LNAI 6245

Jürgen Dix
João Leite
Guido Governatori
Wojtek Jamroga (Eds.)

Computational Logic in Multi-Agent Systems

11th International Workshop, CLIMA XI
Lisbon, Portugal, August 2010
Proceedings

Springer

Jürgen Dix João Leite
Guido Governatori Wojtek Jamroga (Eds.)

Computational Logic in Multi-Agent Systems

11th International Workshop, CLIMA XI
Lisbon, Portugal, August 16-17, 2010
Proceedings

Springer

Volume Editors

Jürgen Dix
Clausthal University of Technology
Julius-Albert-Str. 4, 38678 Clausthal-Zellerfeld, Germany
E-mail: dix@tu-clausthal.de

João Leite
CENTRIA, Universidade Nova de Lisboa
2829-516 Caparica, Portugal
E-mail: jleite@di.fct.unl.pt

Guido Governatori
NICTA
P.O. Box 6030, St Lucia, 4067, QLD, Australia
E-mail: guido.governatori@nicta.com.au

Wojtek Jamroga
University of Luxembourg
Campus Kirchberg
6, rue Richard Coudenhove-Kalergi, 1359 Luxembourg, Luxembourg
E-mail: wojtek.jamroga@uni.lu

Library of Congress Control Number: 2010931764

CR Subject Classification (1998): I.2, F.3, D.2, C.2.4, F.4.1, D.3

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-642-14976-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-14976-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

These are the proceedings of the 11th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA-XI), held during August 16–17, in Lisbon, collocated with the 19th European Conference on Artificial Intelligence (ECAI-2010).

Multi-agent systems are communities of problem-solving entities that can perceive and act upon their environment in order to achieve both their individual goals and their joint goals. The work on such systems integrates many technologies and concepts from artificial intelligence and other areas of computing as well as other disciplines. Over recent years, the agent paradigm gained popularity, due to its applicability to a full spectrum of domains, such as search engines, recommendation systems, educational support, e-procurement, simulation and routing, electronic commerce and trade, etc. Computational logic provides a well-defined, general, and rigorous framework for studying the syntax, semantics and procedures for the various tasks in individual agents, as well as the interaction between, and integration among, agents in multi-agent systems. It also provides tools, techniques and standards for implementations and environments, for linking specifications to implementations, and for the verification of properties of individual agents, multi-agent systems and their implementations.

The purpose of the CLIMA workshops is to provide a forum for discussing techniques, based on computational logic, for representing, programming and reasoning about agents and multi-agent systems in a formal way.

Former CLIMA editions have been conducted in conjunction with other major Computational Logic and AI events such as CL in 2000, ICLP in 2001 and 2007, FLoC in 2002, LPNMR and AI-Math in 2004, JELIA in 2004 and 2008 and MATES in 2009. In 2005 CLIMA was not associated with any major event.

The 11th edition of the CLIMA workshop featured some new elements. For the first time, these are not post-workshop proceedings, but regular proceedings published in time for the workshop. In addition, CLIMA featured two thematic Special Sessions.

Norms and Normative Multi-Agent Systems: Norms are pervasive in everyday life and influence the conduct of the entities subject to them. One of the main functions of norms is to regulate the behavior and relationships of agents. Accordingly, any agent or multi-agent system, if intended to operate in or model a realistic environment has to take into account norm regulating. Norms have been proposed in multi-agent systems and computer science to deal with coordination issues, to deal with security issues of multi-agent systems, to model legal issues in electronic institutions and electronic commerce, to model multi-agent organizations, etc.

Logics for Games and Strategic Reasoning: Strategic reasoning occurs in many multi-agent systems. This is especially evident in game-theoretical and decision-theoretical models of MAS, but also in more informal settings using the game metaphor (e.g., computer games or social network services). Mathematical logic can contribute to this view in many ways by providing specifications, models, and/or algorithms for game-like scenarios. We invited papers that address how logic can contribute to our understanding, modeling and analysis of games, but also ones that investigate how the metaphor of games and strategies can help in constructing and using logical formalisms.

We believe that emphasizing particular areas each year helps to attract attention to these important research topics.

In line with the high standards of previous CLIMA editions, the review process was very selective, the final acceptance rate being below 50%. From 31 submissions, a Program Committee of 54 top-level researchers from 18 countries and 11 additional reviewers selected 14 papers for presentation, authored by 43 researchers worldwide.

These proceedings feature the 14 regular papers, as well as abstracts of four invited talks, given by Toby Walsh (NICTA, Australia), Ron van der Meyden (University of New South Wales, Australia), Thomas Ågotnes (University of Bergen, Norway), and Stefan Woltran (TU Vienna, Austria).

Toby Walsh considered in his talk "Is Computational Complexity a Barrier to Manipulation?" voting rules and whether they can be manipulated. While the well-known Gibbard-Satterthwaite theorem (related to Arrow's paradox) states that most systems can be manipulated, a way out might be that manipulation is *computationally too expensive* to carry out. He suggests studying this problem empirically and investigates various interesting phase transitions (which prove to be important in other areas).

Ron van der Meyden considers "Games and Strategic Notions in Information Flow Security." A computer system or a group of agents often do not want to disclose confidential information. But adversaries might be able to deduce such data. The talk was about giving precise definitions of information flow security and their complexity from the viewpoint of epistemic logic.

Thomas Ågotnes talked about "Group Announcements: Logic and Games." He reviewed the classical results in dynamic epistemic logics where the problem of changing knowledge of individual agents is modeled. Public announcement logic emerged out of this research. Thomas extends this work by considering *group announcements*, where a subgroup of agents announce truthfully their knowledge. He discussed the logical and rational principles governing this kind of behavior.

Stefan Woltran discussed "Strong Equivalence in Argumentation." Equivalence in classical monotonic logic is well understood. But equivalence in non-monotonic theories (i.e., the same answer sets, default extensions, etc.) is quite different. The talk presented the main results obtained in the last few years and related them to argumentation frameworks.

The three contributions accepted for the *Norms and Normative Multi-Agent Systems* session address the problem of verifying the compliance of business processes against relevant regulations, the diagnosis of commitment violations, and the modeling agent and multi-agent systems using institutions and norms.

The paper "What Happened to My Commitment? Exception Diagnosis Among Misalignment and Misbehavior" by Özgür Kafali and Federico Chesani Torroni discusses the issue of detecting violation of commitments in an e-Contracts scenario. The authors argue that one of the possible reasons for misalignment of commitment in contracts depends on temporal aspects. Accordingly a contract and the commitments it contains are formalized in $\mathcal{REC}$, a form of reactive event calculus. They then propose an architecture and an algorithm for the diagnosis of the possible types of misalignments (i.e., violations) of commitments.

Davide D'Aprile, Laura Giordano, Valentina Gliozzi, Alberto Martelli, Gian Luca Pozzato and Daniele Theseider Dupré in their contribution on "Verifying Business Process Compliance by Reasoning About Actions" propose the use of temporal action theory as a common language to model business processes as well as the norms governing the processes. Temporal action theory combines answer set programming and dynamic linear time temporal logic. The combination allows for the representation in the same language of the business processes and the norms governing them (where obligations are represented as commitments). In this way verifying compliance amounts to checking that no execution of the process leaves some commitments unfulfilled. The verification can then be done using bounded model checking.

The aim of Jomi F. Hübner, Olivier Boissier and Rafael H. Bordini in "From Organisation Specification to Normative Programming in Multi-Agent Organisations" is to provide an automatic translation from a specification in a high-level organization modeling language ($\mathcal{M}$OISE) to a program in a simpler normative programming language (NOPL) that can be interpreted by an artifact-based organization management structure (OMI) with the aim of bringing more flexibility to developing an OMI. This is claimed to be the case since one can more easily change the translation than a Java implementation of the OMI.

The special session on *Logic for Games and Strategic Reasoning* consisted of four papers. In the paper titled "Exploring the Boundary of Half Positionality," Alessandro Bianco, Marco Faella, Fabio Mogavero and Aniello Murano consider languages of infinite words that can be interpreted as winning conditions in infinite games. Half positionality means that the proponent in the game can restrict her search for a winning strategy to positional strategies only. As the main result, the authors describe a novel sufficient condition for half positionality, more general than what was previously known. The paper has a strong link to work on application of games in logic – more precisely in verification techniques for strategic and temporal logics, where restricting the search to a limited set of simple possibilities is an important issue.

In "Playing Extensive Form Games in Parallel," Sujata Ghosh, R. Ramanujam and Sunil Easaw Simon consider a player playing simultaneously against different opponents in two extensive games. For the analysis, the authors propose a variant of dynamic logic for EF games. They present a complete axiomatization of the logic and show that it is decidable. Thus, the focus of the paper is application of logical formalisms to analysis of a specific kind of agent interaction, and characterizing general properties of such interactions.

"Finding Uniform Strategies for Multi-Agent Systems" by Jan Calta, Dmitry Shkatov and Holger Schlingloff presents another study of game-like interactions. An algorithm for finding uniform strategies in imperfect information games is proposed, which finds all maximal uniform strategies for enforcing a property expressible in alternating-time temporal logic (ATL). This can be seen as a semantic study of an important variant of strategic logics, but also as a step towards automated program synthesis for decentralized distributed systems of agents.

Finally, Jonathan Zvesper and Krzysztof Apt present their "Proof-Theoretic Analysis of Rationality for Strategic Games with Arbitrary Strategy Sets." Here the focus is again on characterization of general properties of games. More precisely, the authors provide an axiomatic proof of the statement "true common belief of rationality implies that the players will choose only strategies that survive the iterated elimination of strictly dominated strategies," where rationality means playing only strategies one believes to be best responses. An appropriate proof system is introduced, proved sound, and then shown capable of providing a formal derivation for the statement.

The remaining CLIMA papers cover a variety of topics, such as logics, knowledge representation, reasoning, agent programming languages, to name a few. A short summary of each paper follows.

The contribution by Sara Miner More and Pavel Naumov on "Hypergraphs of Multiparty Secrets" investigates the completeness of an axiomatization of a logic of secrets. The aim of the logic of secrets is to study the interdependencies between pieces of information, called secrets because they might be known by some parties and unknown by others.

The paper "Combining Logics in Simple Type Theory" by Christoph Benzmüller provides a good insight into the advances of the field on combining logics, in particular on the use of simple type theory (STT). Specifically, the author presents an embedding of (normal) quantified multimodal logic, and of other logics such as intuitionistic logics, access control logics, and logics for spatial reasoning in STT, and illustrates how STT can be used to reason about or within the combination of the logics embedded in SST. It also provides a set of experiments on the use of STT automated reasoners for the solution of several problems.

In "Speculative Abductive Reasoning for Hierarchical Agent Systems," Jiefei Ma, Krysia Broda, Randy Goebel, Hiroshi Hosobe, Alessandra Russo and Ken Satoh introduce the concept of speculative reasoning. This framework allows one to manage multiple revisable answers within the context of multi-agent systems, by introducing a novel abductive framework to hierarchical speculative reasoning.

"Formal Semantics of a Dynamic Epistemic Logic for Describing Knowledge Properties of Pi-Calculus Processes" by Pedro Arturo Gongora, Francisco Hernandez-Quiroz and Eric Ufferman presents an embedding of the process algebra of π-calculus in a multi-agent framework, in particular the notions of agency and knowledge. This is done through a variant of multi-agent dynamic epistemic logic, where epistemic actions of agents are π-calculus processes. In the semantics, these processes are translated to corresponding model updates. Thus, the proposal allows one to model agent interaction by a collection of π-calculus processes. Moreover, given a collection of such processes, we can derive dynamics of knowledge that it entails.

"Model Checking Agent Programs by Using the Program Interpreter" by Sung-Shik Jongmans, Koen Hindriks and M. Birna van Riemsdijk reports a new approach to explicit-state, on-the-fly model checking for agent programs. The idea is to reuse the program interpreter for generating the state space. The model checking algorithm is built on top of it by implementing efficient transformations of temporal properties to Buchi automata and an efficient bookkeeping mechanism that maintains track of the states that have already been visited. The approach is evaluated experimentally, with very promising results.

The paper "An Agent Language with Destructive Assignment and Model-Theoretic Semantics" by Robert Kowalski and Fariba Sadri presents an agent language that combines agent functionality with model-theoretic semantics. The underlying framework is that of abductive logic programming (ALP).

"A Dialogue Games Framework for the Operational Semantics of Logic Agent-Oriented Languages," by Stefania Costantini and Arianna Tocchio, introduces an operational semantics based on dialogue games that have their roots in the philosophy of argumentation. This allows for a uniform and modular way of modeling all the components of the interpreter.

We thank all the authors of all the papers submitted to CLIMA-XI for submitting papers and for revising their contributions to be included in these proceedings. We are very grateful to the members of the CLIMA-XI Program Committee and the additional reviewers. Their service ensured the high quality of the accepted papers.

A special thank you goes to Ulle Endriss, the ECAI-2010 Workshop Chair, and to the local organizers in Lisbon for their help and support. We are very grateful to them for handling the registration and for a very enjoyable social program.

