Box\phi$ is clearly false on a finite prefix if $\neg\phi$ holds in some state of this prefix.

From these examples, we can see that the truth of an LTL formula evaluated on a finite prefix of a trace depends on how this trace progresses beyond the finite prefix. If a formula is false on any progression, it is also false on the finite prefix. Similarly, if it is true on any progression, it is true on the finite prefix. In all other cases, we cannot determine the truth of the formula. This intuition is reflected by a 3-valued semantics of LTL as presented in [3] in the context of monitoring. In the 3-valued semantics, the truth value of an LTL formula evaluated on a finite trace is $\top$ (true), $\bot$ (false), or "?" (unknown). We use (a slightly adapted version of) the definition of [3] for evaluating LTL formulas on finite prefixes. We define the semantics of LTL formulas over traces of mental states in terms of the semantics over belief bases as follows, where t^b is derived from t by keeping only the belief bases of each mental state: $t, i \models_{LTL} \chi \Leftrightarrow t^b, i \models_{LTL} \chi$.

Definition 8 *(3-valued LTL semantics)*
Let M be a set of mental states. Finite traces over M are elements of M^* and infinite traces are elements of M^ω. Both are typically denoted by t. Let $t, t' \in M^* \cup M^\omega$ be (finite or infinite) traces. Concatenation of traces t, t' is defined as usual and simply denoted as tt'; we stipulate that if $t \in M^\omega$, then $tt' = t$. The truth value of an LTL_3 formula φ with respect to a trace $t \in M^* \cup M^\omega$, denoted by $[t \models \chi]$, is an element of $\{\top, \bot, ?\}$ and defined as follows.

$$[t \models \chi] = \begin{cases} \top \text{ if } \forall t' \in M^\omega : tt' \models_{LTL} \chi, \\ \bot \text{ if } \forall t' \in M^\omega : tt' \not\models_{LTL} \chi, \\ ? \text{ otherwise.} \end{cases}$$

Corollary 1. *If* $t = \epsilon$, $[t \models \varphi] = ?$ *if* $\varphi \not\equiv \bot$ *and* $\varphi \not\equiv \top$.

The only difference between our definition and the one of [3] is that in our definition t can also be infinite. This allows us to investigate our semantics also for an infinite lookahead horizon in a uniform way. In [3], a technique based on the construction of a finite state machine from an LTL formula and a finite trace is presented for determining the truth value of an LTL_3 formula at runtime in implemented systems, which provides a basis for implementing Layers 2 and 3 of our architecture explained in the next sections.

The need for a 3-valued semantics in our approach highlights an important difference with the use of LTL in planning approaches such as [10,15]. In a planning context, the plans under consideration are always complete. That is, it is not taken into account that these plans might progress beyond their final state. This means that, e.g., a formula $\Diamond\phi$ in those approaches is considered to

be false if ϕ does not hold in some state resulting from execution of the plan, and true otherwise. In our semantics, on the other hand, the truth value of the formula is unknown if ϕ does not hold on some state of the finite prefix.

5 RASA Layer 2: Goals as Hard Constraints

Goals of an agent are temporal formulas. The semantics of agent programs provided in Section 3 accounts for the role such goals have in selecting actions using action selection rules. Action selection rules allow an agent to derive actions from its beliefs and goals in a *reactive* manner, but we argue that this layer in the architecture does not yet account for the full role that such goals can have in the action selection mechanism of a rational agent.

If an agent has the ability to *lookahead* a (finite) number of steps, it can also use its goals to avoid selecting those actions that prevent the realization of (some of) the agent's goals. In this section we define the second layer of RASA which accounts for this role of goals. Goals thus viewed introduce additional constraints on action selection to the effect of excluding those actions of which the agent foresees that they will not satisfy its goals. Here we consider such constraints to be *hard* constraints, meaning that any foreseen violation of goals by performing an action will force a rational agent to choose an alternative action (if possible) or become inactive (in line with previous work, cf. [13]). That is, actions will only be selected if for all that is known the action may still eventually allow satisfaction of the goals of the agent.

In this view of goals as hard constraints, achievement goals of the form $\Diamond\phi$ per se do not add any additional constraints on action selection since an agent with a finite lookahead horizon will not be able to conclude that an action will prohibit realizing such a goal. Achievement goals thus may be said to have no "selective force" beyond the rule-based mechanism of the action selection architecture. The role of a goal such as being at location A, $\Diamond at(A)$, thus is mainly to guide this selection process and, in order to avoid wasting resources, to remove such goals as reasons for action when they have been achieved (see the *Progress* operator of Section 3.2). Other types of goals such as maintenance goals do have a selective force in this sense; for example, if $\Box\phi$ is a goal of the agent, an agent can conclude that it is no longer possible to satisfy this goal if ϕ does not hold in some state within the lookahead horizon. An agent thus can use such goals to filter certain options for action generated by the rule-based layer one.[6] For example, the maintenance goal of having a minimum level of fuel in the tank of an agent will prevent selection of an action $goto(A)$ in case this would drop fuel levels below this minimum, assuming that going somewhere consumes fuel.

[6] One could argue that layer one should also make sure that such maintenance goals are not violated. However, trying to account for such general constraints on agent behaviour in the rules in an ad hoc manner will often lead to less understandable programs, which is why we argue for the separation of concerns provided by the proposed RASA.

Of course, the agent needs to consider possible future effects of a course of action allowed by the agent program to check such constraints, which is why a lookahead horizon needs to be defined to do so. The lookahead horizon is an integer specifying the length of a prefix of a possible trace of the agent program. Here the 3-valued semantics for LTL discussed in the previous section is particularly useful since it allows us to evaluate goals on finite prefixes of traces. The fact that an achievement goal $\Diamond\phi$ does not have selective force, is reflected by the fact that its truth value in the 3-valued LTL semantics will be "?" (the "unknown" value).

The filtering of action options by means of verifying whether finite prefixes of a trace of an agent program satisfy the goals of an agent changes the meaning of that agent program. It does not change the fact that an agent can continue performing actions infinitely. We next show how the semantics of an agent program can be defined to capture the effects of the second layer of the RASA formally in the context of GOAL. A *prefix* of a computation t is an initial finite sequence of t or t itself. A prefix of length n of a computation t is denoted by $t^{\langle n\rangle}$ with $n \in \mathbb{N}\cup\{\infty\}$, where $t^{\langle\infty\rangle}$ is defined as t. $\mathbb{N}$ is the set of natural numbers including 0, and ∞ is the first infinite ordinal. The lookahead horizon, i.e., the number of execution steps that an agent can lookahead, is denoted by h. The idea is that we take the set of possible execution traces $R_{\mathcal{A}}$ that may be selected according to the first layer, and filter out those traces that do not satisfy one or more of the agent's goals, when looking ahead h steps from some state on the trace.

We define a filter function $\sigma^h_{\mathcal{A}}$ inductively on the set of traces $R_{\mathcal{A}}$ and on the time point i on such a trace. An agent will use its goals to restrict selection of actions from the start, i.e. time 0, which explains why the base case of the inductive definition starts at -1. The base case defines the starting point of traces $R_{\mathcal{A}}$ that need to be filtered. For technical reasons the filter function is defined as two different components, and in addition to σ we define a function ς. The idea is that the filter function $\sigma^h(i)$ returns all (infinite) traces that do not violate the goals of the agent on a finite prefix of length h starting from state i, while the function $\varsigma^h(i)$ returns all (finite) prefixes of traces that do not violate the goals of an agent given a lookahead capability of h until the end of that prefix but that do violate some goal in any possible next state.

Definition 9 *(Goal Filter Functions σ and ς)*
Let $\mathcal{A}$ be some agent with meaning $R_{\mathcal{A}}$ and let $h \in \mathbb{N}$ be a horizon. Let $t[i]$ be the tail of trace t starting in the i-th state of t. Then the *goal filter functions* $\sigma^h_{\mathcal{A}}$ and $\varsigma^h_{\mathcal{A}}$ are defined by simultaneous induction as follows:

$$\begin{aligned}
\sigma^h_A(-1) &= R_{\mathcal{A}},\\
\sigma^h_A(i) &= \{t \in \sigma^h_A(i-1) \mid \forall \chi \in \Gamma^t_i : [t[i]^{\langle h\rangle} \models_{LTL} \chi] \neq \bot\}
\end{aligned}$$

$$\begin{aligned}
\varsigma^h_{\mathcal{A}}(-1) &= \emptyset,\\
\varsigma^h_{\mathcal{A}}(i) &= \varsigma^h_{\mathcal{A}}(i-1) \cup \{t^{\langle i\rangle} \mid t \in \sigma^h_A(i-1), \exists \chi \in \Gamma^t_i : [t[i]^{\langle h\rangle} \models \chi] = \bot\}, \text{ for } i \geq 0
\end{aligned}$$

To avoid complicating the definition of the function ς it is defined slightly too general and includes prefixes that do have continuations that satisfy all of an agent's goals. In order to eliminate these prefixes and keep only maximal prefixes of traces that cannot be extended given the agent's goals we additionally introduce a function $maxPrefix$. Let $t \sqsubset t'$ denote that t is a strict prefix of t' and T a set of (in)finite traces. Then $t \in maxPrefix(T)$ iff there is no $t' \in T$ such that $t \sqsubset t'$. Using the definitions of the goal filter function(s), we can now provide a simple definition of the semantics of traces induced by the second layer in the action selection architecture. The meaning of an agent at this layer is denoted by $H_{\mathcal{A}}$ and defined as the maximal elements of the limit of the filter functions.