June 2010

Jürgen Dix
João Leite
Guido Governatori
Wojtek Jamroga

Organisation

Workshop Chairs

Jürgen Dix	Clausthal University of Technology, Germany
João Leite	New University of Lisbon, Portugal

Special Session Chairs

Guido Governatori	NICTA, Australia
Wojtek Jamroga	University of Luxembourg, Luxembourg

Program Committee

Thomas Ågotnes	University of Bergen, Norway
Natasha Alechina	University of Nottingham, UK
José Júlio Alferes	New University of Lisbon, Portugal
Trevor Bench-Capon	The University of Liverpool, UK
Guido Boella	University of Turin, Italy
Giacomo Bonanno	University of California Davis, USA
Rafael Bordini	Fed. University of Rio Grande do Sul, Brazil
Gerhard Brewka	University of Leipzig, Germany
Jan Broersen	Utrecht University, The Netherlands
Nils Bulling	Clausthal University of Technology, Germany
Stefania Costantini	University of L'Aquila, Italy
Matteo Cristani	University of Verona, Italy
Mehdi Dastani	Utrecht University, The Netherlands
Louise Dennis	The University of Liverpool, UK
Hans van Ditmarsch	University of Seville, Spain
Michael Fisher	The University of Liverpool, UK
Chiara Ghidini	University of Trento, Italy
Valentin Goranko	Technical University of Denmark, Denmark
Davide Grossi	University of Amsterdam, The Netherlands
Hisashi Hayashi	Toshiba, Japan
Paul Harrenstein	University of Munich, Germany
Andreas Herzig	University of Toulouse, France
Koen Hindriks	Delft University, The Netherlands
Wiebe van der Hoek	The University of Liverpool, UK
Katsumi Inoue	National Insitute of Informatics, Japan
Alessio Lomuscio	Imperial College London, UK
Jenny Eriksson Lundström	Uppsala University, Sweden
Viviana Mascardi	University of Genova, Italy
Paola Mello	University of Bologna, Italy

John-Jules Meyer	Utrecht University, The Netherlands
Naoyuki Nide	Nara Women's University, Japan
Mehmet Orgun	Macquarie University, Australia
Eric Pacuit	Tilburg University, The Netherlands
Maurice Pagnucco	University of New South Wales, Australia
Wojciech Penczek	Polish Academy of Sciences, Poland
Duy Hoang Pham	PTIT, Vietnam
Gabriella Pigozzi	University of Luxembourg, Luxembourg
Enrico Pontelli	New Mexico State University, USA
Franco Raimondi	Middlesex University, UK
Antonino Rotolo	University of Bologna, Italy
Chiaki Sakama	Wakayama University, Japan
Pierre-Yves Schobbens	University of Namur, Belgium
Tran Cao Son	New Mexico State University, USA
Fariba Sadri	Imperial College London, UK
Ken Satoh	National Institute of Informatics, Japan
Kostas Stathis	Royal Holloway, University of Lodon, UK
Michael Thielscher	University of New South Wales, Australia
Leon van der Torre	University of Luxembourg, Luxembourg
Paolo Torroni	University of Bologna, Italy
M. Birna van Riemsdijk	Delft University, The Netherlands
Marina De Vos	University of Bath, UK
Cees Witteveen	Delft University, The Netherlands

Additional Reviewers

Armin Hezart	Macquare University, Australia
Brian Logan	University of Nottingham, UK
Emil Weydert	University of Luxembourg, Luxembourg
Federico Chesani	University of Bologna, Italy
Gauvain Bourgne	National Institute of Informatics, Japan
Ioana Boureanu	Imperial College London, UK
Magdalena Kacprzak	Technical University of Bialystok, Poland
Marco Montali	Univesrity of Bologna, Italy
Roie Zivan	Ben-Gurion University of the Negev, Israel
Timothy Norman	University of Aberdeen, UK

CLIMA Steering Committee

Jürgen Dix	Clausthal University of Technology, Germany
Michael Fisher	University of Liverpool, UK
João Leite	New University of Lisbon, Portugal
Francesca Toni	Imperial College London, UK
Fariba Sadri	Imperial College London, UK
Ken Satoh	National Institute of Informatics, Japan
Paolo Torroni	University of Bologna, Italy

Table of Contents

Invited Talks

Is Computational Complexity a Barrier to Manipulation? 1
Toby Walsh

Games, Norms and Strategic Notions in Information Flow Security (Abstract) 8
Ron van der Meyden

Group Announcements: Logic and Games (Extended Abstract) 9
Thomas Ågotnes

Strong Equivalence in Argumentation (Abstract) 14
Stefan Woltran

Regular Papers

Hypergraphs of Multiparty Secrets 15
Sara Miner More and Pavel Naumov

Combining Logics in Simple Type Theory 33
Christoph Benzmüller

Speculative Abductive Reasoning for Hierarchical Agent Systems 49
Jiefei Ma, Krysia Broda, Randy Goebel, Hiroshi Hosobe, Alessandra Russo, and Ken Satoh

Formal Semantics of a Dynamic Epistemic Logic for Describing Knowledge Properties of π-Calculus Processes 65
Pedro Arturo Góngora, Eric Ufferman, and Francisco Hernández-Quiroz

What Happened to My Commitment? Exception Diagnosis among Misalignment and Misbehavior 82
Özgür Kafalı, Federico Chesani, and Paolo Torroni

Verifying Business Process Compliance by Reasoning about Actions 99
Davide D'Aprile, Laura Giordano, Valentina Gliozzi, Alberto Martelli, Gian Luca Pozzato, and Daniele Theseider Dupré

From Organisation Specification to Normative Programming in Multi-Agent Organisations 117
Jomi F. Hübner, Olivier Boissier, and Rafael H. Bordini

Finding Uniform Strategies for Multi-agent Systems 135
Jan Calta, Dmitry Shkatov, and Holger Schlingloff

Playing Extensive Form Games in Parallel 153
Sujata Ghosh, R. Ramanujam, and Sunil Simon

Exploring the Boundary of Half Positionality 171
Alessandro Bianco, Marco Faella, Fabio Mogavero, and Aniello Murano

Proof-Theoretic Analysis of Rationality for Strategic Games with Arbitrary Strategy Sets 186
Jonathan A. Zvesper and Krzysztof R. Apt

An Agent Language with Destructive Assignment and Model-Theoretic Semantics 200
Robert Kowalski and Fariba Sadri

Model Checking Agent Programs by Using the Program Interpreter 219
Sung-Shik T.Q. Jongmans, Koen V. Hindriks, and M. Birna van Riemsdijk

A Dialogue Games Framework for the Operational Semantics of Logic Agent-Oriented Languages 238
Stefania Costantini and Arianna Tocchio

Author Index 257

Is Computational Complexity a Barrier to Manipulation?

Toby Walsh

NICTA and University of NSW, Sydney, Australia
toby.walsh@nicta.com.au

Abstract. When agents are acting together, they may need a simple mechanism to decide on joint actions. One possibility is to have the agents express their preferences in the form of a ballot and use a voting rule to decide the winning action(s). Unfortunately, agents may try to manipulate such an election by misreporting their preferences. Fortunately, it has been shown that it is NP-hard to compute how to manipulate a number of different voting rules. However, NP-hardness only bounds the worst-case complexity. Recent theoretical results suggest that manipulation may often be easy in practice. To address this issue, I suggest studying empirically if computational complexity is in practice a barrier to manipulation. The basic tool used in my investigations is the identification of computational "phase transitions". Such an approach has been fruitful in identifying hard instances of propositional satisfiability and other NP-hard problems. I show that phase transition behaviour gives insight into the hardness of manipulating voting rules, increasing concern that computational complexity is indeed any sort of barrier. Finally, I look at the problem of computing manipulation of other, related problems like stable marriage and tournament problems.

1 Introduction

The Gibbard Satterthwaite theorem proves that, under some simple assumptions, a voting rule can always be manipulated. In an influential paper [1], Bartholdi, Tovey and Trick proposed an appealing escape: perhaps it is computationally so difficult to find a successful manipulation that agents have little option but to report their true preferences? To illustrate this idea, they demonstrated that the second order Copeland rule is NP-hard to manipulate. Shortly after, Bartholdi and Orlin proved that the more well known Single Transferable Voting (STV) rule is NP-hard to manipulate [2]. Many other voting rules have subsequently been proven to be NP-hard to manipulate [3]. There is, however, increasing concern that worst-case results like these do not reflect the difficulty of manipulation in practice. Indeed, several theoretical results suggest that manipulation may often be easy (e.g. [4]).

2 Empirical Analysis

In addition to attacking this question theoretically, I have argued in a recent series of papers that we may benefit from studying it empirically [5,6]. There are several reasons why empirical analysis is useful. First, theoretical analysis is often restricted to particular distributions like uniform votes. Manipulation may be very different in practice due to correlations in the preferences of the agents. For instance, if all preferences are single-peaked then there voting rules where it is in the best interests of all agents to state their true preferences. Second, theoretical analysis is often asymptotic so does not reveal the size of hidden constants. The size of such constants may be important to the actual computational cost of computing a manipulation. In addition, elections are typically bounded in size. Is asymptotic behaviour relevant to the size of elections met in practice? An empirical study may quickly suggest if the result extends to more candidates. Finally, empirical studies can suggest theorems to prove. For instance, our experiments suggest a simple formula for the probability that a coalition is able to elect a desired candidate. It would be interesting to derive this exactly.

3 Voting Rules

My empirical studies have focused on two voting rules: single transferable voting (STV) and veto voting. STV is representative of voting rules that are NP-hard to manipulate without weights on votes. Indeed, as I argue shortly, it is one of the few such rules. Veto voting is, on the other hand, a simple representative of rules where manipulation is NP-hard when votes are weighted or (equivalently) we have uncertainty about how agents have voted. The two voting rules therefore cover the two different cases where computational complexity has been proposed as a barrier to manipulation.

STV proceeds in a number of rounds. Each agent totally ranks the candidates on a ballot. Until one candidate has a majority of first place votes, we eliminate the candidate with the least number of first place votes Ballots placing the eliminated candidate in first place are then re-assigned to the second place candidate. STV is used in a wide variety of elections including for the Australian House of Representatives, the Academy awards, and many organizations including the American Political Science Association, and the International Olympic Committee. STV has played a central role in the study of the computational complexity of manipulation. Bartholdi and Orlin argued that:

> *"STV is apparently unique among voting schemes in actual use today in that it is computationally resistant to manipulation."* (page 341 of [2]).

By comparison, the veto rule is a much simpler scoring rule in which each agent gets to cast a veto against one candidate. The candidate with the fewest vetoes wins. There are several reasons why the veto rule is interesting to study. The veto rule is very simple to reason about. This can be contrasted with other voting rules like STV. Part of the complexity of manipulating the STV rule appears to come from reasoning about what happens between the different rounds. The veto rule, on the other hand, has a single round. The veto rule is also on the borderline of tractability since constructive manipulation (that is, ensuring a particular candidate wins) of the veto rule by a coalition of weighted agents is NP-hard but destructive manipulation (that is, ensuring a particular candidate does not win) is polynomial [3].

4 Voting Distributions

Empirical analysis requires collections of votes on which to compute manipulations. My analysis starts with one of the simplest possible scenarios: elections in which each vote is equally likely. We have one agent trying to manipulate an election of m candidates in which n other agents vote. Votes are drawn uniformly at random from all $m!$ possible votes. This is the Impartial Culture (IC) model. In many real life situations, however, votes are correlated with each other. I therefore also considered single-peaked preferences, single-troughed preferences, and votes drawn from the Polya Eggenberger urn model [7]. In an urn model, we have an urn containing all possible votes. We draw votes out of the urn at random, and put them back into the urn with a additional votes of the same type (where a is a parameter). This generalizes both the Impartial Culture model ($a = 0$) and the Impartial Anonymous Culture ($a = 1$) model. Real world elections may differ from these ensembles. I therefore also sampled some real voting records [8,9]. Finally, one agent on their own is often unable to manipulate the result. I therefore also considered coalitions of agents who are trying to manipulate elections.

5 Results

My experiments suggest different behaviour occurs in the problem of computing manipulations of voting rules than in other NP-hard problems like propositional satisfiability and graph colouring [10,11]. For instance, we often did not see a rapid transition that sharpens around a fixed point as in satisfiability [12]. Many transitions appear smooth and do not sharpen towards a step function as problem size increases. Such smooth phase transitions have been previously seen in polynomial problems [13]. In addition, hard instances often did not occur around some critical parameter. Figures 1 to 3 reproduce some typical graphs from [6].

Similar phase transition studies have been used to identify hard instances of NP-hard problems like propositional satisfiability [12,17,18], constraint satisfaction

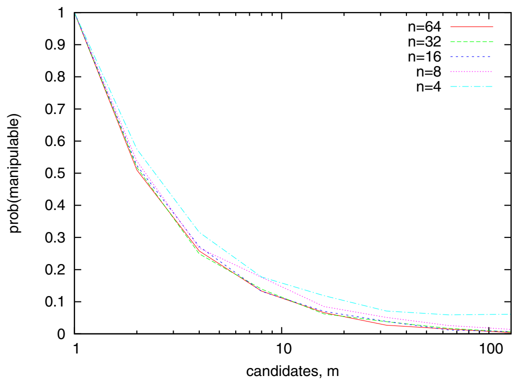


Fig. 1. Manipulability of correlated votes. The number of agents n is fixed and we vary the number of candidates m. The y-axis measures the probability that the manipulator can make a random candidate win.

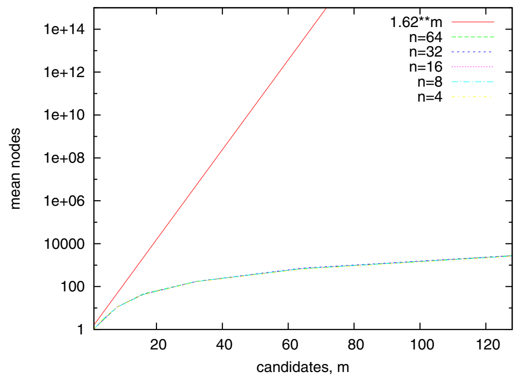


Fig. 2. Search to compute if an agent can manipulate an election with correlated votes. The number of agents n is fixed and we vary the number of candidates m. The y-axis measures the mean number of search nodes explored to compute a manipulation or prove that none exists. Median and other percentiles are similar. 1.62^m is the published worst-case bound for the recursive algorithm used to compute a manipulation [3].

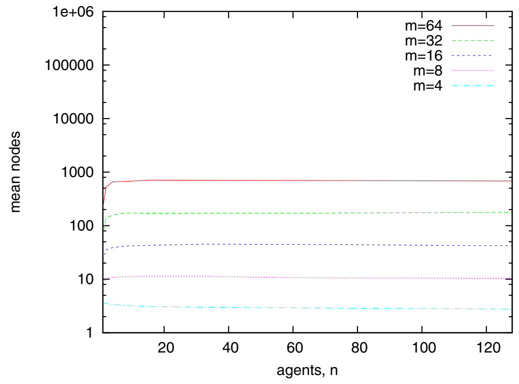


Fig. 3. Search to compute if an agent can manipulate an election with correlated votes. The number of candidates m is fixed and we vary the number of agents n.

[19,20,21,22,23], number partitioning [24,25,26], Hamiltonian circuit [27,28], and the traveling salesperson problem [29,30]. Phase transition studies have also been used to study polynomial problems [31,32] as well as higher complexity classes [33,34] and optimization problems [35,36]. Finally, phase transition studies have been used to study problem structure like small worldiness [37] and high degree nodes [38].

6 Other Manipulation Problems

Another multi-agent problem in which manipulation may be an issue is the stable marriage problem. This is the well-known problem of matching men to women so that no man and woman who are not married to each other both prefer each other. It has a wide variety of practical applications such as a matching doctors to hospitals. As with voting, an important issue is whether agents can manipulate the result by mis-reporting their preferences. Unfortunately, Roth [14] proved that *all* stable marriage procedures can be manipulated. We might hope that computational complexity might also be a barrier to manipulate stable marriage procedures. In joint work with Pini, Rossi and Venable, I have proposed a new stable marriage procedures that is NP-hard to manipulate [15]. Another advantage of this new procedure is that, unlike the Gale-Shapley algorithm, it does not favour one sex over the other. Our procedure picks the stable matching that is most preferred by the most popular men and women. The most preferred men and women are chosen using a voting rule. We prove that, if the voting rule used is STV then the resulting stable matching procedure is also NP-hard to manipulate. We conjecture that other voting rules which are NP-hard to manipulate will give rise to stable matching procedures which are also NP-hard to manipulate.

The final domain in which I have studied computational issues surrounding manipulation is that of (sporting) tournaments (joint work with Russell) [16]. Manipulating a tournament is slightly different to manipulating an election. In a sporting tournament, the voters are also the candidates. Since it is hard (without bribery or similar mechanisms) for a team to play better than it can, we consider just manipulations where the manipulators can throw games. We show that we can decide how to manipulate round robin and cup competitions, two of the most popular sporting competitions in polynomial time. In addition, we show that finding the minimal number of games that need to be thrown to manipulate the result can also be determined in polynomial time. Finally, we give a polynomial time proceure to calculate the probability that a team wins a cup competition under manipulation.

7 Conclusions

I have argued that empirical studies can provide insight into whether computational complexity is a barrier to the manipulation. Somewhat surprisingly, almost every one of the many millions of elections in the experiments in [5,6] was easy to manipulate or to prove could not be manipulated. Such experimental results increase the concerns that computational complexity is indeed a barrier to manipulation in practice. Many other voting rules have been proposed which could be studied in the future. Two interesting rules are maximin and ranked pairs. These two rules have only recently been shown to be NP-hard to manipulate, and are members of the small set of voting rules which are NP-hard to manipulate without weights or uncertainty [39]. These results demonstrate that empirical studies can provide insight into the computational complexity of computing manipulations. It would be interesting to consider similar phase transition studies for related problems like preference elicitation [40,41].

Acknowledgements. NICTA is funded by the Department of Broadband, Communications and the Digital Economy, and the Australian Research Council.

References

1. Bartholdi, J., Tovey, C., Trick, M.: The computational difficulty of manipulating an election. Social Choice and Welfare 6, 227–241 (1989)
2. Bartholdi, J., Orlin, J.: Single transferable vote resists strategic voting. Social Choice and Welfare 8, 341–354 (1991)
3. Conitzer, V., Sandholm, T., Lang, J.: When are elections with few candidates hard to manipulate. Journal of the Association for Computing Machinery 54 (2007)
4. Xia, L., Conitzer, V.: A sufficient condition for voting rules to be frequently manipulable. In: EC 2008: Proc. of the 9th ACM Conference on Electronic Commerce, pp. 99–108. ACM, New York (2008)
5. Walsh, T.: Where are the really hard manipulation problems? The phase transition in manipulating the veto rule. In: Proc. of 21st IJCAI, Int. Joint Conf. on Artificial Intelligence, pp. 324–329 (2009)
6. Walsh, T.: An empirical study of the manipulability of single transferable voting. In: Proc. of the 19th ECAI. IOS Press, Amsterdam (2010)
7. Berg, S.: Paradox of voting under an urn model: the effect of homogeneity. Public Choice 47, 377–387 (1985)
8. Gent, I., Walsh, T.: Phase transitions from real computational problems. In: Proc. of the 8th Int. Symposium on Artificial Intelligence, pp. 356–364 (1995)
9. Gent, I., Hoos, H., Prosser, P., Walsh, T.: Morphing: Combining structure and randomness. In: Proc. of the 16th National Conf. on AI. AAAI, Menlo Park (1999)
10. Cheeseman, P., Kanefsky, B., Taylor, W.: Where the really hard problems are. In: Proc. of the 12th IJCAI, Int. Joint Conf. on Artificial Intelligence, pp. 331–337 (1991)
11. Walsh, T.: The constrainedness knife-edge. In: Proc. of the 15th National Conf. on AI. AAAI, Menlo Park (1998)
12. Mitchell, D., Selman, B., Levesque, H.: Hard and Easy Distributions of SAT Problems. In: Proc. of the 10th National Conf. on AI, pp. 459–465. AAAI, Menlo Park (1992)
13. Walsh, T.: From P to NP: COL, XOR, NAE, 1-in-k, and Horn SAT. In: Proc. of the 17th National Conf. on AI. AAAI, Menlo Park (2002)
14. Roth, A.E.: The economics of matching: Stability and incentives. Mathematics of Operations Research 7, 617–628 (1982)
15. Pini, M., Rossi, F., Venable, K., Walsh, T.: Manipulation and gender neutrality in stable marriage procedures. In: 8th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), pp. 665–672 (2009)
16. Russell, T., Walsh, T.: Manipulating tournaments in cup and round robin competitions. In: Rossi, F., Tsoukiàs, A. (eds.) Algorithmic Decision Theory, First Int. Conf. (ADT 2009), pp. 26–37 (2009)
17. Gent, I., Walsh, T.: The SAT phase transition. In: Cohn, A.G. (ed.) Proc. of 11th ECAI, pp. 105–109. John Wiley & Sons, Chichester (1994)
18. Gent, I., Walsh, T.: The satisfiability constraint gap. Artificial Intelligence 81 (1996)
19. Prosser, P.: Binary constraint satisfaction problems: Some are harder than others. In: Proc. of the 11th ECAI, pp. 95–99 (1994)
20. Smith, B.: The phase transition in constraint satisfaction problems: A closer look at the mushy region. In: Proc. of the 11th ECAI (1994)
21. Gent, I., MacIntyre, E., Prosser, P., Walsh, T.: Scaling effects in the CSP phase transition. In: Montanari, U., Rossi, F. (eds.) CP 1995. LNCS, vol. 976, pp. 70–87. Springer, Heidelberg (1995)
22. Gent, I., MacIntyre, E., Prosser, P., Walsh, T.: The constrainedness of search. In: Proc. of the 13th National Conf. on AI, pp. 246–252. AAAI, Menlo Park (1996)

23. Gent, I., MacIntyre, E., Prosser, P., Smith, B., Walsh, T.: Random constraint satisfaction: Flaws and structure. Constraints 6, 345–372 (2001)
24. Mertens, S.: A physicist's approach to number partitioning. Theoretical Computer Science 265, 79–108 (2001)
25. Gent, I., Walsh, T.: Phase transitions and annealed theories: Number partitioning as a case study. In: Proc. of 12th ECAI (1996)
26. Gent, I., Walsh, T.: Analysis of heuristics for number partitioning. Computational Intelligence 14, 430–451 (1998)
27. Frank, J., Gent, I., Walsh, T.: Asymptotic and finite size parameters for phase transitions: Hamiltonian circuit as a case study. Information Processing Letters 66, 241–245 (1998)
28. Vandegriend, B., Culberson, J.: The Gn,m phase transition is not hard for the Hamiltonian Cycle problem. Journal of Artificial Intelligence Research 9, 219–245 (1998)
29. Gent, I., Walsh, T.: The TSP phase transition. Artificial Intelligence 88, 349–358 (1996)
30. Zhang, W.: Phase transitions and backbones of the asymmetric traveling salesman problem. Journal of Artificial Intelligence Research 21, 471–497 (2004)
31. Grant, S., Smith, B.: The phase transition behaviour of maintaining arc consistency. In: Wahlster, W. (ed.) Proc. of the 12th ECAI, pp. 175–179. Wiley, Chichester (1996)
32. Gent, I., MacIntyre, E., Prosser, P., Shaw, P., Walsh, T.: The constrainedness of arc consistency. In: Smolka, G. (ed.) CP 1997. LNCS, vol. 1330, pp. 327–340. Springer, Heidelberg (1997)
33. Gent, I., Walsh, T.: Beyond NP: the QSAT phase transition. In: Proc. of the 16th National Conf. on AI. AAAI, Menlo Park (1999)
34. Bailey, D., Dalmau, V., Kolaitis, P.: Phase transitions of PP-complete satisfiability problems. In: Proc. of the 17th IJCAI, Int. Joint Conf. on Artificial Intelligence, pp. 183–189 (2001)
35. Slaney, J., Walsh, T.: Phase transition behavior: from decision to optimization. In: Proc. of the 5th Int. Symposium on the Theory and Applications of Satisfiability Testing, SAT 2002 (2002)
36. Larrosa, J., Meseguer, P.: Phase transition in MAX-CSP. In: Wahlster, W. (ed.) Proc. of the 12th ECAI, pp. 190–194. Wiley, Chichester (1996)
37. Walsh, T.: Search in a small world. In: Proc. of 16th IJCAI, Int. Joint Conf. on Artificial Intelligence (1999)
38. Walsh, T.: Search on high degree graphs. In: Proc. of 17th IJCAI, Int. Joint Conf. on Artificial Intelligence (2001)
39. Xia, L., Zuckerman, M., Procaccia, A., Conitzer, V., Rosenschein, J.: Complexity of unweighted coalitional manipulation under some common voting rules. In: Proc. of 21st IJCAI, Int. Joint Conf. on Artificial Intelligence, pp. 348–353 (2009)
40. Walsh, T.: Uncertainty in preference elicitation and aggregation. In: Proc. of the 22nd National Conf. on AI. AAAI, Menlo Park (2007)
41. Walsh, T.: Complexity of terminating preference elicitation. In: 7th Int. Joint Conf. on Autonomous Agents and Multiagent Systems, IFAAMAS (2008)

Games, Norms and Strategic Notions in Information Flow Security[*]

Ron van der Meyden

University of New South Wales
meyden@cse.unsw.edu.au

Abstract. One of the concerns in the theory of computer security has been the question of what information an adversary is able to deduce about the secrets that a system is intended to maintain. Giving precise definitions of information flow security has proved to be a subtle matter. Some of the definitions that have been developed make explicit reference to strategic behaviour of the adversary. We present a perspective on these aspects of information security from epistemic logic and the theory of synthesis from logical specifications, and describe some recent results on the computational complexity of definitions of information flow security. Results concerning several types of games are drawn upon in the proofs of these complexity results. We also consider a normative aspect, viz, the use in implementations of access control permission policies to enforce an information flow security policy.

[*] Work supported by Australian Research Council Discovery Grant DP1097203.

Group Announcements: Logic and Games (Abstract of Invited Talk)

Thomas Ågotnes*

Department of Information Science and Media Studies,
University of Bergen
P.O. Box 7802, N-5020 Bergen, Norway
thomas.agotnes@infomedia.uib.no

Abstract. The logic of truthful public announcements has been studied as an extension of traditional epistemic logic. The topic of this talk is *group announcements*, truthful public announcements made simultaneously by the members of a group of agents. I will discuss group announcements from two sides. First, the logic of group announcements can be studied by adding quantification over group announcements to the logical language. Second, the game theory of group announcements can be studied by assuming that each agent has preferences over epistemic states (here, represented as an epistemic goal formula).

1 Group Announcements

Epistemic logic, the logic of knowledge, has been around since the 1960s [11,21], but has peaked in popularity recently – in particular in the fields of computer science, AI and multi-agent systems [7,14]. The key construct in (propositional, multi-agent) epistemic logic is of the form $K_i\phi$, expressing the fact that agent i knows ϕ.

One of the most prominent recent developments of epistemic logic is *dynamic* epistemic logic (DEL, see [20] for a recent overview). DEL describes the possible information-changing actions available to individual agents, and their knowledge pre- and post conditions. The perhaps simplest dynamic epistemic logic is *public announcement logic (* PAL*)* [16,8], which makes epistemic logic dynamic by adding modal operators $\langle\psi\rangle$, where ψ is a formula. The intended meaning of $\langle\psi\rangle\phi$ is that ψ is true and that after it is publicly announced, ϕ will be true.

The topic of this talk is *group announcements*; announcements made by a group of agents in the sense that each agent in the group truthfully and publically announces one formula. An agent can only make a truthful announcement if she knows the announcement to be true, and a group announcement for group G (a set of agents) is thus of the form

$$\bigwedge_{i\in G} K_i\phi_i$$

where ϕ_i, $i \in G$, is a formula.

* The talk is based on joint work with Hans van Ditmarsch, Philippe Balbiani and Pablo Seban.

Two questions that we think are interesting are:

1. *What are the logical principles of group announcements?* E.g.: What is the logical relationship between different statements about group announcements? How complex is it to reason about group announcements? Etc.
2. *What are the rational principles of group announcements?* I.e., which group announcements will rational agents actually make? This question naturally leads to a game-theoretic scenario.

2 Group Announcement Logic

A recent research direction in dynamic epistemic logic is *quantification* over formulae. Van Benthem [18] and Balbiani et al. [5] suggested adding the standard diamond to public announcement logic, with the interpretation that $\Diamond\phi$ means "there is an announcement after which ϕ". Or formally: there is a formula[1] ψ such that $\langle\psi\rangle\phi$ holds. The resulting logic is called Arbitrary Public Announcement Logic (APAL) [5,6]. In the interpretation of $\Diamond$ in APAL, however, there is no assumption about *who* makes the announcement – or indeed that it *can* be truthfully made by any agent in the system. Group announcements are exactly the announcements that can be made by the agents in the system.