Definition 10 *(Semantics of GOAL Agent with Goals as Hard Constraints)* The meaning of a GOAL agent H_A that applies hard constraints in addition to its rule-based action selection is defined by:

$$H_{\mathcal{A}} = maxPrefix\big(\bigcap_{i=-1}^{\infty} \sigma_{\mathcal{A}}^{h}(i) \cup \bigcup_{i=-1}^{\infty} \varsigma_{\mathcal{A}}^{h}(i)\big)$$

It should be clear from the definition of the meaning of a GOAL agent that the semantics introduced clearly distinguishes between the different layers of RASA. Moreover, given the computational properties of LTL_3 the semantics of the second layer can be realized computationally if the initial prefixes of length h of the traces induced by the first layer can be efficiently generated. In the next section we introduce the third layer to complete our formal picture of the informal RASA discussed in Section 2.

6 RASA Layer 3: Preferences as Soft Constraints

In the third layer of RASA an agent aims to select those traces that maximize satisfaction of its preferences. Whereas the second layer *eliminates* any action options that would lead to violation of a goal, the third layer only eliminates action options if *more preferred alternatives* are available. An agent thus might prefer to have 60 units of fuel minimally in its tank but in case there are no actions enabled (as determined by the agent program) that would allow the agent to satisfy this preference it would still select one of these actions; contrast this with a maintenance goal which would prevent the agent from doing anything at all in this case (in case there would not be an option to refuel). Similarly to goals, preferences are expressed using LTL. For example, preferring to go first to location A before going to location B can be expressed by $\neg(\neg at(A) U at(B)) \wedge \Diamond at(B)$. Since LTL formulas express properties of traces, this allows an agent to express that it prefers one way of achieving a goal, i.e., one particular trace, over another, if multiple courses of action for realizing the goal are available. Such preferences are represented by a so-called *preference structure*. A preference structure consists of a sequence of LTL formulas.

Definition 11 *(Preference Structure)*
A *preference structure* Ψ is a sequence $(\chi_1, \ldots, \chi_n)$ of LTL formulas.

This preference structure expresses that traces on which some χ_i is satisfied are preferred over traces on which χ_i is not satisfied, and the satisfaction of χ_i is preferred over the satisfaction of χ_j for $j > i$. That is, if χ_1 is satisfied on a trace t but not on another trace t', t is preferred over t'. If both t and t' do not satisfy χ_1, but t satisfies χ_2 and t' does not, t is again preferred over t', etc. This interpretation of the preference structure thus induces a *lexicographic* ordering on traces.

This ordering can formally be defined as follows. We use Ψ_t to denote the sequence $(b_1, \ldots, b_n)$, where $b_i = [t \models \chi_i]$ for $1 \leq i \leq n$, i.e., $b_i \in \{\top, \bot, ?\}$; we use Ψ_t^i to denote b_i. We now use the ordering $\bot < ? < \top$ to define a lexicographic preference ordering on traces on the basis of a preferences structure. That is, if for a χ_i of the preference structure we have $[t \models \chi_i] = \top$, this is better than when $[t \models \chi_i] = ?$, which is again better than when $[t \models \chi_i] = \bot$.

Definition 12 *(Lexicographic Preference Ordering)*
Let $\Psi = (\chi_1, \ldots, \chi_n)$ be a preference structure and $t, t' \in M^* \cup M^\omega$ be finite or infinite traces. Then we say that trace t is (lexicographically) preferred over t' with respect to Ψ, written $t \prec_\psi t'$, iff:

$$\exists 1 \leq j \leq n : \forall 1 \leq i < m : (l_t^i = l_{t'}^i \text{ and } l_t^j < l_{t'}^j)$$

We also write $t \preccurlyeq_\psi t'$ if $t \prec_\psi t'$ or $\psi_t = \psi_{t'}$.

The main advantage of using a lexicographic preference order $\prec_\psi$ is that such a preference order on traces is a total preorder, which means that any two traces are either equally good with respect to the preference order, or one is better than the other. Other kinds of preference orderings such as those discussed in [6] may be considered, but investigating this is left for future research.

As an example, given that the preference to go first to location A before goint to B is ranked higher than the preference to keep a minimum fuel level of 60, even if refuelling would be possible while going to B and not while going to A, the agent would choose to first go to A using the lexicographic preference ordering.

Our use of a lexicographic preference order is inspired by the work of [10,15]. There are, however, many differences with [10,15] and our approach. Instead of integrating preferences into the situation calculus as in [10] we propose a uniform framework for goals and preferences that is integrated into a programming framework for rational agents, and instead of compiling preferences into certain programs we have taken a more direct approach by defining a layered action selection architecture that is made precise by means of a formal semantics. Finally, we use three-valued LTL to evaluate goals and preferences on (prefixes of) traces.

Preferences are progressed or updated through time, just like goals. The idea is that a preference such as $p \wedge \bigcirc q \wedge \Diamond r$ has been satisfied when now p holds, in the next state q and possibly sometime thereafter r holds. Such a preference cannot be satisfied anymore when p is not true currently, and to express this we use the *Progression* operator of Section 3.2 to keep track of such facts. Since

preferences are progressed during execution of the agent, we extend mental states of an agent with the agent's preference structure.

Definition 13 *(Mental State with Preferences)*
Let $m = \langle \Sigma, \Gamma \rangle$ be a mental state, and Ψ a preference structure. A mental state with preferences then simply is the tuple $\langle \Sigma, \Gamma, \Psi \rangle$.

Definition 14 *(Progression of Preference structure)*
The progression of a preference structure $\Psi = (\chi_1, \ldots, \chi_n)$ from a mental state $\langle \Sigma, \Gamma, \Psi \rangle$ to a new state $\langle \Sigma', \Gamma', \mathit{Progress}(\Psi, \Sigma') \rangle$ is simply defined as the progression of each of the individual preference in the structure, i.e.

$$\mathit{Progress}(\Psi, \Sigma') = (\mathit{Progress}(\chi_1, \Sigma'), \ldots, \mathit{Progress}(\chi_n, \Sigma'))$$

The semantics of action execution of Definition 6 is changed accordingly as follows, where $m = \langle \Sigma, \Gamma, \Psi \rangle$ and $c = \textbf{if } \psi \textbf{ then } \mathtt{a}$ is an action:

$$\frac{m \models \psi \quad \mathcal{M}(\mathtt{a}, m) = \langle \Sigma', \Gamma' \rangle}{\langle \Sigma, \Gamma, \Psi \rangle \xrightarrow{c} \langle \Sigma', \Gamma', \mathit{Progress}(\Psi, \Sigma') \rangle}$$

The main difference between goals and preferences in our approach is that goals are used as hard constraints to guide action selection whereas preferences are used as soft constraints. That is, an agent would like to satisfy all of its preferences but if this is not possible it will simply choose to satisfy those that are most preferred, if any can be satisfied at all. The preference order introduced above can be used for this purpose. In the third layer of RASA those traces are selected from the remaining ones (those that survived filtering by layer 2) that are maximal elements in this order. As with the second layer, the third layer of RASA modifies the meaning of an agent program. In order to specify the effects of this layer on the semantics of agent programs, we introduce some notation again. We use $\mathit{max}(T, \prec_\psi)$ to denote the maximal elements of a set of traces T under the lexicographic preference order $\prec_\psi$, induced by the preference structure Ψ. The preference filter function of layer 3 in our action selection architecture can then be defined similarly to that of the goal filter function.

Definition 15 *(Preference Filter Function ψ)*
Let $\mathcal{A}$ be some agent using layer 2 goal filtering with meaning $H_\mathcal{A}$ and let $h \in \mathbb{N}$ be a horizon. Then the *preference function* $\psi^h_\mathcal{A}$ is defined by induction as follows:

$$\begin{aligned} \psi^h_A(-1) &= H_\mathcal{A}, \\ \psi^h_A(i) &= \{t \mid t^{\langle i+h \rangle} \in \mathit{max}(\{t[i]^{\langle h \rangle} \mid t \in \psi^h_A(i-1)\}, \prec_\psi)\} \end{aligned}$$

Similar to Section 5, the semantics of an agent that uses all three layers of the action selection architecture is defined as the limit of the preference filter function over the traces obtained from layer 2. The definition is somewhat simpler since an agent can always continue on a given trace since it only needs to select a maximum continuation of its past action performance but does not have to stop acting altogether due to a potential violation of a hard constraint. This is one of the main differences between layer 2 and 3.