Group Announcement Logic (GAL) [2,1] extends PAL with an operator $\langle G\rangle$, for every group of agents G, for quantifying over group announcements by G. The formula

$$\langle G\rangle\phi$$

has the intended meaning that there is a group announcement $\alpha = \bigwedge_{i\in G} K_i\phi_i$ for G such that $\langle\alpha\rangle\phi$ holds[2]. The $\langle G\rangle$ operator can be seen as a "coalitional ability" operator of the form known from Coalition Logic [15] and Alternating-time Temporal Logic [4], in the sense that $\langle G\rangle\phi$ means that G has the ability (here, by making a group announcement) to make ϕ come about. GAL thus bridges two active strands of research in logics for multi-agent systems, *viz.* dynamic epistemic logics and logics of coalitional ability.

As an example, the PAL formula

$$\langle K_a\phi_a\rangle\langle K_b\phi_b\rangle(K_a\psi \wedge K_b\psi \wedge \neg K_c\psi)$$

expresses the fact that a can make the announcement ϕ_a after which b can make the announcement ϕ_b after which both a and b will both know some formula ψ but c will not. In GAL, however, this can be weakened to

$$\langle a\rangle\langle b\rangle(K_a\psi \wedge K_b\psi \wedge \neg K_c\psi)$$

expressing the fact that a and b can make *some* announcements (in the same sequence) achieving the goal. We think that the GAL formalism can be useful to

[1] The quantification is restricted to the language without the $\Diamond$ operator to avoid circularity.

[2] Again, the quantification is restricted to the language without the $\langle G\rangle$ operators.

model check security protocols, e.g., to answer questions of the type "does there exist a protocol consisting of an announcement by a followed by an announcement by b achieving the goal" (where "announcement" means communication that is assumed to be intercepted).

I discuss how GAL can be used to express potentially interesting properties involving, e.g., sequences of group announcements and/or interaction properties between knowledge and ability. For example, it turns out that the property "there is a sequence, of arbitrary length, of group announcements by G, after which ϕ will be true" can be expressed in GAL.

By combining the knowledge and the ability operators, GAL can be used to express, e.g., $K_a\langle a\rangle\phi$ ("a knows that she is able to make ϕ true by some announcement") and $\langle a\rangle K_a\phi$ ("a can make some announcement after which she will know ϕ"). There are many subtleties involved in the combination of knowledge and (general, not necessarily announcement) ability operators [13,12]. One of them is the *de dicto*/*de re* distinction: knowing (*de dicto*) *that* you have the ability to achieve a goal (without necessarily knowing how) vs. knowing (*de re*) *how* you can achieve the goal (knowing which action will achieve the goal). It is not surprising that the former is expressed by $K_a\langle a\rangle\phi$; it is perhaps more surprising that the latter is in fact expressed[3] by $\langle a\rangle K_a\phi$: a knows that a particular announcement will make ϕ true if and only if she can make some announcement that will make her learn ϕ.

Key meta-logical results include a complete axiomatisation, and a characterisation of the complexity of the model checking problem (PSPACE-complete). Further details can be found in [1].

3 Public Announcement Games

An agent can typically choose between several truthful announcements in a given circumstance. Let us assume that each agent in a group choose announcements simultaneously. Which announcement will each agent choose, assuming that she is rational? And, thus, which group announcement will be made? In order to answer such questions, preferences over epistemic states (possibly of several different agents) must be assumed. Since epistemic states after a group announcement will depend on the announcements chosen by *all* the agents, we have a *game theoretic* scenario.

A simple model of agents' preferences is a (typically epistemic) *goal formula* γ_i for each agent – a formula that agent wants to be true. For example, a's goal might be $K_b\psi \wedge \neg K_c\psi$ – that b learns ψ without c learning it. Logical formulae are used in the same way in Boolean games [9,10] to represent binary preferences. A strategic form game can now be defined in a natural way: the strategies for an agent is the set of announcements she can make (true formulae of the form $K_i\phi$). A strategy profile is then a group announcement $\alpha = \bigwedge_{i\in N} K_i\phi_i$ (where N is the set of all agents) and an agent gets utility 1 if $\langle\alpha\rangle\gamma_i$ holds and 0 otherwise.

[3] Assuming the S5 properties of knowledge.

Such *public announcement games* are a particular type of strategic games with imperfect information, where there is an intimate relationship between strategies, knowledge, and utility.

I discuss properties of public announcement games, possible solution concepts, and the relationship between epistemic properties and game theoretic properties. An example of a simple interaction property is that if an agent's goal formula is in the *positive fragment* of the language [17,19] (essentially, only contains negation immediately preceding atomic propositions), then that agent knows *de re* that she has a weakly dominant strategy (there is a strategy she knows is weakly dominant).

Further details about public announcement games can be found in [3].

References

1. Ågotnes, T., Balbiani, P., van Ditmarsch, H.P., Seban, P.: Group announcement logic. Journal of Applied Logic 8(1), 62–81 (2010)
2. Ågotnes, T., van Ditmarsch, H.P.: Coalitions and announcements. In: Padgham, L., Parkes, D., Muller, J., Parsons, S. (eds.) Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, pp. 673–680 (May 2008)
3. Ågotnes, T., van Ditmarsch, H.: What will they say? – Public announcement Games. Synthese (Knowledge Rationality and Action) (to appear 2010)
4. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM 49, 672–713 (2002)
5. Balbiani, P., Baltag, A., van Ditmarsch, H.P., Herzig, A., Hoshi, T., De Lima, T.: What can we achieve by arbitrary announcements? A dynamic take on Fitch's knowability. In: Samet, D. (ed.) Proceedings of TARK XI, Louvain-la-Neuve, Belgium, pp. 42–51. Presses Universitaires de Louvain (2007)
6. Balbiani, P., Baltag, A., van Ditmarsch, H.P., Herzig, A., Hoshi, T., De Lima, T.: 'Knowable' as 'known after an announcement'. Review of Symbolic Logic 1(3), 305–334 (2008)
7. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press, Cambridge (1995)
8. Gerbrandy, J.D., Groeneveld, W.: Reasoning about information change. Journal of Logic, Language, and Information 6, 147–169 (1997)
9. Harrenstein, P.: Logic in Conflict. PhD thesis, Utrecht University (2004)
10. Harrenstein, P., van der Hoek, W., Meyer, J.-J.C., Witteveen, C.: Boolean games. In: van Benthem, J. (ed.) Proceeding of the Eighth Conference on Theoretical Aspects of Rationality and Knowledge (TARK VIII), Siena, Italy, pp. 287–298 (2001)
11. Hintikka, J.: Knowledge and Belief. Cornell University Press, Ithaca (1962)
12. Jamroga, W., Ågotnes, T.: Constructive knowledge: what agents can achieve under imperfect information. Journal of Applied Non-Classical Logics 17(4), 423–475 (2007)
13. Jamroga, W., van der Hoek, W.: Agents that know how to play. Fundamenta Informaticae 63, 185–219 (2004)
14. Meyer, J.-J.C., van der Hoek, W.: Epistemic Logic for AI and Computer Science. Cambridge Tracts in Theoretical Computer Science, vol. 41. Cambridge University Press, Cambridge (1995)

15. Pauly, M.: A modal logic for coalitional power in games. Journal of Logic and Computation 12(1), 149–166 (2002)
16. Plaza, J.A.: Logics of public communications. In: Emrich, M.L., Pfeifer, M.S., Hadzikadic, M., Ras, Z.W. (eds.) Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems: Poster Session Program, ORNL/DSRD-24, pp. 201–216. Oak Ridge National Laboratory (1989)
17. van Benthem, J.F.A.K.: One is a lonely number: on the logic of communication. Technical report, University of Amsterdam. ILLC Research Report PP-2002-27 (material presented at the Logic Colloquium 2002 (2002)
18. van Benthem, J.F.A.K.: What one may come to know. Analysis 64(2), 95–105 (2004)
19. van Ditmarsch, H.P., Kooi, B.P.: The secret of my success. Synthese 151, 201–232 (2006)
20. van Ditmarsch, H.P., van der Hoek, W., Kooi, B.P.: Dynamic Epistemic Logic. Synthese Library, vol. 337. Springer, Heidelberg (2007)
21. von Wright, G.H.: An Essay in Modal Logic. North Holland, Amsterdam (1951)

Strong Equivalence in Argumentation*

Stefan Woltran

Institute of Information Systems
Vienna University of Technology, Austria
woltran@dbai.tuwien.ac.at

Abstract. The problem of equivalence has received substantial attention in the knowledge-representation (KR) community in the past several years. This is due to the fact that the replacement theorem from classical logic does not necessarily hold in typical (non-monotonic) KR formalisms. In fact, the problem is as follows: Consider a theory S is replaced by another theory S' within a larger knowledge base T. Naturally, one wants to ensure that the resulting knowledge base $(T \setminus S) \cup S'$ has the same meaning as T. But this is not guaranteed by standard equivalence between S and S' under nonmonotonic semantics, and therefore, stronger notions of equivalence are required. In particular, the following definition of equivalence guarantees that a replacement as discussed above is faithful: two theories S and S' are called strongly equivalent, if and only if $S \cup T$ and $S' \cup T$ have the same same meaning for each theory T.

In this, talk we first give a brief overview of seminal results on strong equivalence from the areas of datalog and answer-set programming. Then, we focus on argumentation and present recent characterisations for strong equivalence between argumentation frameworks with respect to the most important semantics proposed for such frameworks. We also discuss some variants of strong equivalence, which are defined in terms of acceptance. Since argumentation is an inherently dynamic process, it is of great importance to understand the effect of incorporating new information into given argumentation frameworks. By its definition, strong equivalence gives some fundamental insight into this issue.

* This work is supported by the Vienna Science and Technology Fund (WWTF) under grant ICT08-028.

Hypergraphs of Multiparty Secrets

Sara Miner More and Pavel Naumov

Department of Mathematics and Computer Science
McDaniel College, Westminster, Maryland 21157, USA
{smore,pnaumov}@mcdaniel.edu

Abstract. The paper considers interdependencies between secrets in a multiparty system. Each secret is assumed to be known only to a certain fixed set of parties. These sets can be viewed as edges of a hypergraph whose vertices are the parties of the system. The main result is a complete and decidable logical system that describes interdependencies that may exist on a fixed hypergraph. The properties of interdependencies are defined through a multi-argument relation called *independence*, which is a generalization of a binary relation also known as nondeducibility.

1 Introduction

In this paper, we study properties of interdependencies between pieces of information. We call these pieces *secrets* to emphasize the fact that they might be known to some parties and unknown to others. Below, we first describe two relations for expressing interdependencies between secrets. Next, we discuss these relations in the context of collaboration networks which specify the available communication channels for the parties establishing the secrets.

Relations on Secrets. One of the simplest relations between two secrets is *functional dependence*, which we denote by $a \rhd b$. It means that the value of secret a reveals the value of secret b. This relation is reflexive and transitive. A more general and less trivial form of functional dependence is functional dependence between sets of secrets. If A and B are two sets of secrets, then $A \rhd B$ means that, together, the values of all secrets in A reveal the values of all secrets in B. Armstrong [1] presented a sound and complete set of axioms for this relation.

These axioms are known in database literature as Armstrong's axioms [2, p. 81]. Beeri, Fagin, and Howard [3] suggested a variation of Armstrong's axioms that describe properties of multi-valued dependency.

Not all dependencies between two secrets are functional. For example, if secret a is a pair $\langle x, y\rangle$ and secret b is a pair $\langle y, z\rangle$, then there is an interdependence between these secrets in the sense that not every value of secret a is compatible with every value of secret b. However, neither $a \rhd b$ nor $b \rhd a$ is necessarily true. If there is no interdependence at all between two secrets, then we will say that the two secrets are *independent*. In other words, secrets a and b are independent if any possible value of secret a is compatible with any possible value of secret b. We denote this relation between two secrets by $[a, b]$. This relation was introduced by

Sutherland [4] and is also known as *nondeducibility* in the study of information flow. Halpern and O'Neill [5] proposed a closely related notion called f-secrecy.

Like functional dependence, independence also can be generalized to relate two sets of secrets. If A and B are two such sets, then $[A, B]$ means that any consistent combination of values of the secrets in A is compatible with any consistent combination of values of the secrets in B. Note that "consistent combination" is an important condition here, since some interdependence may exist between secrets in set A even while the entire set of secrets A is independent from the secrets in set B. The following is an example of a non-trivial property expressible in this language:

$$[A\cup B\,,\, C] \to ([A\,,\, B] \to [A\,,\, B\cup C]).$$

A sound and complete axiomatization of all such properties was given by More and Naumov [6]. Essentially the same axioms were shown by Geiger, Paz, and Pearl [7] to provide a complete axiomatization of the independence relation between sets of random variables in probability theory. A complete logical system that combines independence and functional dependence predicates for *single* secrets was described by Kelvey, More, Naumov, and Sapp [8].

Secrets in Networks. So far, we have assumed that the values of secrets are determined a priori. In the physical world, however, secret values are often generated, or at least disseminated, via interaction between several parties. Quite often such interactions happen over a network with fixed topology. For example, in social networks, interaction between nodes happens along connections formed by friendship, kinship, financial relationship, etc. In distributed computer systems, interaction happens over computer networks. Exchange of genetic information happens along the edges of the genealogical tree. Corporate secrets normally flow over an organization chart. In cryptographic protocols, it is often assumed that values are transmitted over well-defined channels. On social networking websites, information is shared between "friends". Messages between objects on an UML interaction diagram are sent along connections defined by associations between the classes of the objects.

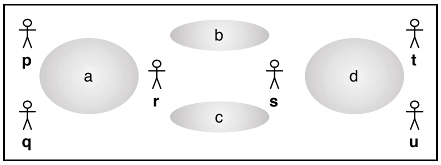


Fig. 1. Collaboration network H_0

In this paper, we will use the notion of *collaboration network* to refer to the topological structure that specifies which secrets are known to which parties. An example of such network is given on Figure 1. In this network, parties p, q and r share secret a; parties r and s share secrets b and c; and parties s, t and u share secret d. If different secrets are established completely independently, then possession of one or several of these secrets reveals no information about the other secrets. Assume, however, that secrets are not picked completely independently. Instead, each party with access to multiple secrets may enforce some desired interdependence between the values of these secrets. These "local" interdependencies between secrets known to a single party may result in a "global" interdependence between several secrets, not all of which are known to any single party. Given the fixed topology of the collaboration network, we study what global interdependencies between secrets may exist in the system.