Definition 16 *(Semantics of GOAL with Preferences)*
The meaning of a GOAL agent P_A that applies soft constraints with horizon h in addition to the action selection mechanisms of layer 1 and 2 is defined by:

$$P_{\mathcal{A}} = \bigcap_{i=-1}^{\infty} \psi_{\mathcal{A}}^{h}(i)$$

Definition 16 completes the specification of each of the three layers of the rational action selection architecture. The informal discussion of the architecture in which we distinguished three layers of action selection is mirrored in respectively Definitions 7, 10, and 16. The formal approach has shown that it is feasible to define a transparent and rich RASA into an agent programming language, that integrates beliefs, goals and preferences as tools for choosing the right action. In particular, the approach offers a uniform framework based on temporal logic to express goals and preferences.

7 Conclusion and Related Work

In this paper, we have proposed a layered rational action selection architecture which specifies a rational action selection mechanism in which hard and soft constraints are integrated. We have shown how the proposed architecture can be formalized in the context of the GOAL agent programming language and how it modifies the semantics of agent programs. In order to obtain a uniform framework that naturally allows for the integration of hard and soft constraints including such concepts as achievement goals, maintenance goals, as well as (temporally extended) preferences, we have used linear temporal logic for the representation of goals and preferences.

The way in which we use temporal logic is inspired by work in planning where temporal logic is used to "guide" planners through the search space [1] and to select preferred plans [10,2,15]. One of the differences is that the RASA is defined for agent programming languages and, in contrast with planners that would compare *complete plans* against the available constraints, these constraints are taken into account continuously *during* execution of an agent program. Technically, this has resulted in the use of 3-valued LTL for the realization of RASA in GOAL.

The practical realizability of our approach is partly facilitated by the use of techniques, in particular the progression algorithm of [1] and the implementation of 3-valued LTL of [3], that have been shown to be implementable. The investigation of restrictions of the LTL language for representing goals to make checking of entailment of a goal from the goal base feasible, is left for future research.

Acknowledgements

We would like to thank Moritz Hammer for pointing us to [3].

References

1. Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artificial Intelligence 166 (2000)
2. Baier, J.A., Bacchus, F., McIlraith, S.A.: A heuristic search approach to planning with temporally extended preferences. In: IJCAI, pp. 1808–1815 (2007)
3. Bauer, A., Leucker, M., Schallhart, C.: Monitoring of real-time properties. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 260–272. Springer, Heidelberg (2006)
4. Bienvenu, M., Fritz, C., McIlraith, S.A.: Planning with qualitative temporal preferences. In: Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), pp. 134–144 (2006)
5. Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A.: Multi-Agent Programming: Languages, Platforms and Applications. Springer, Berlin (2005)
6. Brewka, G.: A rank based description language for qualitative preferences. In: Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), pp. 303–307 (2004)
7. de Boer, F., Hindriks, K., van der Hoek, W., Meyer, J.-J.: A Verification Framework for Agent Programming with Declarative Goals. Journal of Applied Logic 5, 277–302 (2007)
8. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science. Formal Models and Semantics, vol. B, pp. 996–1072. Elsevier, Amsterdam (1990)
9. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 189–208 (1971)
10. Fritz, C., McIlraith, S.A.: Decision-theoretic golog with qualitative preferences. In: KR, pp. 153–163 (2006)
11. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann, San Francisco (2004)
12. Hindriks, K.: Modules as Policy-Based Intentions: Modular Agent Programming in GOAL. In: Dastani, M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS, vol. 4908, pp. 156–171. Springer, Heidelberg (2008)
13. Hindriks, K., van Riemsdijk, B.: Satisfying Maintenance Goals. In: Baldoni, M., Son, T.C., van Riemsdijk, M.B., Winikoff, M. (eds.) DALT 2007. LNCS, vol. 4897, pp. 86–103. Springer, Heidelberg (2008)
14. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.-J.C.: Agent Programming with Declarative Goals. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS, vol. 1986, pp. 228–243. Springer, Heidelberg (2001)
15. Son, T.C., Pontelli, E.: Planning with preferences using logic programming. Theory and Practice of Logic Programming 6(5), 559–607 (2006)

Strategic Agent Communication: An Argumentation-Driven Approach

Jamal Bentahar[1], Mohamed Mbarki[2], John-Jules Ch. Meyer[3], and Bernard Moulin[2]

[1] Concordia University, Concordia Institute for Information Systems Engineering (CIISE), Canada
bentahar@ciise.concordia.ca
[2] Laval University, Department of Computer Science and Software Engineering, Canada
{mohamed.mbarki,bernard.moulin}@ift.ulaval.ca
[3] Utrecht University, Department of Information and Computer Science, The Netherlands
jj@cs.uu.nl

Abstract. This paper proposes a formal framework for agent communication where agents can reason about their goals using strategic reasoning. This reasoning is argumentation-based and enables agents to generate a set of strategic goals depending on a set of constraints. Subgoals are generated using this reasoning and they can be cancelled or substituted for alternatives during the dialogue progress. An original characteristic of this framework is that agents can use this strategic reasoning together with a tactic reasoning to persist in the achievement of their goals by considering alternatives depending on a set of constraints. Tactic reasoning is responsible of selecting the communicative acts to perform in order to realize the strategic goals. Some constraints are fixed when the conversation starts and others during the dialogue progress. The paper also discusses the computational complexity of such a reasoning.

Keywords: Agent Communication, Social Commitments, Argumentation, Strategic Reasoning, Complexity.

1 Introduction

In multi-agent systems, agents are designed to accomplish particular tasks and they should be able to generate, adopt, drop and achieve their goals [22,35,37]. In the modern research into this field, agents are equipped with reasoning capabilities expressed in computational logics [4,15,19,31,33]. These agents often have to interact with each other in order to achieve their goals that are subject to a set of constraints, which can be revised during the dialogue. However, in the most recent approaches of goals modeling (e.g. [1,10,27,28,37]) considering constraints and their dynamics is often neglected.

M. Baldoni et al. (Eds.): DALT 2008, LNAI 5397, pp. 233–250, 2009.

The aim of this paper is to take further step toward strategic agent communication using argumentative reasoning. We propose a formal framework for communicating agents that are able to reason about goals using strategic reasoning. These goals, called *strategic goals*, could be achieved by identifying some sub-goals and prospective alternatives. In addition, using a tactic reasoning, agents can select the communicative acts to perform in order to realize the strategic goals considering a set of constraints. This reasoning is based, as for human beings, on psychological and philosophical approaches which are not yet formalized (see for example [5,34]). These approaches suppose that agents use strategic reasoning, which guides their participation in dialogues. Such a reasoning helps agents to create *dialogue strategies*. More precisely, our framework allows agents to generate a set of sub-goals depending on a set of constraints and to find alternatives of these sub-goals if they cannot be achieved. These sub-goals must be justified through argumentation using agents' beliefs and the information made public during the conversation.

The generation of goals and satisfaction of constraints are thus supported by arguments. The motivation behind using argumentation is that this technique has been proven to be efficient when reasoning about incomplete and inconsistent information, which is generally the case in agent communication [6,9,13,32]. Also, argumentation is a kind of non-monotonic reasoning, and when new information arise through conversation, new arguments can be built and some old arguments can become invalid (because attacked by the new arguments and cannot be defended). The idea is to use argumentation in order to give agents the possibility to modify or reject their sub-goals and follow up or revise the strategy they identify. Alternative sub-goals can also be considered using argumentation in order to enable agents to persist in the achievement of their goals, even in the case where some sub-goals are canceled. Although persistence phenomenon is important to ensure the conversation success, such a phenomenon is not addressed by the current communication models (e.g. [1,14,18,26,27,28]). The purpose of this paper is to address this issue.

In this paper, we illustrate our model by an example of negotiating cars between two agents. We suppose that an agent Ag_1 (seller) tries to convince an agent Ag_2 (buyer) to buy a car. In this example, the seller's goal, noted by B, is *the sale of a car*. An example of constraint the seller should consider is: *the price must be higher than* 10.000 $dollars$, and an example of constraint the buyer should satisfy is *the car should not be manufactured in the country* X.

This approach is fundamentally different from hierarchical planning and procedural dynamic and continuous planning approaches such as those proposed in [17,21], which are based on making plans by task decompositions, keeping and incrementally modifying alternative plans during the execution, and switching to an alternative plan when necessary. The main difference is that in our approach the sub-goals are not plans but represent a strategy that agents can follow in order to achieve their goals. This strategy is subject to a set of constraints the agent should satisfy. Also, the mechanism behind generating sub-goals and alternatives and selecting constraints to be satisfied is argumentation-based, which

allows agents to reason about the alternatives by considering these constraints. Such an argumentation-based reasoning helps agents to decide about the strategy to be followed and to readapt this strategy when new information arise. This is more flexible than procedural planning-based approaches, and then more suitable for autonomous agents. Our strategic reasoning is close to the *anticipation behavior* [24]. This is similar to *chess game* in which players start by some strategies that determine the first moves to play (generally between 5 and 9 moves), and the upcoming moves are decided by considering the current game situation. Also, although this approach and planning with declarative goals seem to be similar and have many features in common, they are different in the sense that our proposal focuses more on strategy and tactic issues by using argumentation. In our context of agent communication, this is achieved by reasoning on the agent's beliefs and what the addressee made public during the past states of the conversation. In this context, argumentation is a suitable technique as it guides the selection of new sub-goals and new constraints depending on the new available information.

The contributions of this work are:

1. The proposal of a new declarative approach for agent communication allowing agents to reason about the strategies they should follow before and during the conversation. Such a strategic reasoning enables agents to calculate a cognitive representation of the manner of achieving a goal in terms of strategic goals. Also, the approach is argumentation-based, which allows agents to reason about incomplete and inconsistent information and to select *justified* goals and sub-goals.
2. The formalization of the goal persistence and the computing of the initial constraints associated with each goal and the constraints which can be generated during the dialogue. This allows the persistence in achieving the goals by using alternative goals.
3. A complexity analysis of the underlying reasoning.

The idea ultimately is to go beyond the current approaches into agent communication focusing more on protocols [3,14]. Our purpose is to advance research in this field by moving from protocol specifications to strategy and tactic reasoning [8,18].

Paper Overview. This paper is organized as follows. In Section 2, we introduce the fundamental ideas of our argumentation-based agent communication approach. In particular, we define the strategic reasoning used by this approach. In Section 3, we present our formal framework allowing reasoning about goals. We present and discuss the notions of constraints, strategic goals and possible alternatives. We also address the complexity issue of this framework. Before concluding, we compare our approach to some related work in Section 4.

2 Agent Communication Approach

In our agent communication approach, we distinguish between the agent model and the dialogue model (or the conversational model). The agent model is mainly

based on the agents' mental states. The conversational model is based on the philosophical concept of social commitment (SC) [7,11,29,38] and argumentation. This model enables agents to support their social commitments or to attack those of the other participants using arguments. These two models are illustrated in Fig.1 and are detailed later.

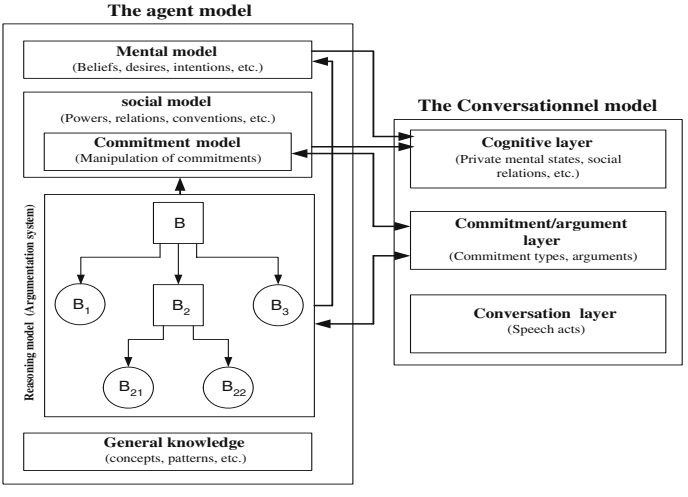

Fig. 1. The links between the agent architecture and communication model

2.1 Agent Model

Based on its mental states and other types of information (coming essentially from the social context and the conversational context), an agent can have a global vision about how to achieve its goals (here we are considering conversation goals, which are goals an agent can achieve by communicating, for example convincing another agent to adopt some opinions). This vision is considered as a strategy. Our work is based on psychological and philosophical approaches which are not yet formalized [5,34]. In philosophy, and according to van Dijk [34], a dialogue strategy is defined as *a global cognitive representation of the manner of achieving some goals*. The strategy concept was also dealt with by Bange [5] who considers that a strategy enables one to associate the intentional element with the cognitive element of the action. In his opinion, *a strategy consists in choosing a certain number of intermediate and subordinate goals whose realization through partial actions may lead in an adequate way to the realization of the final goal*. This notion of intermediate and subordinate goals is similar to the concept of *landmarks* proposed in [30].

Consequently, we define a dialogue strategy as a function associating a goal to a set of sub-goals, which we call *strategic goals*. These sub-goals are selected and arranged in order to achieve the conversation goal. The realization of this goal is thus conditioned by the realization of its strategic sub-goals. Other

sub-goals could be determined from the existing goals depending on the current state of conversation. Let us consider our negotiation example introduced in the introduction.

Example 1. The seller agent can choose the strategy of achieving three strategic goals B_1, B_2 and B_3 in order to realize the goal B, which is *selling the car*. These sub-goals can be defined as follows and the method of choosing the strategic goals is detailed in Section 3.

B_1 = *"Know how much the buyer would like to invest in the purchase of a car as well as his preferences".*

B_2 = *"Propose a car which could interest the buyer".*

B_3 = *"Convince the buyer to accept this proposal".*

To achieve the goals B_1 and B_3, the agent strategy consists of trying to achieve the sub-goals B_{11}, B_{12}, B_{31}, and B_{32}. These new sub-goals are defined as follows.

B_{11} = *"Know the model of the car preferred by the buyer".*

B_{12} = *"Know how much the buyer would like to invest".*

B_{31} = *"Convince the buyer that the price of the proposed car is reasonable".*

B_{32} = *"Convince the buyer that the proposed car consumes like small cars on the long distances".*

We represent the set of agent's goals by a tree. The root of the tree represents the main goal (designated by a square), the nodes represent strategic goals (also designated by squares), and the leaves represent elementary goals (designated by circles) which can be reached by performing speech acts. For each defined strategic goal, it might be possible to generate some alternatives. Therefore, an agent might have several strategies to achieve the same goal. In our example, we suppose that the seller is unable to convince the buyer to accept its offer (i.e. the seller is unable to achieve the goal B_3 by using elementary actions) because of a new constraint. In this case, the seller agent can persist in achieving its goal by considering a prospective alternative of the goal B_2. For example, the seller agent may propose another car which could satisfy the new buyer's interest, and this new bid will be an alternative goal for B_2, named B'_2.

Example 2. Moreover, if the seller agent finds during the dialogue progress that the buyer agent is not interested in the fact that the car is economic, then it may suggest an alternative to the strategic goal B_{32}, denoted B'_{32}. For example, the seller will try to convince the buyer that the spare parts for the proposed car are available and not expensive. The strategic goals B'_2 and B'_{32} may be defined as follows.

B'_2 = *"Propose another car which could satisfy a new buyer's interest".*

B'_{32} = *"Convince the buyer that the spare parts of this car are available and not expensive".*

To achieve a same goal, an agent can have several alternative strategies depending on the subset of constraints the agent decide to satisfy. The main goal, sub-goals, and constraints can be expressed in a logical language. The set of constraints may be inconsistent. However, the subset of constraints the agent decide to satisfy should be consistent.

2.2 Dialogue Model

Our communication model is based on the agent architecture suggested in [7]. This model is composed of three layers: mental, social, and reasoning layers. The mental layer includes beliefs, desires, goals, etc. The social layer captures social concepts such as SCs, conventions, roles, etc. Agents use their reasoning capabilities to reason about their mental states and the social concepts. The agent's reasoning capabilities are represented by the reasoning layer using an argumentation system (see Fig.1). Several argumentation theories and frameworks have been proposed in the literature (see for example [9,12,20,25]). However, only few frameworks do consider goals generation and strategic reasoning [2]. An adaptation of these frameworks is needed, particularly in terms of adding sub-goals generation and constraints dynamics. In this paper, we use an argumentation approach based on propositional logic like the one used in [7]. Such an approach allows agents to reason about the internal structure of arguments in terms of premises and conclusion. This approach comprises essentially a logical language, a definition of the argument concept, a definition of the attack relation between arguments, and finally a definition of acceptability [6,13]. In our approach, we distinguish between *internal arguments* and *external arguments*. Internal arguments are used to manage the incoherences of the agent's beliefs. However, external arguments are used to manage the incoherences between the agent's beliefs and the information transmitted by the addressee. Thus, an agent would be able to support the facts on which it is committed and to justify its communicative acts. In this paper, argumentation is used within strategic and tactic reasoning in terms of the generation of sub-goals to be achieved in order to achieve the main goal and in terms of the constraints to be considered.

3 Formal Framework of the Strategic Reasoning and Its Complexity

3.1 Argumentation-Based Strategic Goals

A strategic goal can have one or more alternatives, and the replacement of this strategic goal by one of its alternatives enables agents to achieve the same main goal (the conversation goal), but with different *constraints*. The subset of constraints to be satisfied and the subset of sub-goals to be realized in order to achieve the conversation goal determine the adopted strategy. In our framework, goals and constraints are propositional formulas expresed in some propositional languages. The selection of a set of sub-goals to be achieved must be supported by internal arguments. For this reason, we use the *explanatory argument concept*,

inspired by Amgoud and Kaci [1], and we define a new concept: the *realization argument*. A given goal can be supported by these two types of arguments. On one hand, the explanatory arguments justify the choice of the strategic goals in terms of agent's beliefs and the information made public during the conversation. On the other hand, the realization arguments determine the set of strategic goals necessary to achieve a goal. We define in this section an argumentation-driven framework to generate the set of constraints related to a strategic goal. We also define the generation of the strategic goals and their alternatives in order to achieve the conversation goal, while respecting the set of constraints related to this goal. In the rest of the paper, Γ indicates a possibly inconsistent knowledge base (beliefs, goals, ...) with no deductive closure, Δ indicates a possibly inconsistent constraint base with no deductive closure, and $\vdash$ stands for classical inference.