We will say that the local interdependencies define a *protocol*. For the collaboration network H_0 depicted in Figure 1, for example, we can imagine the following protocol. Parties p, q and r together pick a random value a from set $\{0,1\}$. Next, party r chooses values b and c from $\{0,1\}$ in such a way that $a = b + c \mod 2$ and sends both of these values to party s. Party s computes $d = b + c \mod 2$ and shares value d with parties t and u. In this protocol, it is clear that the values of a and d will always match. Hence, for this specific protocol, we can say that $a \rhd d$ and $d \rhd a$, but at the same time, $[a, b]$ and $[a, c]$.

The functional dependence and independence examples above are for a single protocol, subject to a particular set of local interdependencies between secrets. If the network remains fixed, but the protocol is changed, then secrets which were previously interdependent may no longer be so, and vice versa. For example, for network H_0 above, the claim $a \rhd d$ will no longer be true if, say, party s switches from enforcing the local condition $d = b + c \mod 2$ to enforcing the local condition $d = b$. In this paper, we study properties of relations between secrets that follow from the topological structure of the collaboration network, no matter which specific protocol is used. Examples of such properties for network H_0 are $a \rhd d \to b, c \rhd d$ and $[\{a\}, \{b, c\}] \to [a, d]$.

A special case of the collaboration network is an undirected graph collaboration network in which any secret is shared between at most two parties. In an earlier work [9], we considered this special case and gave a complete axiomatic system for the independence relation between single secrets in that setting. In fact, we axiomatized a slightly more general relation $[a_1, a_2, \ldots, a_n]$ between multiple *single* secrets, which means that any possible values of secrets $a_1, \ldots, a_n$ can occur together.

In a more recent work, currently under review, we developed a complete logical system that describes the properties of the functional dependence relation $A \rhd B$ between sets of secrets over graph collaboration networks. This system includes Armstrong's axioms and a new Gateway axiom that captures properties of functional dependence specific to the topology of the collaboration network.

In the current paper, we focus on independence and generalize our results from collaboration networks defined by standard graphs to those defined by hypergraphs. That is, we examine networks where, as in Figure 1, a secret can be shared between more than two parties. In this setting, we give a complete and decidable system of axioms for the relation $[a_1, a_2, \ldots, a_n]$. In terms of the proof of completeness, the most significant difference between the earlier work [9] and this one is in the construction of the parity protocol in Section 7.1.

2 Hypergraphs

A collaboration network where a single secret can be shared between multiple parties can be described mathematically as a hypergraph in which vertices are parties and (hyper)edges are secrets. In this section, we will introduce the hypergraph terminology that is used later in the paper.

Definition 1. *A hypergraph is pair $H = \langle V, E\rangle$, where*

1. *V is a finite set, whose elements are called "vertices".*
2. *E is a finite multiset of non-empty subsets of V. Elements of E are called "edges". Elements of an edge are called the "ends" of the edge.*

Note that we use "mulitisets" in the above definition to allow for multiple edges between the same set of ends. Also note that, as is common in hypergraph literature [10, p. 1], we exclude empty edges from consideration.

Definition 2. *For any set of vertices V' of a hypergraph H, by $Out(V')$ we mean the set of edges in H that contain ends from both set V' and the complement of V'. By $In(V')$ we mean the set of edges in H that contain only ends from V'.*

From the collaboration network perspective, V' is a group of parties, $Out(V')$ is the public interface of this group (secrets that the group members share with non-members) and $In(V')$ is the set of secrets only known within group V'. For example, for the collaboration network defined by hypergraph H_0 on Figure 1, if $V' = \{r, s\}$, then $Out(V') = \{a, d\}$ and $In(V') = \{b, c\}$.

A *path* in a hypergraph is an alternating sequence of edges and vertices in which adjacent elements are incident. It will be convenient to assume that paths start and end with edges rather than with vertices. Paths will be assumed to be simple, in the sense that no edge or vertex is repeated in the path, with the exception that the last edge in the path may be the same as the first. In this case, the path is called cyclic. For example, a, r, b, s, c is a path in H_0 of Figure 1.

Definition 3. *A gateway between sets of edges A and B is a set of edges G such that every path from A to B contains at least one edge from G.*

For instance, set $\{b, c\}$ is a gateway between single-element sets $\{a\}$ and $\{d\}$ on the hypergraph H_0 from Figure 1. Note also that in the definition above, sets A, B, and G are not necessarily disjoint. Thus, for example, for any set of edges A, set A is a gateway between A and itself. Also, note that the empty set is a gateway between any two components of the hypergraph that are not connected one to another.

Definition 4. *If X is an arbitrary set of vertices of a hypergraph $H = \langle V, E\rangle$, then the truncation of set X from H is a hypergraph $H' = \langle V \setminus X, E'\rangle$, where*

$$E' = \{e \setminus X \mid e \in E \text{ and } e \setminus X \neq \varnothing\}.$$

Truncated hypergraph H' is also commonly [10, p. 3] referred to as the subhypergraph of H induced by the set of vertices $V \setminus X$.

3 Protocol: A Formal Definition

Definition 5. *A semi-protocol over a hypergraph $H = \langle V, E\rangle$ is a pair $\mathcal{P} = \langle Val, Loc\rangle$ such that*

1. *$Val(e)$ is an arbitrary set of "values" for each edge $e \in E$,*
2. *$Loc = \{Loc_v\}_{v \in V}$ is a family of relations, indexed by vertices (parties) of the hypergraph H, which we call "local conditions". If $e_1, \dots e_k$ is the list of all edges incident with vertex v, then $Loc_v \subseteq Val(e_1) \times \cdots \times Val(e_k)$.*

Definition 6. *A run of a semi-protocol $\langle Val, Loc\rangle$ is a function r such that*

1. *$r(e) \in Val(e)$ for any edge $e \in E$,*
2. *If $e_1, \dots e_k$ is the list of all edges incident with vertex $v \in V$, then the statement $Loc_v(r(e_1), \dots, r(e_k))$ is true.*

Definition 7. *A protocol is any semi-protocol that has at least one run.*

The set of all runs of a protocol $\mathcal{P}$ is denoted by $\mathcal{R}(\mathcal{P})$.

Definition 8. *A protocol $\mathcal{P} = \langle Val, Loc\rangle$ is called finite if the set $Val(e)$ is finite for every edge e of the hypergraph.*

The following definition of independence is identical to the one given earlier [9] for standard graphs.

Definition 9. *A set of edges $Q = \{q_1, \dots, q_k\}$ is independent under protocol $\mathcal{P}$ if for any runs $r_1, \dots, r_k \in \mathcal{R}(\mathcal{P})$ there is a run $r \in \mathcal{R}(\mathcal{P})$ such that $r(q_i) = r_i(q_i)$ for any $i \in \{1, \dots, k\}$.*

4 Language of Secrets

By $\Phi(H)$, we denote the set of all collaboration network properties specified by hypergraph H that are expressible through the independence predicate. More formally, $\Phi(H)$ is a minimal set of formulas defined recursively as follows: (i) for any finite subset A of the set of edges of hypergraph H, formula $[A]$ is in $\Phi(H)$, (ii) the false constant $\perp$ is in set $\Phi(H)$, and (iii) for any formulas ϕ and $\psi \in \Phi(H)$, the implication $\phi \to \psi$ is in $\Phi(H)$. As usual, we assume that conjunction, disjunction, and negation are defined through $\to$ and $\perp$.

Next, we define a relation $\vDash$ between a protocol and a formula from $\Phi(H)$. Informally, $\mathcal{P} \vDash \phi$ means that formula ϕ is true under protocol $\mathcal{P}$.

Definition 10. *For any protocol $\mathcal{P}$ over a hypergraph H, and any formula $\phi \in \Phi(H)$, we define the relation $\mathcal{P} \vDash \phi$ recursively as follows:*

1. *$\mathcal{P} \nvDash \perp$,*
2. *$\mathcal{P} \vDash [A]$ if the set of edges A is independent under protocol $\mathcal{P}$,*
3. *$\mathcal{P} \vDash \phi_1 \to \phi_2$ if $\mathcal{P} \nvDash \phi_1$ or $\mathcal{P} \vDash \phi_2$.*

In this paper, we study the formulas $\phi \in \Phi(H)$ that are true under *any* protocol $\mathcal{P}$ over a fixed hypergraph H. Below we describe a formal logical system for such formulas. This system, like earlier systems defined by Armstrong [1], More and Naumov [11,9] and by Kelvey, More, Naumov, and Sapp [8], belongs to the set of deductive systems that capture properties of secrets. In general, we refer to such systems as *logics of secrets*. Since this paper is focused on only one such system, here we call it *the logic of secrets* of hypergraph H.

5 Logic of Secrets

In this section we will define a formal deductive system for the logic of secrets and give examples of proofs in this system. The soundness, completeness, and decidability of this system will be shown in the next two sections.

5.1 Formal System: Axioms and Rules

For any hypergraph $H = \langle V, E\rangle$, we will write $H \vdash \phi$ to state that formula $\phi \in \Phi(H)$ is provable in the logic of secrets of hypergraph H. The deductive system for this logic, in addition to propositional tautologies and Modus Ponens inference rule, consists of the *Small Set* axiom, the *Gateway* axiom, and the *Truncation* inference rule, defined below:

Small Set Axiom. $H \vdash [A]$, where $A \subseteq E$ and $|A| < 2$.

Gateway Axiom. $H \vdash [A, G] \rightarrow ([B] \rightarrow [A, B])$, where G is a gateway between sets of edges A and B such that $A \cap G = \varnothing$.

Truncation Rule. If $H' \vdash \phi$, then $H \vdash [Out(X)] \rightarrow \phi$, where H' is obtained from H by the truncation of set X.

The soundness of this system will be demonstrated in Section 6.

Theorem 1 (monotonicity). $H \vdash [A] \rightarrow [B]$, *for any hypergraph H and any subset B of a set of edges A of hypergraph H.*

Proof. Consider sets B and $\varnothing$. Since there are no paths connecting these sets, any set of edges is a gateway between these sets. In particular $A \setminus B$ is such a gateway. Taking into account that sets B and $A \setminus B$ are disjoint, by the Gateway axiom, $H \vdash [B, A \setminus B] \rightarrow ([\varnothing] \rightarrow [B])$. By the Small Set axiom, $H \vdash [B, A \setminus B] \rightarrow [B]$. By assumption $B \subseteq A$, we get $H \vdash [A] \rightarrow [B]$. □

5.2 Proof Examples

Our first example refers to hypergraph H_1 in Figure 2. It shows parties p and q that have secrets a and c, respectively, that they do not share with each other, and secret b that they both know.

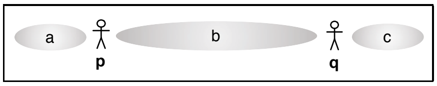


Fig. 2. Hypergraph H_1

Theorem 2. $H_1 \vdash [a, b] \to [a, c]$.

Proof. Set $\{b\}$ is a gateway between sets $\{a\}$ and $\{c\}$. Thus, by the Gateway axiom, $H_1 \vdash [a, b] \to ([c] \to [a, c])$. At the same time, $H_1 \vdash [c]$, by the Small Set axiom. Therefore, $H_1 \vdash [a, b] \to [a, c]$. □

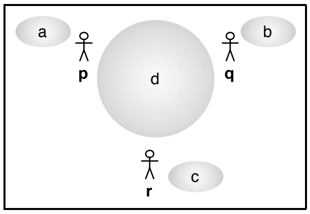


Fig. 3. Hypergraph H_2

Our second example deals with the collaboration network defined by hypergraph H_2 on Figure 3. Here, parties p, q, and r have individual secrets a, b, c, and together share secret d.

Theorem 3. $H_2 \vdash [a, d] \to ([b, d] \to [a, b, c])$.

Proof. Note that set $\{d\}$ is a gateway between sets $\{a\}$ and $\{b, d\}$. Thus, by the Gateway axiom,

$$H_2 \vdash [a, d] \to ([b, d] \to [a, b, d]). \quad (1)$$

Next, observe that set $\{d\}$ is a gateway between sets $\{a, b\}$ and $\{c\}$. Thus, by the Gateway axiom, $H_2 \vdash [a, b, d] \to ([c] \to [a, b, c])$. By the Small Set axiom, $H_2 \vdash [c]$. Hence,

$$H_2 \vdash [a, b, d] \to [a, b, c]. \quad (2)$$

From statements (1) and (2), it follows that $H_2 \vdash [a, d] \to ([b, d] \to [a, b, c])$. □

Our third and final example refers to hypergraph H_3 depicted in Figure 4. In the proof we will also refer to hypergraph H_3', shown in the same figure, which is the result of the truncation of set $\{q, r, u, v\}$ from hypergraph H_3.

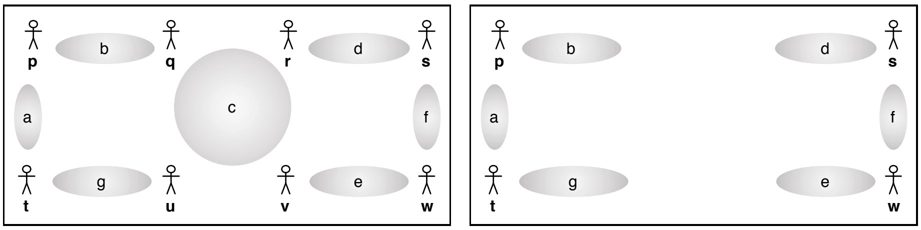


Fig. 4. Hypergraphs H_3 (left) and H_3' (right)

Theorem 4. $H_3 \vdash [b, d, g, e] \to [a, f]$.

Proof. Note that in the truncated hypergraph H_3', the empty set is a gateway between the single element sets $\{a\}$ and $\{f\}$. Thus, by the Gateway axiom, $H_3' \vdash [a] \to ([f] \to [a, f])$. By the Small Set axiom, $H_3' \vdash [a]$ and $H_3' \vdash [f]$. Hence, $H_3' \vdash [a, f]$. By the Truncation rule, $H_3 \vdash [Out(q, r, u, v)] \to [a, f]$. Since $Out(q, r, u, v) = \{b, d, g, e\}$, we get $H_3 \vdash [b, d, g, e] \to [a, f]$. □

6 Soundness

The proof of soundness, particularly for the Gateway axiom and Truncation rule, is non-trivial. For each axiom and inference rule, we provide its justification as a separate theorem.