Definition 1 (*Explanatory Argument*). *An explanatory argument of an agent Ag is a pair* $(H, G_{Ag}h)$ *where* $G_{Ag}h$ *is an Ag's goal expressed as a formula in a logical language* $\mathcal{L}$ *and* H *is a subset of* Γ *such that: i)* H *is consistent, and ii)* $H \vdash G_{Ag}h$. H *is called the support of the argument which justifies the choice of the goal* $G_{Ag}h$.

Definition 2 (*Minimal Explanatory Argument*). *An explanatory argument of an agent Ag* $(H, G_{Ag}h)$ *is minimal iff* H *is minimal (i.e. there is no subset of* H *which satisfies i and ii of Definition 1.*

Definition 3 (*Realization Argument*). *A realization argument of an agent Ag is a triplet* $(\Phi_G^{Ag}, G_{Ag}h, C)$ *where* Φ_G^{Ag} *is a finite set of Ag's goals* $(\Phi_G^{Ag} \subseteq \Gamma)$, $G_{Ag}h$ *is an Ag's goal, and* $C \subseteq \Delta$ *is a finite set of constraints such that: i) all the goals of* Φ_G^{Ag} *are supported by (minimal) explanatory arguments, ii)* $\Phi_G^{Ag} \cup C$ *is consistent, and iii)* $\Phi_G^{Ag} \cup C \vdash G_{Ag}h$. Φ_G^{Ag} *is called the support of the argument (i.e. the set of the strategic goals necessary to the realization of the goal* $G_{Ag}h$*).*

Definition 4 (*Minimal Realization Argument*). *A realization argument of an agent Ag* $(\Phi_G^{Ag}, G_{Ag}h, C)$ *is minimal iff* $\Phi_G^{Ag} \cup C$ *is minimal (i.e. there is no subset of* $\Phi_G^{Ag} \cup C$ *which satisfies i, ii and iii of Definition 3.*

Definition 5 (*Attack Relation between Explanatory Arguments*). *Let* $(H, G_{Ag}h)$ *and* $(H', G_{Ag'}h')$ *be two explanatory arguments of two agents Ag and* Ag' *respectively.* $(H, G_{Ag}h)$ *attacks* $(H', G_{Ag'}h')$ *iff* $H \vdash \neg G_{Ag}h$.

Definition 6 (*Attack Relation between Realization Arguments*). *Let* $(\Phi_G^{Ag}, G_{Ag}h, C)$ *and* $(\Phi_G'^{Ag'}, G_{Ag'}h', C')$ *be two realization arguments of agents Ag and* Ag' *respectively.* $(\Phi_G^{Ag}, G_{Ag}h, C)$ *attacks* $(\Phi_G'^{Ag'}, G_{Ag'}h', C')$ *iff* $\Phi_G^{Ag} \cup C \vdash \neg G_{Ag'}h'$.

The *acceptability* of these two types of arguments can be defined in the same way as in [13]. Simply put, an argument is *acceptable* w.r.t a set of free-conflict arguments S (i.e. a set of arguments that are not attacking each other) iff if this

argument is attacked by another one, there exists an argument in the set S that attacks the attacker. Other acceptability semantics are defined in [13] such as *grounded* and *preferred* semantics, and using them will not change the core of this work.

The set Φ_G^{Ag} represents the sub-goals that Ag can use to achieve the goal $G_{Ag}h$ while satisfying a set of constraints C ($C \subseteq \Delta$). So, the problem is: given an agent goal ($G_{Ag}h$) and a set of constraints C, what are the sub-goals Φ_G^{Ag} to realize in order to achieve this goal. In the following propositions we consider the complexity of this problem. For the concepts of complexity theory, refer to [16,23].

Proposition 1. *Given a subset $H \subseteq \Gamma$ and an Ag's goal $G_{Ag}h$. Deciding whether $(H, G_{Ag}h)$ is a non-necessarily minimal argument is in $P^{\|NP}$ (P with parallel queries to NP).*

Proof. According to Definition 1, $(H, G_{Ag}h)$ is a non-necessarily minimal argument iff H is consistent and $H \vdash h$. Since consistency checking is NP-complete and establishing proof in propositional logic is co-NP-complete, the problem is in Δ_2^P. Because the two conditions are independent, they can be checked in parallel. Consequently the problem is in $P^{\|NP}$. ■

Proposition 2. *Let Γ be a knowledge base and $G_{Ag}h$ a conclusion. Deciding if there is an explanatory argument over Γ $(H, G_{Ag}h)$ is in Σ_2^P.*

Proof. The following algorithm resolves the problem: 1) guess a subset $H \subseteq \Gamma$; 2) check if $(H, G_{Ag}h)$ is an explanatory argument. From Proposition 1, this problem is in Σ_2^P. ■

Proposition 3. *Let $(H, G_{Ag}h)$ be an explanatory argument. Deciding whether $(H, G_{Ag}h)$ is minimal or not is in $P^{\|NP}$.*

Proof. The following algorithm resolves the problem: $\forall$ *subformula* $x \in H$, check if $H - \{x\}$ is consistent and $H - \{x\} \vdash h$. If H consists of n symbols, then it has no more than n sub-formulae. Thus, from Proposition 1 checking the minimality can be done with a polynomial number of parallel calls to an NP-oracle. It follows that this problem is in $P^{\|NP}$. ■

As a consequence of Proposition 3, the minimality criterion is not an additional source of complexity.

Proposition 4. *Given a finite set of Ag's sub-goals Φ_G^{Ag}, an Ag's goal $G_{Ag}h$, and a finite set of constraints C. Deciding if $(\Phi_G^{Ag}, G_{Ag}h, C)$ is a realization argument is in $P^{\|\Sigma_2^P}$.*

Proof. According to Definition 3, $(\Phi_G^{Ag}, G_{Ag}h, C)$ is a realization argument if All the goals of Φ_G^{Ag} are supported by explanatory arguments. If $|\Phi_G^{Ag}| = n$, then from Proposition 2, n calls to a Σ_2^P-oracle are needed. Because each sub-goal could be checked independently from the others, the n calls could be done in parallel. From Propositions 1, 2, and 3, the other conditions could be checked in $P^{\|NP}$. It follows that the decision problem is in $P^{\|\Sigma_2^P}$. ■

Theorem 1. *Let Γ be a knowledge base, $G_{Ag}h$ a conclusion, and C a set of constraints. Deciding if there is a realization argument over Γ $(\Phi_G^{Ag}, G_{Ag}h, C)$ is in Σ_3^P.*

Proof. The following algorithm resolves the problem: 1) guess a subset $(\Phi_G^{Ag} \subseteq \Gamma$; 2) check if $(\Phi_G^{Ag}, G_{Ag}h, C)$ is a realization argument. From Proposition 4, this problem is in Σ_3^P. ■

By using explanatory and realization arguments, we can define the *strategic goals* and their possible *alternatives* in order to achieve a given goal.

Definition 7 (*Strategic Goal*). *Let Φ_G^{Ag} be a finite set of Ag's goals, C be a finite set of constraints, and $StrG(G_{Ag}h)$ be a set of strategic goals (i.e. sub-goals) necessary to realize a given goal $G_{Ag}h$. $G_{Ag}h'$ is a strategic goal of $G_{Ag}h$ $(G_{Ag}h' \in StrG(G_{Ag}h))$ iff there is an acceptable realization argument $(\Phi_G^{Ag}, G_{Ag}h, C)$ such that: $G_{Ag}h' \in \Phi_G^{Ag}$.*

The fact that Φ_G^{Ag} is minimal makes $G_{Ag}h'$ necessary for the realization of $G_{Ag}h$. However, $G_{Ag}h'$ could be substituted for another sub-goal called alternative goal.