Theorem 5 (Small Set). *For any hypergraph $H = \langle V, E\rangle$ and any set of edges A that has at most one element, if $\mathcal{P}$ is an arbitrary protocol over H, then $\mathcal{P} \vDash [A]$.*

Proof. If $A = \varnothing$, then $\mathcal{P} \vDash [A]$ follows from the existence of at least one run of any protocol (see Definition 7). If $A = \{a_1\}$, consider any run $r_1 \in \mathcal{R}(\mathcal{P})$. Pick r to be r_1. This guarantees that $r(a_1) = r_1(a_1)$. □

Theorem 6 (Gateway). *For any hypergraph $H = \langle V, E\rangle$, and any gateway G between sets of edges A and B, if $\mathcal{P} \vDash [A, G]$, $\mathcal{P} \vDash [B]$, and $A \cap G = \varnothing$, then $\mathcal{P} \vDash [A, B]$.*

Proof. Assume $\mathcal{P} \vDash [A, G]$, $\mathcal{P} \vDash [B]$, and $A \cap G = \varnothing$. Let $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_k\}$. Consider any $r_1, \dots, r_{n+k}$. It will be sufficient to show that there is $r \in \mathcal{R}(\mathcal{P})$ such that $r(a_i) = r_i(a_i)$ for any $i \le n$ and $r(b_i) = r_{n+i}(b_i)$ for any $i \le k$. By the assumption $\mathcal{P} \vDash [B]$, there is $r_b \in \mathcal{R}(\mathcal{P})$ such that

$$r_b(b_i) = r_{n+i}(b_i) \qquad \text{for any } i \le k. \tag{3}$$

By the assumptions $\mathcal{P} \vDash [A, G]$ and $A \cap G = \varnothing$, there must be a run r_a such that

$$r_a(c) = \begin{cases} r_i(c) \text{ if } c = a_i \text{ for } i \le n, \\ r_b(c) \text{ if } c \in G. \end{cases} \tag{4}$$

Next, consider hypergraph $H' = \langle V, E \setminus G\rangle$. By the definition of a gateway, no single connected component of hypergraph H' can contain edges from set A and set $B \setminus G$ at the same time. Let us divide all connected components of H' into two subhypergraphs H'_a and H'_b such that H'_a contains no edges from $B \setminus G$ and H'_b contains no edges from A. Components that do not contain edges from either A or $B \setminus G$ can be arbitrarily assigned to either H'_a or H'_b.

By definition (4), runs r_a and r_b agree on each edge of the gateway G. We will now construct a combined run r by "sewing" together portions of r_a and r_b with the "stitches" placed along gateway G. Formally,

$$r(c) = \begin{cases} r_a(c) & \text{if } c \in H_a, \\ r_a(c) = r_b(c) & \text{if } c \in G, \\ r_b(c) & \text{if } c \in H_b. \end{cases} \tag{5}$$

Let us first prove that r is a valid run of the protocol $\mathcal{P}$. For this, we need to prove that it satisfies local conditions Loc_v at every vertex v. Without loss of generality, assume that $v \in H'_a$. Hence, on all edges incident with v, run r agrees with run r_a. Thus, run r satisfies Loc_v simply because r_a does.

Next, we will show that $r(a_i) = r_i(a_i)$ for any $i \leq n$. Indeed, by equations (4) and (5), $r(a_i) = r_a(a_i) = r_i(a_i)$. Finally, we will need to show that $r(b_i) = r_{n+i}(b_i)$ for any $i \leq k$. This, however, trivially follows from equation (3) and equation (5). □

Theorem 7 (Truncation). *Assume that hypergraph H' is obtained from H by the truncation of set X and that $\phi \in \Phi(H')$. If $\mathcal{P}' \vDash \phi$ for any protocol $\mathcal{P}'$ over hypergraph H', then $\mathcal{P} \vDash [Out(X)] \to \phi$ for any protocol $\mathcal{P}$ over hypergraph H.*

Proof. Suppose that there is a protocol $\mathcal{P}$ over H such that $\mathcal{P} \vDash [Out(X)]$, but $\mathcal{P} \nvDash \phi$. We will construct a protocol $\mathcal{P}'$ over H' such that $\mathcal{P}' \nvDash \phi$.

Let $\mathcal{P} = \langle Val, Loc \rangle$. Note that, for any edge e, not all values from $Val(e)$ may actually be used in the runs of this protocol. Some values could be excluded by the particular local conditions of $\mathcal{P}$. To construct protocol $\mathcal{P}' = \langle Val', Loc' \rangle$ over hypergraph H', for any edge e of H' we define $Val'(e)$ as the set of values that are actually used by at least one run of the protocol $\mathcal{P}$:

$$Val'(e) = \{r(e) \mid r \in \mathcal{R}(\mathcal{P})\}.$$

The local condition Loc'_v at any vertex v of hypergraph H' is the same as under protocol $\mathcal{P}$. To show that protocol $\mathcal{P}'$ has at least one run, notice that the restriction of any run of $\mathcal{P}$ to edges in H' constitutes a valid run of $\mathcal{P}'$.

Lemma 1. *For any run $r' \in \mathcal{R}(\mathcal{P}')$ there is a run $r \in \mathcal{R}(\mathcal{P})$ such that $r(e) = r'(e)$ for each edge e in hypergraph H'.*

Proof. Consider any run $r' \in \mathcal{R}(\mathcal{P}')$. By definition of Val', for any $e \in Out(X)$ there is a run $r_e \in \mathcal{R}(\mathcal{P})$ such that $r'(e) = r_e(e)$. Since $\mathcal{P} \vDash [Out(X)]$, there is a run $r_X \in \mathcal{R}(\mathcal{P})$ such that $r_X(e) = r_e(e) = r'(e)$ for any $e \in Out(X)$.

We will now construct a combined run $r \in \mathcal{R}(\mathcal{P})$ by "sewing" together r_X and r' with the "stitches" placed in set $Out(X)$. Formally,

$$r(e) = \begin{cases} r_X(e) & \text{if } e \in In(X), \\ r_X(e) = r'(e) & \text{if } e \in Out(X), \\ r'(e) & \text{otherwise.} \end{cases}$$

We just need to show that r satisfies Loc_v at every vertex v of hypergraph H. Indeed, if $v \in X$, then run r is equal to r_X on all edges incident with v. Thus, it satisfies the local condition because run r_X does. Alternatively, if $v \notin X$, then run r is equal to run r' on all edges incident with v. Since r' satisfies local condition Loc'_v and, by definition, $Loc'_v \equiv Loc_v$, we can conclude that r again satisfies condition Loc_v.

Lemma 2. *$\mathcal{P} \vDash [Q]$ if and only if $\mathcal{P}' \vDash [Q]$, for any set of edges Q in H'.*

Proof. Assume first that $\mathcal{P} \vDash [Q]$ and consider any runs $r'_1, \dots, r'_n \in \mathcal{R}(\mathcal{P}')$. We will construct a run $r' \in \mathcal{R}(\mathcal{P}')$ such that $r'(q_i) = r'_i(q_i)$ for every $i \in \{1, \dots, n\}$. Indeed, by Lemma 1, there are runs $r_1, \dots, r_n \in \mathcal{R}(\mathcal{P})$ that match runs $r'_1, \dots, r'_n$

on all edges in H'. By the assumption that $\mathcal{P} \vDash [Q]$, there must be a run $r \in \mathcal{R}(\mathcal{P})$ such that $r(q_i) = r_i(q_i)$ for all $i \in \{1, \ldots, n\}$. Hence, $r(q_i) = r_i(q_i) = r'_i(q_i)$ for all $i \in \{1, \ldots, n\}$. Let r' be a restriction of run r to the edges in H'. Since the local conditions of protocols $\mathcal{P}$ and $\mathcal{P}'$ are the same, $r' \in \mathcal{R}(\mathcal{P}')$. Finally, we notice that $r'(q_i) = r(q_i) = r'_i(q_i)$ for any $i \in \{1, \ldots, k\}$.

Next, assume that $\mathcal{P}' \vDash [Q]$ and consider any runs $r_1, \ldots, r_n \in \mathcal{R}(\mathcal{P})$. We will show that there is a run $r \in \mathcal{R}(\mathcal{P})$ such that $r(q_i) = r_i(q_i)$ for all $i \in \{1, \ldots, n\}$. Indeed, let $r'_1, \ldots, r'_n$ be the restrictions of runs $r_1, \ldots, r_n$ to the edges in H'. Since the local conditions of these two protocols are the same, $r'_1, \ldots, r'_n \in \mathcal{R}(\mathcal{P}')$. By the assumption that $\mathcal{P}' \vDash [Q]$, there is a run $r' \in \mathcal{R}(\mathcal{P}')$ such that $r'(q_i) = r'_i(q_i) = r_i(q_i)$ for all $i \in \{1, \ldots, n\}$. By Lemma 1, there is a run $r \in \mathcal{R}(\mathcal{P})$ that matches r' everywhere in H'. Therefore, $r(q_i) = r'(q_i) = r_i(q_i)$ for all $i \in \{1, \ldots, n\}$.

Lemma 3. *For any formula $\psi \in \Phi(H')$, $\mathcal{P} \vDash \psi$ if and only if $\mathcal{P}' \vDash \psi$.*

Proof. We use induction on the complexity of ψ. The base case follows from Lemma 2, and the induction step is trivial.

The statement of Theorem 7 immediately follows from Lemma 3. □

7 Completeness

Our main result is the following completeness theorem for the logic of secrets:

Theorem 8. *For any hypergraph H, if $\mathcal{P} \vDash \phi$ for all finite protocols $\mathcal{P}$ over H, then $H \vdash \phi$.*

We prove this theorem by contrapositive. At the core of this proof is the construction of a finite protocol. This protocol will be formed as a composition of several simpler protocols, where each of the simpler protocols is defined recursively. The base case of this recursive definition comes from the family of "parity" protocols $\{\mathcal{P}_A\}_A$ defined below.

7.1 Parity Protocol $\mathcal{P}_A$

Let $H = \langle V, E \rangle$ be a hypergraph and A be a subset of E. We define the "parity protocol" $\mathcal{P}_A$ over H as follows. The set of values of any edge e in hypergraph H is $\{0, 1\}^e$, or the set of boolean functions on e. Thus, a run r of the protocol will be a function that maps an edge into a function from the ends of this edge into boolean values: $r(e)(v) \in \{0, 1\}$, where e is an edge

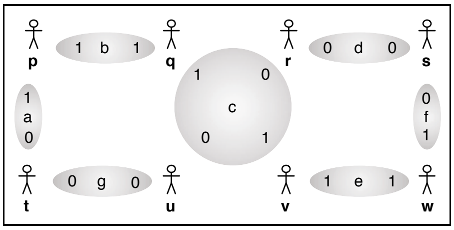


Fig. 5. Parity protocol run on graph H_3

and v is an end of e. It will be more convenient, however, to think about a run as a two-argument function $r(e, v) \in \{0, 1\}$. We will graphically represent this function by placing boolean values at each end of each edge of the hypergraph. See Figure 5 for an example.

Not all assignments of boolean values to the ends of an edge e will be permitted in the parity protocol. Namely, if $e \notin A$, then the sum of all values assigned to the ends of e must be equal to zero modulo 2:

$$\sum_{v\in e} r(e, v) = 0 \mod 2. \tag{6}$$

However, if $e \in A$, then no restriction on the assignment of boolean values to the ends of e will be imposed. This defines the set of values $Val(e)$ for each edge e under the protocol $\mathcal{P}_A$.

The second restriction on the runs will require that the sum of all values assigned to ends incident with any vertex v is also equal to zero modulo 2:

$$\sum_{e\in E(v)} r(e, v) = 0 \mod 2, \tag{7}$$

where $E(v)$ is the set of all edges incident with v. The latter restriction specifies the local condition Loc_v for each vertex v. The protocol $\mathcal{P}_A$ is now completely defined. We just need to prove the existence of at least one run that satisfies all local conditions. Indeed, consider the run r such that $r(e, v) = 0$ for any end v of any edge e. This run clearly satisfies restrictions (6) and (7).

Theorem 9. *For any run r of the parity protocol $\mathcal{P}_A$,*

$$\sum_{e\in A}\sum_{v\in e} r(e, v) = 0 \mod 2.$$

Proof. Let $H = \langle V, E\rangle$. Using equations (7) and (6),

$$\sum_{e\in A}\sum_{v\in e} r(e, v) = \sum_{e\in E}\sum_{v\in e} r(e, v) - \sum_{e\notin A}\sum_{v\in e} r(e, v) =$$
$$= \sum_{v\in V}\sum_{e\in E(v)} r(e, v) - \sum_{e\notin A} 0 = \sum_{v\in V} 0 - 0 = 0 \mod 2.$$
□

Recall that we defined a path to start and end with edges rather than vertices.

Definition 11. *For any path $\pi = e_0, v_1, e_1, \ldots, e_n$ in a hypergraph H and any run r of the parity protocol $\mathcal{P}_A$, we define r_π as*

$$r_\pi(e, v) = \begin{cases} 1 - r(e, v) & \textit{if } e = e_i, v = v_{i+1} \textit{ or } v = v_i, e = e_{i+1} \textit{ for some } i < n, \\ r(e, v) & \textit{otherwise.}\end{cases}$$

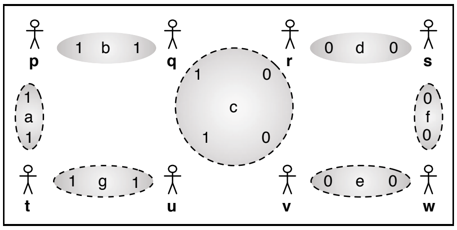


Fig. 6. Run r_π

Informally, r_π is obtained from r by "flipping" the boolean value at each end along path π. For example, Figure 6 depicts the "flipped" run r_π, where π is $a, t, g, u, c, v, e, w, f$, and run r is the run from Figure 5. The edges along path π are indicated with dashed lines in Figure 6.