Definition 8 (*Alternative Goal*). *Let Φ_G^{Ag} be a finite set of Ag's goals, C be a finite set of constraints, and $AltG(G_{Ag}h_i/G_{Ag}h)$ be a set of alternative goals of a strategic goal $G_{Ag}h_i$ relative to a given goal $G_{Ag}h$. $G_{Ag}h_j$ is an alternative goal of $G_{Ag}h_i$ $(G_{Ag}h_j \in AltG(G_{Ag}h_i/G_{Ag}h))$ iff:*

1. *$(\Phi_G^{Ag}, G_{Ag}h, C)$ is an acceptable realization argument such that $G_{Ag}h_i \in \Phi_G^{Ag}$.*
2. *$(\Phi_G^{Ag} - \{G_{Ag}h_i\} \cup \{G_{Ag}h_j\}, G_{Ag}h, C)$ is also an acceptable realization argument of $G_{Ag}h$.*

Proposition 5. *Let $G_{Ag}h_j$ be an alternative goal of a strategic goal $G_{Ag}h_i$ relative to a given goal $G_{Ag}h$. $G_{Ag}h_j$ is also a strategic goal of $G_{Ag}h$.*

Proof. According to the second condition of Definition 8, there is realization argument $(\Phi_G^{Ag}, G_{Ag}h, C)$ of $G_{Ag}h$ such that $G_{Ag}h_j \in \Phi_G^{Ag}$. Consequently and according to Definition 7, $G_{Ag}h_j$ is a strategic goal of $G_{Ag}h$. ■

Proposition 6. *Deciding if there are strategic goals to achieve a given goal is in Σ_3^P.*

Proof. This result is a straightforward consequence of Theorem 1. ■

Theorem 2. *Deciding if there is an alternative goal of a given strategic goal is in $P^{\|\Sigma_P^2}$.*

Proof. Let $\Psi_G^{Ag} = \Gamma - \Phi_G^{Ag}$ be the set of Ag's goals not used in the realization argument of the given strategic goal $G_{Ag}h$. The following algorithm decides the problem: For each goal $G_{Ag}h_j \in \Psi_G^{Ag}$, decides if $(\Phi_G^{Ag} - \{G_{Ag}h_i\} \cup \{G_{Ag}h_j\}, G_{Ag}h, C)$ is a realization argument. By Proposition 4, this can be done with a polynomial number of calls to an Σ_2^P-oracle. Because these verifications could be done in parallel, we are done. ■

3.2 Generation and Satisfaction of Goals and Constraints

The argumentation-based strategic reasoning presented above enables agents to generate the strategic goals that ensure the realization of the conversation goal while respecting, in each dialogue step, the set of constraints related to this goal. The idea is that when the dialogue progresses, before adapting a new set of strategic goals, agents should find an acceptable explanatory argument to justify the selection of this set. Thereafter, they should find an acceptable realization argument so that the new constraints associated to the identified strategic goals along with the previous constraints are satisfied. As defined in Definition 3, realization arguments need explanatory arguments. These acceptable arguments should be built from the agent's beliefs and the public information conveyed by the addressee during the previous dialogue steps. Before discussing some formal properties and the argumentative reasoning algorithm for generating and satisfying goals and constraints, let us define the following notions:

Definition 9 (*Elementary Goal*). *A goal ϕ is elementary iff it cannot be decomposed into strategic goals.*

Definition 10 (*Super-Goal*). *Let $(\Phi_G^{Ag}, G_{Ag}h, C)$ be an acceptable realization argument. $G_{Ag}h$ is the super-goal of a strategic goal ϕ iff $\phi \in \Phi_G^{Ag}$.*

We have the following proposition:

Proposition 7. *Let ϕ, C_ϕ, and C be a new strategic goal, the associated constraints, and the previous constraints respectively. The consistency of C_ϕ and C is a sufficient condition, but not a necessary one for the realization of ϕ.*

Proof. Let ϕ' be a goal. According to Definition 7, ϕ is a strategic goal of ϕ' iff there is a realization argument (Φ_G^{Ag}, ϕ', C) such that $\phi \in \Phi_G^{Ag}$. Consequently, and according to Definition 3, ϕ and C are consistent. If ϕ is elementary, it could be realized. Otherwise, strategic goals of ϕ should be found. This means that a new realization argument$(\Phi_G'^{Ag}, \phi, C \cup C_\phi)$ should be built. According to Definition 3, the consistency of $\Phi_G'^{Ag}$ and $C \cup C_\phi$ is only one condition, the second condition is the logical entailment of ϕ. Therefore, the consistency of C and C_ϕ is necessary but not sufficient for the realization of ϕ. ■

As a direct result of this proposition, if the new constraints are not consistent with the previous ones, the identified strategic goal should be dropped, and an alternative should be found. This sheds the light on two possible scenarios when identifying strategic goals depending on the satisfaction of associated constraints. The first scenario is *following up the strategy*, and the second one is *changing the strategy*. The first scenario takes place when all the identified strategic goals can be satisfied, which means that an acceptable realization argument $(\Phi_G^{Ag}, \phi, C \cup C_\phi)$ can be built for each strategic goal ϕ considering previous and new constraints $C \cup C_\phi$. The second scenario takes place when at least one of the strategic goals cannot be realized because the new constraints are not consistent with the previous constraints or because the new constraints cannot be satisfied

even if they are consistent with the previous constraints. In this case an alternative should be identified. Our solution is to reconsider the super-goal of the non-realized strategic goal and try to find new strategic goals of this super-goal using the same argumentative reasoning. This means that building a new realization argument of the super-goal considering the new constraints. This process is sketched in Algorithm 1. In this algorithm, the function $SuperGoal(\phi)$ returns the super-goal of a given strategic goal ϕ.

Algorithm 1. Strategic goals identification

Input: The conversation goal $G_{Ag}h$ and the associated constraints C
Initialization: Q: an empty FIFO queue
1: Given $G_{Ag}h$ and C, compute Φ_G^{Ag} by building an acceptable realization argument $(\Phi_G^{Ag}, G_{Ag}h, C)$
2: $\forall\phi \in \Phi_G^{Ag}$, put ϕ in Q
3: while Q is not empty do
4: $\phi \leftarrow First(Q)$ % the first element of Q
5: Remove ϕ from Q
6: if ϕ is not elementary, then
7: $C \leftarrow C \cup C_\phi$ % C_ϕ are the constraints associated to ϕ
8: Given ϕ and C, if $\exists(\Phi_G^{Ag}, \phi, C)$ an acceptable realization argument
9: then $\forall\phi \in \Phi_G^{Ag}$, put ϕ in Q % *Follow up the strategy*
10: else % *Change the strategy*
11: while ϕ is different from the conversation goal do
12: $\phi \leftarrow SuperGoal(\phi)$
13: Remove from Q the sub-goals of ϕ previously computed
14: Given ϕ and C, if $\exists(\Phi_G^{Ag}, \phi, C)$ an acceptable realization argument
15: then $\forall\phi \in \Phi_G^{Ag}$, put ϕ in Q
16: exit() % exit second while
17: end while
18: if ϕ is equal to the conversation goal then return false
19: end while
20: return true

We have the following two theorems about the termination, soundness and completeness of Algorithm 1:

Theorem 3. *Algorithm 1 terminates either by identifying the elementary strategic goals for the realization of the conversation goal or by identifying the case where this goal cannot be realized.*

Proof. Algorithm 1 includes two embedded whiles. The second while always terminates since its condition becomes true when *SuperGoal* function returns the conversation goal. This is always possible because the number of strategic goals is finite. In this case there is at least a strategic goal of the conversation goal that can not be realized. The algorithm terminates by returning false. If the termination of this second while is forced by exit(), the first while always terminates since

the number of strategic goals is finite and each time a strategic goal is dealt with, this goal is get removed from the queue. In this case, the algorithm successfully terminates by identifying the elementary strategic goals. ■

Theorem 4. *The conversation goal $G_{Ag}h$ can be realized iff Algorithm 1 terminates by identifying a set of elementary strategic goals for the realization of $G_{Ag}h$.*

Proof. On one hand, let us suppose that Algorithm 1 terminates by identifying a set of elementary strategic goals. Therefore, the algorithm terminates when the queue is empty. This means that all the strategic goals are supported by acceptable realization arguments. Because the constraints related to the new strategic goals are considered, building these acceptable realization arguments means that the whole constraints are satisfied. Therefore, the conversation goal can be realized by realizing the identified elementary goals.

On the other hand, let us suppose that the conversation goal $G_{Ag}h$ which is the input of Algorithm 1 can be realized. Therefore, it can be composed into a set of elementary strategic goals that can be realized. However, this set is not necessarily unique. Algorithm 1 will identify a possible set since many alternatives are tried until this set is identified. This is possible thanks to the backtracking processed in the second while using the *SuperGoal* function. Algorithm 1 will then terminate when elementary strategic goals are identified. ■

As indicated in Section 2, the set of the agent's goals are represented by a tree in which the root represents the conversation goal, the nodes represent decomposable strategic goals, and the leaves represent *elementary* strategic goals (i.e. strategic goals that cannot be decomposed). This tree is built progressively while the dialogue progresses. To simplify the notation, an agent's conversation goal will be denoted by B and its strategic goals will be denoted by B_{ij}. The principal idea of the goals decomposition is that for each conversation or strategic goal we have:

1. If the goal is elementary, then the agent tries to satisfy it by using a tactical reasoning. This reasoning enables agents to choose the most relevant communicative act in order to achieve this goal. More details about this type of reasoning can be found in [8].
2. If the goal is decomposable, then the agent must calculate, using the argumentative process described above, the sub-goals to achieve in order to realize the initial goal. For each sub-goal, the agent can have several alternatives. The achievement of these alternatives can provide the same result as that provided by the initial sub-goals. An alternative is identified when the associated realization argument is built. However, if many alternatives are possible, deciding which one is the best is out of the paper scope. Here, an alternative is adopted once it is identified. The set of the strategic goals which can be used to achieve the same goal are connected by a concave arc, as indicated in Fig.2. In this figure, the goal B may be decomposed into two goals B_{11} and B_{21}. For the goal B_{11}, we can have several alternatives $(B_{12}, \ldots, B_{1n})$ and realizing B requires the achievement of B_{11} or one

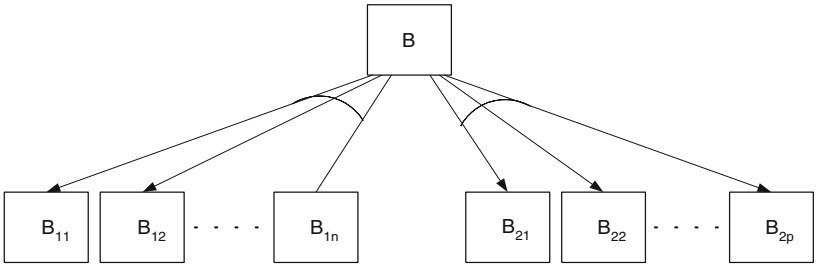

Fig. 2. Example of decomposition of a conversation or strategic goal

of its alternatives (there is thus a disjunction). In the same way, the realization of B requires achieving B_{21} or one of its alternatives: $B_{22}, \ldots, B_{2p}$. The realization of B requires then the achievement of a goal from the set $\{B_{11}, B_{12}, \ldots, B_{1n}\}$ and a goal from the set $\{B_{21}, B_{22}, \ldots, B_{2p}\}$ (there is thus a conjunction).

In a general way, the definition of the set of constraints related to a set of strategic goals is defined as follows:

Definition 11 (*Constraint Definition*). *Let B be a conversation or strategic goal and $\mathcal{E}$ be a function associating elementary goals to a set of constraints. The constraints associated with the set of the strategic goals ($StrG(B)$) of the goal B is given by the function $\mathcal{C}$ which is defined in Table 1.*

In Definition 11, the function $\mathcal{C}$ takes as argument a set of goals (or a set of graphs) and gives us a set of subsets of constraints representing the set of possible scenarios (i.e. each subset represents a scenario). In the practice, this function is implemented using the argumentation machinery explained above. α and β are two sets of constraints. $B(1)$ represents the set composed by the strategic goal B_1 of the goal B and its possible alternatives. $\overline{B(1)}$ represents the remaining goals in the tree representing the goal B. In a general way, $B(i)$ represents the set including the i^{th} strategic goal of the goal B and its possible alternatives and $\overline{B(i)}$ represents the remaining goals of the tree representing the goal B. For example, in Fig. 2 we have: $B(1) = \{B_{11}, B_{12}, \ldots, B_{1n}\}$ and $\overline{B(1)} = \{B_{21}, B_{22}, \ldots, B_{2p}\}$. Furthermore, we consider that the constraints related to an elementary goal are generated from the speech act performed to achieve this goal. For example, for

Table 1. Constraint Generation Function

$\mathcal{C}(\emptyset)$	$= \{\emptyset\}$
$\mathcal{C}(\{B\})$	$= \mathcal{E}(B)$ if B is elementary
$\mathcal{C}(\{B_i\})$	$= \bigcup_{\alpha \in \mathcal{C}(\{B(i)\})} \bigcup_{\beta \in \mathcal{C}(\{\overline{B(i)}\})} \{\alpha \cup \beta\}$ if B is not elementary
$\mathcal{C}(\{B_1, B_2, \ldots, B_n\})$	$= \mathcal{C}(\{B_1\}) \cup \mathcal{C}(\{B_2\}) \cup \ldots \cup \mathcal{C}(\{B_n\})$

the speech act: "*I sell you my watch for 5 dollars*", the function $\mathcal{E}$ gives us the set which contains the constraint: "*the price is equal to 5 dollars*". We have the following proposition and its proof is straightforward.

Proposition 8 (*Constraints Satisfaction*). *Let B be a conversation goal and $Ctr(B)$ be the set of its initial constraints. The set of constraints $Ctr(B)$ is satisfied if there is a set $C_i \in \mathcal{C}(\{B\})$ such that: $Ctr(B) \cup C_i$ is consistent.*

As a result of this proposition, a conversation goal B cannot be achieved if there is no set of constraints in the set $\mathcal{C}(\{B\})$ which is consistent with the set $Ctr(B)$ (set of constraints associated with the agent's conversation goal). In other words, an agent that is able to satisfy the constraints which appear during the dialogue would be able to achieve its conversation goal. The reason is that this agent will be able to build an acceptable realization argument supporting the strategic goals of the conversation goal by considering the new constraints.

4 Discussion and Related Work

The framework proposed in this paper has many advantages. First, it provides a technique for reasoning about communication strategies. Also, it enables agents to reason about strategic goals and the dynamics of the associated constraints. In fact, unlike existing frameworks [1,10,27,28,37]), our proposal allows agents to adjust their strategies by considering the new constraints and identifying possible alternatives using argumentation reasoning. In our negotiation example (Examples 1 and 2), the existing frameworks do not guarantee the realization of the conversation goal B (selling the car), if the goal B_{32} (convincing the buyer by the fact that the car is economic), cannot be achieved. In our approach, the goal B can still be achieved by reconsidering the decomposition of the goal B when new constraints appear. For instance, let us assume that during the dialogue the seller learns that the buyer does not need an economic car (for example because he is currently working close to home). In this case, the seller is unable to convince the buyer by achieving the goal B_{32}, that the proposed car is economic. Thereafter, the seller will give up the goal B_{32} and will try to persist in the realization of his conversation goal which is selling the proposed car by using the alternative goal B'_{32}. This goal aims at trying to convince the buyer that the spare parts for the proposed car are available and not expensive.

Formalizing goals has attracted a special attention in the last decade within multi-agent systems. Winikoff et al. [37] discuss the two aspects of goals: declarative (a description of the state sought), and procedural (a set of plans for achieving the goal). A framework integrating these two aspects is proposed. Nigam and Leite [22] use dynamic logic programming to represent the agent's goals and their evolution. van Riemsdijk et al. [35,36] investigate the dynamics of declarative goals in agent programming. In these proposals internal and external motivations such as norms, obligations, and impositions about adopting and dropping goals are considered. Our work is inline with these proposals by considering declarative goals, but the context, motivations and underlying techniques

are different. Also, some interesting frameworks in the context of agent communication have been proposed by Amgoud and Kaci [1], Shapiro and Brewka [27], Shapiro et al. [28], Sardina et al. [26], and Caelen [10].

The approach proposed by Amgoud and Kaci [1] considers the realization of a goal as the realization of a plan which is composed of a set of sub-goals, called conditions. In this approach, an initial goal is subject to some conditions (in terms of agent's beliefs) so that it is adopted or followed by the agent. Consequently, this approach is interested in the generation of the initial goals (followed by the agent) and of the conditional goals which compose the plans of the initial goals. In fact, this approach considers the conditional goals as constraints to be satisfied in order to achieve an initial goal. However, it is different from ours in which we distinguish between the agents' constraints and the goals. The agent's goals reflect its objectives, whereas the constraints reflect the limits encountered to realize these objectives. Another fundamental difference is that our approach considers strategic reasoning and strategic goals which are different from conversation goals. Indeed, a strategic goal may be achieved, substituted for one of its alternatives, or rejected if the strategy is being changed, (i.e. if one of its constraints is not satisfied). In contrast, a conversation goal can only be realized or failed. To achieve a conversation goal, the agent must have a strategic reasoning which enables it to follow strategic sub-goals. The realization of a conversation or strategic goal is conditioned mainly by the capability of the agent to convince its addressee by using an argumentative process. More precisely, in our approach we are interested in realizing a conversation goal in terms of the performance of its strategic sub-goals by respecting the constraints (imposed beforehand on the agent or generated by the agent for each goal according to its beliefs).

In the approach proposed by Shapiro and his colleagues [28], an agent adopts a goal with reference to a request on behalf of another agent. An agent maintains its goal as long as it did not receive a request for cancellation of this goal. Furthermore, in [27], if an agent believes a goal is impossible to achieve, it is dropped. However, if the agent later believes that it was mistaken about the impossibility of achieving the goal, the agent might readopt the goal. Contrary to this approach, we do not consider in this paper the problem of goal change, but rather we deal with the problem of achieving conversation goals in terms of sub-goals and constraints that should be satisfied. In other words, our approach does not address goal change, but it tackles the realization of conversation goals in terms of strategic sequences of actions and constraints. Readopting goals is a form of persistence which is completely different from ours in which we consider other strategies to achieve the same goal.