Theorem 10. *For any $r \in \mathcal{P}_A$ and any path π in a hypergraph H, if π is a cycle or starts and ends with edges that belong to set A, then $r_\pi \in \mathcal{R}(\mathcal{P}_A)$.*

Proof. Run r_π satisfies condition (6) because r_π is different from r at exactly two ends of any non-terminal edge of path π. The same run r_π satisfies condition (7) at every vertex v of the hypergraph, because path π includes either zero or two ends of edges incident at vertex v. □

Theorem 11. *If $|A| > 1$ and hypergraph H is connected, then for any $e \in A$ and any $g \in \{0, 1\}$ there is a run $r \in \mathcal{R}(\mathcal{P}_A)$ such that $\sum_{v\in e} r(e, v) = g \mod 2$.*

Proof. Each protocol has at least one run. Let r be a run of the protocol $\mathcal{P}_A$. Suppose that $\sum_{v\in e} r(e, v) \neq g \mod 2$. Since $|A| > 1$ and hypergraph H is connected, there is a path π that connects edge e with an edge $a \in A$ such that $a \neq e$. Notice that $\sum_{v\in e} r_\pi(e, v) = \sum_{v\in e} r(e, v) + 1 = g \mod 2$. □

Theorem 12. *If $|A| > 1$ and hypergraph H is connected, then $\mathcal{P}_A \nvDash [A]$.*

Proof. Let $A = \{a_1, \dots, a_k\}$. Pick any boolean values $g_1, \dots, g_k$ such that $g_1 + \dots + g_k = 1 \mod 2$. By Theorem 11, there are runs $r_1, \dots, r_k \in \mathcal{R}(\mathcal{P}_A)$ such that $\sum_{v\in a_i} r_i(a_i, v) = g_i \mod 2$ for any $i \le k$. If $\mathcal{P}_A \vDash [A]$, then there is a run $r \in \mathcal{R}(\mathcal{P}_A)$ such that $r(a_i, v) = r_i(a_i, v)$ for any $v \in a_i$ and any $i \le k$. Therefore, $\sum_{v\in a_1} r(a_1, v) + \dots + \sum_{v\in a_k} r(a_k, v) = \sum_{v\in a_1} r_1(a_1, v) + \dots + \sum_{v\in a_k} r_k(a_k, v) = g_1 + \dots + g_k = 1 \mod 2$. This contradicts Theorem 9. □

Theorem 13. *If A and B are two sets of edges of a hypergraph $H = \langle V, E\rangle$, such that each connected component of hypergraph $\langle V, E \setminus B\rangle$ contains at least one edge from A, then $\mathcal{P}_A \vDash [B]$.*

Proof. Let $B = \{b_1, \dots, b_k\}$. Consider any runs $r_1, \dots, r_k \in \mathcal{R}(\mathcal{P}_A)$. We will prove that there is a run $r \in \mathcal{R}(\mathcal{P}_A)$ such that $r(b_i, v) = r_i(b_i, v)$ for any $v \in b_i$ and any $i \le k$. Indeed, protocol $\mathcal{P}_A$ has at least one run. Call it $\hat{r}$. We will modify run $\hat{r}$ to satisfy the condition $\hat{r}(b_i, v) = r_i(b_i, v)$ for any $v \in b_i$ and any $i \le k$. Our modification will consist of repeating the following procedure for each $i \le k$ and each $v \in b_i$ such that $\hat{r}(b_i, v) \neq r_i(b_i, v)$:

1. If $b_i \in A$, then, by the assumption of the theorem, there must be a path $e_0, v_1, e_1, v_2, e_2 \dots, e_n$ in the hypergraph $\langle V, E \setminus B\rangle$ such that $e_0 \in A$, and

$v \in e_n$. Consider path $\pi = e_0, v_1, e_1, v_2, e_2 \ldots, e_n, v, b_i$ in hypergraph H. By Theorem 10, $\hat{r}_\pi \in \mathcal{R}(\mathcal{P}_A)$. Note also that $\hat{r}_\pi(b_j, u) = \hat{r}(b_j, u)$ for all j and all $u \in b_j$ with the exception of $j = i$ and $u = v$. In the case that $j = i$ and $u = v$, we have $\hat{r}_\pi(b_j, u) = 1 - \hat{r}(b_j, u) = r_i(b_i, u)$. Pick $\hat{r}_\pi$ to be the new $\hat{r}$.

2. If $b_i \notin A$, then, by (6),

$$\sum_{v \in b_i} \hat{r}(b_i, v) = 0 = \sum_{v \in b_i} r_i(b_i, v) \mod 2.$$

At the same time, by our assumption, $\hat{r}(b_i, v) \neq r_i(b_i, v)$. Thus there must be $u \in b_i$ such that $u \neq v$ and $\hat{r}(b_i, u) \neq r_i(b_i, u)$. Note that vertices u and v could belong either to the same connected component or to two different connected components of hypergraph $\langle V, E \setminus B\rangle$. We will consider these two subcases separately.

   (a) Suppose u and v belong to the same connected component of hypergraph $\langle V, E \setminus B\rangle$. Thus, there must be a path π' in that hypergraph which connects an edge containing vertex u with an edge containing v. Consider now a cyclic path in hypergraph $H = \langle V, E\rangle$ that starts at edge b_i, via vertex u get on the path π', goes through the whole path π', and via vertex v gets back to b_i. Call this cyclic path π.

   (b) Suppose u and v belong to different connected components of hypergraph $\langle V, E \setminus B\rangle$. Thus, by the assumption of the theorem, hypergraph $\langle V, E \setminus B\rangle$ contains a path $\pi_u = a_u, \ldots, e_u$ that connects an edge $a_u \in A$ with an edge e_u containing end u. By the same assumption, hypergraph $\langle V, E \setminus B\rangle$ must also contain a path $\pi_v = e_v, \ldots, a_v$ that connects an edge e_v, containing end v, with an edge $a_v \in A$. Let $\pi = \pi_u, u, b_i, v, \pi_v$.

   By Theorem 10, $\hat{r}_\pi \in \mathcal{R}(\mathcal{P}_A)$. Note also that $\hat{r}_\pi(b_j, w) = \hat{r}(b_j, w)$ for all j and all $w \in b_j$ with the exception of $j = i$ and $w \in \{u, v\}$. In the case that $j = i$ and $w \in \{u, v\}$, we have $\hat{r}_\pi(b_j, w) = 1 - \hat{r}(b_j, w) = r_i(b_i, w)$. Pick $\hat{r}_\pi$ to be the new $\hat{r}$.

Let r be $\hat{r}$ with all the modifications described above. These modifications guarantee that $r(b_i) = \hat{r}(b_i, v) = r_i(b_i, v)$ for any $v \in b_i$ and any $i \leq k$. $\square$

7.2 Generalized Parity Protocol

In this section, we will generalize the parity protocol through a recursive construction. First, however, we will need to establish the following technical result.

Theorem 14 (protocol extension). *Let $H = \langle V, E\rangle$ be any hypergraph, X be a set of vertices in H and $H' = \langle V', E'\rangle$ be the result of the truncation of X from H. For any finite protocol $\mathcal{P}'$ on H', there is a finite protocol $\mathcal{P}$ on H such that $\mathcal{P} \vDash [Q]$ if and only if $\mathcal{P}' \vDash [Q \cap E']$, for any set $Q \subseteq E$.*

Proof. To define protocol $\mathcal{P}$, we need to specify a set of values $Val(c)$ for each edge $c \in E$ and the set of local conditions Loc_v for each vertex v in hypergraph H. If $c \in E'$, then let $Val(c)$ be the same as in protocol $\mathcal{P}'$. Otherwise, $Val(c) = \{\epsilon\}$, where ϵ is an arbitrary element. The local conditions for vertices in $V \setminus X$ are the same as in protocol $\mathcal{P}'$, and the local conditions for vertices not in X are equal to the boolean constant *True*. This completes the definition of $\mathcal{P}$. Clearly, $\mathcal{P}$ has at least one run r_0 since protocol $\mathcal{P}'$ has a run.

$(\Rightarrow)$: Suppose that $Q \cap E' = \{q_1, \dots, q_k\}$. Consider any $r'_1, \dots, r'_k \in \mathcal{R}(\mathcal{P}')$. Define runs $r_1, \dots, r_k$ as follows, for any $c \in E$:

$$r_i(c) = \begin{cases} r'_i(c) & \text{if } c \in E', \\ \varepsilon & \text{if } c \notin E'. \end{cases}$$

Note that runs r_i and r'_i, by definition, are equal on any edge incident with any vertex in hypergraph H'. Thus, r_i satisfies the local conditions at any such vertex. Hence, $r_i \in \mathcal{R}(\mathcal{P})$ for any $i \in \{1, \dots, k\}$. Since $\mathcal{P} \vDash [Q]$, there is a run $r \in \mathcal{R}(\mathcal{P})$ such that

$$r_i(c) = \begin{cases} r_i(c) & \text{if } c \in Q \cap E', \\ r_0(c) & \text{if } c \in Q \setminus E'. \end{cases}$$

Define r' to be a restriction of r on hypergraph H'. Note that r' satisfies all local conditions of $\mathcal{P}'$. Thus, $r' \in \mathcal{R}(\mathcal{P}')$. At the same time, $r'(q_i) = r_i(q_i) = r'_i(q_i)$ for each $q_i \in Q \cap E'$.

$(\Leftarrow)$: Suppose that $Q = \{q_1, \dots, q_k\}$. Consider any $r_1, \dots, r_k \in \mathcal{R}(\mathcal{P})$, and let $r'_1, \dots, r'_k$ be their respective restrictions to hypergraph H'. Since, for any $i \in \{1, \dots, k\}$, run r'_i satisfies the local conditions of $\mathcal{P}'$ at any node of hypergraph H', we can conclude that $r'_1, \dots, r'_k \in \mathcal{R}(\mathcal{P}')$. By the assumption that $\mathcal{P}' \vDash [Q \cap E']$, there is a run $r' \in \mathcal{R}(\mathcal{P}')$ such that $r'(q) = r'_i(q)$ for any $q \in Q \cap E'$. In addition, $r'(q) = \varepsilon = r'_i(q)$ for any $q \in Q \backslash E'$. Hence, $r'(q_i) = r'_i(q_i)$ for any $i \in \{1, \dots, k\}$. Define run r as follows:

$$r(c) = \begin{cases} r'(c) & \text{if } c \in E', \\ \varepsilon & \text{if } c \notin E'. \end{cases}$$

Note that r satisfies the local conditions of $\mathcal{P}$ at all nodes. Thus, $r \in \mathcal{R}(\mathcal{P})$. In addition, $r(q_i) = r'(q_i) = r'_i(q_i)$ for all $q_i \in Q$. □

We will now prove the key theorem in our construction. The proof of this theorem recursively defines a generalization of the parity protocol.

Theorem 15. *For any hypergraph $H = \langle V, E \rangle$ and any sets $A, B_1, \dots, B_n \subseteq E$, if $H \nvdash \bigwedge_{1 \le i \le n} [B_i] \to [A]$, then there is a finite protocol $\mathcal{P}$ over H such that $\mathcal{P} \nvDash [A]$ and $\mathcal{P} \vDash [B_i]$ for all $i \le n$.*

Proof. Induction on the size of V.

Case 1. If $|A| \le 1$, then, by the Small Set axiom, $H \vdash [A]$. Hence, $H \vdash \bigwedge_{1 \le i \le n} [B_i] \to [A]$, which is a contradiction.

Case 2. Suppose that the edges of hypergraph H can be divided into two non-trivial disconnected sets X and Y. Thus, the empty set is a gateway between $A \cap X$ and $A \cap Y$. By the Gateway axiom,

$$H \vdash [A \cap X] \to ([A \cap Y] \to [A]).$$

Thus, taking into account the assumption $H \nvdash \bigwedge_{1\le i\le n}[B_i] \to [A]$, either

$$H \nvdash \bigwedge_{1\le i\le n}[B_i] \to [A\cap X]$$

or

$$H \nvdash \bigwedge_{1\le i\le n}[B_i] \to [A\cap Y].$$

Without loss of generality, we will assume the former. By Theorem 1,

$$H \nvdash \bigwedge_{1\le i\le n}[B_i \cap X] \to [A\cap X].$$

By the Small Set axiom,

$$H \nvdash [\varnothing] \to (\bigwedge_{1\le i\le n}[B_i \cap X] \to [A\cap X]).$$

Consider the set V_Y of all vertices in component Y. Let H' be the result of the truncation of graph H that removes V_Y from H. Note that $Out(V_Y) = \varnothing$, since sets X and Y are disconnected. Thus, by the Truncation rule,

$$H' \nvdash \bigwedge_{1\le i\le n}[B_i \cap X] \to [A\cap X].$$

By the Induction Hypothesis, there is a protocol $\mathcal{P}'$ on H' such that $\mathcal{P}' \nvDash [A\cap X]$ and $\mathcal{P}' \vDash [B_i \cap X]$, for any $i \le n$. Therefore, by Theorem 14, there is a protocol $\mathcal{P}$ on H such that $\mathcal{P} \nvDash [A]$ and $\mathcal{P} \vDash [B_i]$ for any $i \le n$.