Goals and dialogue strategy concepts were also dealt with by Caelen [10]. He presents an approach for the logical modeling of dialogues in which each interlocutor looks for achieving his goal by using the best possible strategies. The author models a kind of interaction between the goals of each agent and the dialogue strategy without considering agent reasoning. However, in our approach we model strategic reasoning for each agent implied in the conversation using

argumentation. This reasoning, which is non-monotonic, takes into account the nature of dialogue which is a dynamic and a joint activity.

Finally, Sardina and his colleagues [26] propose an approach for the formalization of goals based on the planning which is static in nature. However, dialogue is purely a dynamic activity. Therefore, dialogues cannot be modelled by plans to follow. Our proposal is different because it is based upon argumentation-driven reasoning about strategic goals and constraints. Sub-goals are generated and substituted for alternatives dynamically while the dialogue progresses. The strategy can also be adjusted when other information becomes available.

5 Conclusion

In this paper, we proposed an argumentation-driven approach for interacting agents in a multi-agent setting allowing them to reason about goals. This formalism is based on a strategic reasoning which provides a mechanism to calculate a cognitive representation of the manner of achieving a conversation goal in terms of strategic goals. Agents' strategic goals are supported by two types of arguments: explanatory arguments to justify the choice of the goals and realization arguments which provide the set of sub-goals necessary to achieve these goals given a set of constraints. The formalism also allows agents to persist in achieving their conversation goals by using alternative goals.

As future work, we plan to address the relevance issue within strategic reasoning. Since an agent can have several dialogue strategies for the same conversation goal, we are interested in proposing a method enabling agents to select the most relevant and efficient strategy.

References

1. Amgoud, L., Kaci, S.: On the Generation of Bipolar Goals in Argumentation-based Negotiation. In: Rahwan, I., Moraïtis, P., Reed, C. (eds.) ArgMAS 2004. LNCS, vol. 3366, pp. 192–207. Springer, Heidelberg (2005)
2. Atkinson, K., Bench-Capon, T., McBurney, P.: Computational Representation of Practical Argument. Synthese 152(2), 157–206 (2006)
3. Baldoni, M., Baroglio, C., Martelli, A., Patti, V.: Verification of Protocol Conformance and Agent Interoperability. In: Toni, F., Torroni, P. (eds.) CLIMA 2005. LNCS, vol. 3900, pp. 265–283. Springer, Heidelberg (2006)
4. Baldoni, M., Baroglio, C., Martelli, A., Patti, V., Schifanella, C.: Verifying Protocol Conformance for Logic-Based Communicating Agents. In: Leite, J., Torroni, P. (eds.) CLIMA 2004. LNCS, vol. 3487, pp. 196–212. Springer, Heidelberg (2005)
5. Bange, P.: Analyse Conversationnelle et théorie de l'Action, Paris, Hatier (1992)
6. Bench-Capon, T.J.M.: Persuasion in practical argument using value based argument. Journal of Logic and Computation 13(3), 429–448 (2003)
7. Bentahar, J., Moulin, B., Meyer, J.-J.C., Lespérance, Y.: A New Logical Semantics for Agent Communication. In: Inoue, K., Satoh, K., Toni, F. (eds.) CLIMA 2006. LNCS (LNAI), vol. 4371, pp. 151–170. Springer, Heidelberg (2007)

8. Bentahar, J., Mbarki, M., Moulin, B.: Specification and Complexity of Strategicbsed Reasoning Using Argumentation. In: Maudet, N., Parsons, S., Rahwan, I. (eds.) ArgMAS 2006. LNCS (LNAI), vol. 4766, pp. 142–160. Springer, Heidelberg (2007)
9. Brewka, G.: Dynamic argument systems: A formal model of argumentation processes based on stuation calculus. Journal of Logic and Computation 11(2), 257–282 (2001)
10. Caelen, J.: Stratégies de dialogue. Modèles Formels d'Intéraction (MFI 2003), Lille, CEPADUES Editions (May 2003)
11. Castelfranchi, C.: Commitments: from individual intentions to groups and organizations. In: International Conference on Multi Agent Systems, pp. 41–48 (1995)
12. Chesnevar, C.I., Maguitman, A., Loui, R.: Logical models of argument. ACM Computing Surveys 32, 337–383 (2000)
13. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence 77, 321–357 (1995)
14. Endriss, U., Maudet, N., Sadri, F., Toni, F.: Protocol conformance for logic-based agents. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, pp. 679–684 (2003)
15. Fisher, M., Bordini, R.H., Hirsch, B., Torroni, P.: Computational logics and agents: a roadmap of current technologies and future trends. Computational Intelligence Journal (2006)
16. Johnson, D.S.: A Catalog of Complexity Classes. In: Handbook of Theoretical Computer Science, ch. 2, Elsevier Science Publishers, Amsterdam (1990)
17. Hayashi, H., Tokura, S., Hasegawa, T., Ozaki, F.: Dynagent: An Incremental Forward-Chaining HTN Planning Agent in Dynamic Domains. In: Baldoni, M., Endriss, U., Omicini, A., Torroni, P. (eds.) DALT 2005. LNCS (LNAI), vol. 3904, pp. 171–187. Springer, Heidelberg (2006)
18. Kakas, A.C., Maudet, N., Moraïtis, P.: Layered Strategies and Protocols for Argumentation-Based Agent Interaction. In: Rahwan, I., Moraïtis, P., Reed, C. (eds.) ArgMAS 2004. LNCS, vol. 3366, pp. 64–77. Springer, Heidelberg (2005)
19. Moreira, Á.F., Vieira, R., Bordini, R.H.: Extending the Operational Semantics of a BDI Agent-Oriented Programming Language for Introducing Speech-Act Based Communication. In: Leite, J., Omicini, A., Sterling, L., Torroni, P. (eds.) DALT 2003. LNCS (LNAI), vol. 2990, pp. 135–154. Springer, Heidelberg (2004)
20. Moulin, B., Irandoust, I., Bélanger, M., Desbordes, G.: Explanation and argumentation capabilities: towards the creation of more persuasive agents. Artificial Intelligence Review 17(3), 169–222 (2002)
21. Nau, D.S., Cao, Y., Lotem, A., Munoz-Avila, H.: SHOP: Simple Hierarchical Ordered Planner. In: IJCAI 1999, pp. 968–975 (1999)
22. Nigam, V., Leite, J.: A Dynamic Logic Programming Based System for Agents with Declarative Goals. In: Baldoni, M., Endriss, U. (eds.) DALT 2006. LNCS (LNAI), vol. 4327, pp. 174–190. Springer, Heidelberg (2006)
23. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, Reading (1994)
24. Pezzulo, G., Falcone, R., Hoffmann, J.: Anticipation and Anticipatory Behavior: Editorial. Cognitive Processing 8(2), 67–70 (2007)
25. Prakken, H., Vreeswijk, G.: Logics for defeasible argumentation. In: Handbook of Philosophical Logic, 2nd edn. (2000)

26. Sardina, S., De Giacomo, G., Lesperance, Y., Levesque, H.: On the Limits of Planning over Belief States under Strict Uncertainty. In: Proceedings of the KR 2006 Conference, June 2006, pp. 463–471 (2006)
27. Shapiro, S., Brewka, G.: Dynamic interactions between goals and beliefs. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2007), Hyderabad, India, pp. 2625–2630 (2007)
28. Shapiro, S., Lesperance, Y., Levesque, H.: Goal change. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2005), Denver, CO, pp. 582–588 (2005)
29. Singh, M.P.: A social semantics for agent communication language. In: Dignum, F., Greaves, M. (eds.) Issues in Agent Communication, pp. 31–45 (2000)
30. Smith, I., Cohen, P., Bradshaw, J., Greaves, M., Holmback, H.: Desiging Conversation Policies using Joint Intention Theory. In: Proc. ICMAS 1998, pp. 269–276. IEEE Press, Los Alamitos (1998)
31. Toni, F., Torroni, P.: Computational Logic in Multi-Agent Systems. In: Toni, F., Torroni, P. (eds.) CLIMA 2005. LNCS (LNAI), vol. 3900. Springer, Heidelberg (2006)
32. Torroni, P., Gavanelli, M., Chesani, F.: Argumentation in the Semantic Web. IEEE Intelligent Systems 22(6), 66–74 (2007)
33. van der Hoek, W., Jamroga, W., Wooldridge, M.: A logic for strategic reasoning. In: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2005), pp. 157–164. ACM Inc., New York (2005)
34. van Dijk, T.A., Kintsch, W.: Strategies of Discourse Comprehension. Academic Press, New York (1983)
35. van Riemsdijk, M.B., Dastani, M., Dignum, F.P.M., 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)
36. van Riemsdijk, B., Dastani, M., Meyer, J.-J.: Semantics of declarative goals in agent programming. In: AAMAS 2005, pp. 133–140 (2005)
37. Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative and Procedural Goals in Intelligent Agent Systems. In: KR 2002. Morgan Kaufmann, San Francisco (2002)
38. Yolum, P., Singh, M.P.: Reasoning about Commitments in the Event Calculus: An Approach for Specifying and Executing Protocols. Annals of Mathematics and Artificial Intelligence 42(1-3), 227–253 (2004)

Author Index