Case 3. Suppose there is $i_0 \in \{1,\dots,n\}$ such that at least one connected component of hypergraph $\langle V, E\setminus B_{i_0}\rangle$ does not contain an element of A. We will call this connected component Y. Let V_Y be the set of all vertices in this component. Note that $Out(V_Y)$ is a gateway between $In(V_Y)$ and the complement of $In(V_Y)$. Hence, $Out(V_Y)$ is also a gateway between $A\cap In(V_Y)$ and $A\setminus In(V_Y)$. Therefore, by the Gateway axiom, taking into account that $In(V_Y)\cap Out(V_Y) = \varnothing$,

$$H \vdash [A\cap In(V_Y), Out(V_Y)] \to ([A\setminus In(V_Y))] \to [A]). \tag{8}$$

Recall now that by the assumption of this case, component Y of graph $\langle V, E\setminus B_{i_0}\rangle$ does not contain any elements of A. Hence, $A\cap In(V_Y) \subseteq B_{i_0}$. At the same time, $Out(V_Y)\subseteq B_{i_0}$ by the definition of set V_Y. Thus, from statement (8) and Theorem 1,

$$H \vdash [B_{i_0}] \to ([A\setminus In(V_Y))] \to [A]). \tag{9}$$

By the assumption of the theorem,

$$H \nvdash \bigwedge_{1\le i\le n}[B_i] \to [A]. \tag{10}$$

From statements (9) and (10),

$$H \nvdash \bigwedge_{1\le i\le n}[B_i] \to [A \setminus In(V_Y)].$$

By the laws of propositional logic,

$$H \nvdash [B_{i_0}] \to (\bigwedge_{1\le i\le n}[B_i] \to [A \setminus In(V_Y)]).$$

Since $Out(V_Y) \subseteq B_{i_0}$, by Theorem 1,

$$H \nvdash [Out(V_Y)] \to (\bigwedge_{1\le i\le n}[B_i] \to [A \setminus In(V_Y)]).$$

Again by Theorem 1,

$$H \nvdash [Out(V_Y)] \to (\bigwedge_{1\le i\le n}[B_i \setminus In(V_Y)] \to [A \setminus In(V_Y)]).$$

Let H' be the result of the truncation of set V_Y from hypergraph H. By the Truncation rule,

$$H' \nvdash \bigwedge_{1\le i\le n}[B_i \setminus In(V_Y)] \to [A \setminus In(V_Y)].$$

By the Induction Hypothesis, there is a protocol $\mathcal{P}'$ on H' such that $\mathcal{P}' \nvDash [A \setminus In(V_Y)]$ and $\mathcal{P}' \vDash [B_i \setminus In(V_Y)]$ for any $i \le n$. Therefore, by Theorem 14, there is a protocol $\mathcal{P}$ on H such that $\mathcal{P} \nvDash [A]$ and $\mathcal{P} \vDash [B_i]$ for any $i \le n$.

Case 4. Assume now that (i) $|A| > 1$, (ii) hypergraph H is connected, and (iii) for any $i \in \{1,\dots,n\}$, each connected component of hypergraph $\langle V, E \setminus B_{i_0}\rangle$ contains at least one element of A. Consider the parity protocol $\mathcal{P}_A$ over H. By Theorem 12, $\mathcal{P}_A \nvDash [A]$. By Theorem 13, $\mathcal{P}_A \vDash [B_i]$ for any $i \in \{1,\dots,n\}$. □

7.3 Completeness: Final Steps

Theorem 16. *For any $n \ge 0$ and any finite protocols $\mathcal{P}_1,\dots,\mathcal{P}_n$ over a hypergraph H there is a finite protocol $\mathcal{P}$ over H such that for any set of edges Q of this hypergraph, $\mathcal{P} \vDash [Q]$ if and only if $\mathcal{P}_i \vDash [Q]$ for any $i \le n$.*

Proof. First, consider the case where $n = 0$. Pick any symbol ϵ and define $\mathcal{P}$ to be $\langle Val, Loc\rangle$ such that $Val(c) = \{\epsilon\}$ for any $c \in E$, and local condition Loc_v to be the constant $True$ at every vertex v. By Definition 9, $\mathcal{P} \vDash [C]$ for any $C \subseteq E$.

We will now assume that $n > 0$ and define the composition of protocols $\mathcal{P}_1,\dots,\mathcal{P}_n$. Informally, composition is the result of several protocols run over the same hypergraph without any interaction between the protocols. Formally, suppose that $\mathcal{P}_1 = \langle Val^1, Loc^1\rangle,\dots,\mathcal{P}_n = \langle Val^n, Loc^n\rangle$ and define protocol $\mathcal{P} = \langle Val, Loc\rangle$ as follows:

1. $Val(c) = Val^1(c) \times \cdots \times Val^n(c)$,
2. $Loc_v(\langle c_1^1, \ldots, c_1^n \rangle, \ldots, \langle c_k^1, \ldots, c_k^n \rangle) = \bigwedge_{1 \le i \le n} Loc_v^i(c_1^i, \ldots, c_k^i)$,

To show that $\mathcal{P}$ is a protocol, we need to show that it has at least one run. Let $r^1, \ldots, r^n$ be runs of $\mathcal{P}^1, \ldots, \mathcal{P}^n$. Define $r(c)$ to be $\langle r^1(c), \ldots, r^n(c) \rangle$. It is easy to see that r satisfies the local conditions Loc_v for any vertex v of the hypergraph H. Thus, $r \in \mathcal{R}(\mathcal{P})$.

We will use notation $\{r(c)\}_i$ to denote the ith component of the value of $r(c)$.

Lemma 4. *For any set of edges Q,*

$$\mathcal{P} \vDash [Q] \quad \textit{if and only if} \quad \forall i\ (\mathcal{P}_i \vDash [Q]).$$

Proof. Let $Q = \{q_1, \ldots, q_\ell\}$.

$(\Rightarrow)$: Assume $\mathcal{P} \vDash [Q]$ and pick any $i_0 \in \{1, \ldots, n\}$. We will show that $\mathcal{P}_{i_0} \vDash [Q]$. Pick any runs $r'_1, \ldots, r'_\ell \in \mathcal{R}(\mathcal{P}_{i_0})$. For each $i \in \{1, \ldots, i_0 - 1, i_0 + 1, \ldots, n\}$, select an arbitrary run $r^i \in \mathcal{R}(\mathcal{P}_i)$. We then define a series of composed runs r_j for $j \in \{1, \ldots, \ell\}$ by

$$r_j(c) = \langle r^1(c), \ldots, r^{i_0 - 1}(c), r'_j(c), r^{i_0 + 1}(c), \ldots, r^n(c) \rangle,$$

for each edge $c \in E$. Since the component parts of each r_j belong in their respective sets $\mathcal{R}(\mathcal{P}_i)$, the composed runs are themselves members of $\mathcal{R}(\mathcal{P})$. By our assumption, $\mathcal{P} \vDash [Q]$, thus there is $r \in \mathcal{R}(\mathcal{P})$ such that $r(q_i) = r_i(q_i)$ for any $i_0 \in \{1, \ldots, \ell\}$. Finally, we consider the run r^*, where $r^*(c) = \{r(c)\}_{i_0}$ for each $c \in E$. That is, we let the value of r^* on c be the i_o-th component of $r(c)$. By definition of composition, $r^* \in \mathcal{R}(\mathcal{P}_{i_0})$, and it matches the original $r'_1, \ldots, r'_\ell \in \mathcal{R}(\mathcal{P}_{i_0})$ on edges $q_1, \ldots, q_\ell$, respectively. Hence, we have shown that $\mathcal{P}_{i_0} \vDash [Q]$.

$(\Leftarrow)$: Assume $\forall i\ (\mathcal{P}_i \vDash [Q])$. We will show that $\mathcal{P} \vDash [Q]$. Pick any runs $r_1, \ldots, r_\ell \in \mathcal{R}(\mathcal{P})$. For each $i \in \{1, \ldots, n\}$, each $j \in \{1, \ldots, \ell\}$, and each edge c, let $r_j^i(c) = \{r_j(c)\}_i$. That is, for each c, define a run r_j^i whose value on edge c equals the ith component of $r_j(c)$. Note that by the definition of composition, for each i and each j, r_j^i is a run in $\mathcal{R}(\mathcal{P}_i)$. Next, for each $i \in \{1, \ldots, n\}$, we use the fact that $\mathcal{P}_i \vDash [Q]$ to construct a run $r^i \in \mathcal{R}(\mathcal{P}_i)$ such that $r^i(q_j) = r_j^i(q_j)$. Finally, we compose these n runs $r^1, \ldots, r^n$ to get run $r \in \mathcal{R}(\mathcal{P})$. We note that the value of each edge q_j on r matches the the value of q_j in run $r_j \in \mathcal{R}(\mathcal{P})$, demonstrating that $\mathcal{P} \vDash [Q]$. □

This concludes the proof of Theorem 16. □

We are now ready to prove Theorem 8.

Proof. We give a proof by contradiction. Let X be a maximal consistent set of formulas from $\Phi(H)$ that contains $\neg\phi$. Let $\{A_1, \ldots, A_n\} = \{A \subseteq E \mid [A] \notin X\}$ and $\{B_1, \ldots, B_k\} = \{B \subseteq E \mid [B] \in X\}$. Thus, $H \nvdash \bigwedge_{1 \le j \le k} [B_j] \to [A_i]$, for any $i \le n$, due to the consistency of X. We will construct a protocol $\mathcal{P}$ such that $\mathcal{P} \nvDash [A_i]$ for any $i \le n$ and $\mathcal{P} \vDash [B_j]$ for any $j \le k$.

By Theorem 15, there are finite protocols $\mathcal{P}^1, \dots, \mathcal{P}^n$ such that $\mathcal{P}^i \nvDash [A_i]$ and $\mathcal{P}^i \vDash [B_j]$ for all $i \leq n$ and $j \leq k$. By Theorem 16, there is a protocol $\mathcal{P}$ such that $\mathcal{P} \nvDash [A_i]$ for any $i \leq n$ and $\mathcal{P} \vDash [B_j]$ for any $j \leq k$.

By induction on structural complexity of any formula $\psi \in \Phi(H)$, one can show now that $\mathcal{P} \vDash \psi$ if and only if $\psi \in X$. Thus, $\mathcal{P} \vDash \neg\phi$. Therefore, $\mathcal{P} \nvDash \phi$. □

Corollary 1. *The set* $\{(H, \phi) \mid H \vdash \phi\}$ *is decidable.*

Proof. The complement of this set is recursively enumerable due to the completeness of the system with respect to finite protocols. □

References

1. Armstrong, W.W.: Dependency structures of data base relationships. In: Information Processing, Proc. IFIP Congress, Stockholm, vol. 74, pp. 580–583. North-Holland, Amsterdam (1974)
2. Garcia-Molina, H., Ullman, J., Widom, J.: Database Systems: The Complete Book, 2nd edn. Prentice-Hall, Englewood Cliffs (2009)
3. Beeri, C., Fagin, R., Howard, J.H.: A complete axiomatization for functional and multivalued dependencies in database relations. In: SIGMOD 1977: Proceedings of the 1977 ACM SIGMOD International Conference on Management of Data, pp. 47–61. ACM, New York (1977)
4. Sutherland, D.: A model of information. In: Proceedings of Ninth National Computer Security Conference, pp. 175–183 (1986)
5. Halpern, J.Y., O'Neill, K.R.: Secrecy in multiagent systems. ACM Trans. Inf. Syst. Secur. 12(1), 1–47 (2008) (originally appeared as [12])
6. Miner More, S., Naumov, P.: An independence relation for sets of secrets. Studia Logica 94(1), 73–85 (2010) (originally appeared as [11])
7. Geiger, D., Paz, A., Pearl, J.: Axioms and algorithms for inferences involving probabilistic independence. Inform. and Comput. 91(1), 128–141 (1991)
8. Kelvey, R., Miner More, S., Naumov, P., Sapp, B.: Independence and functional dependence relations on secrets. In: Proceedings of 12th International Conference on the Principles of Knowledge Representation and Reasoning, Toronto, pp. 528–533. AAAI, Menlo Park (2010)
9. Miner More, S., Naumov, P.: On interdependence of secrets in collaboration networks. In: Proceedings of 12th Conference on Theoretical Aspects of Rationality and Knowledge, Stanford University, pp. 208–217 (2009)
10. Berge, C.: Hypergraphs. North-Holland Mathematical Library, vol. 45. North-Holland Publishing Co., Amsterdam (1989); Combinatorics of finite sets, Translated from the French Version
11. Miner More, S., Naumov, P.: An independence relation for sets of secrets. In: Ono, H., Kanazawa, M., de Queiroz, R. (eds.) WoLLIC 2009. LNCS (LNAI), vol. 5514, pp. 296–304. Springer, Heidelberg (2009)
12. Halpern, J.Y., O'Neill, K.R.: Secrecy in multiagent systems. In: Proceedings of the Fifteenth IEEE Computer Security Foundations Workshop, pp. 32–46 (2002)

Combining Logics in Simple Type Theory[⋆]

Christoph Benzmüller[⋆⋆]

Articulate Software, Angwin, CA, U.S.

Abstract. Simple type theory is suited as framework for combining classical and non-classical logics. This claim is based on the observation that various prominent logics, including (quantified) multimodal logics and intuitionistic logics, can be elegantly embedded in simple type theory. Furthermore, simple type theory is sufficiently expressive to model combinations of embedded logics and it has a well understood semantics. Off-the-shelf reasoning systems for simple type theory exist that can be uniformly employed for reasoning *within* and *about* combinations of logics. Combinations of modal logics and other logics are particularly relevant for multi-agent systems.

1 Introduction

Church's simple type theory $\mathcal{STT}$ [18], also known as classical higher-order logic, is suited as a framework for combining classical and non-classical logics. This is what this paper illustrates.

Evidently, $\mathcal{STT}$ has many prominent classical logic fragments, including propositional and first-order logic, the guarded fragment, second-order logic, monadic second-order logic, the basic fragment of $\mathcal{STT}$, etc. Interestingly, also prominent non-classical logics – including quantified multi-modal logics and intuitionistic logic – can be elegantly embedded in $\mathcal{STT}$. It is thus not surprising that also combinations of such logics can be flexibly modeled within $\mathcal{STT}$. Our claim is furthermore supported by the fact that the semantics of $\mathcal{STT}$ is well understood [1,2,8,26] and that powerful proof assistants and automated theorem provers for $\mathcal{STT}$ already exist. The automation of $\mathcal{STT}$ currently experiences a renaissance that has been fostered by the recent extension of the successful TPTP infrastructure for first-order logic [33] to higher-order logic, called TPTP THF [34,35,15]. Exploiting this new infrastructure we will demonstrate how higher-order automated theorem provers and model generators can be employed for reasoning *within* and *about* combinations of logics.

Our work is relevant for multi-agents systems in several ways. Most importantly, modal logics and combinations of modal logics are often employed for modeling multi-agents systems and for reasoning about them.

[⋆] A previous version of this paper has been presented at the World Congress and School on Universal Logic III (UNILOG'2010), Lisbon, Portugal, April 18-25, 2010.

[⋆⋆] This work has been funded by the German Research Foundation (DFG) under grant BE 2501/6-1.

In this paper we present a fresh approach to the automation of logic combinations and we in particular include quantified modal logics. For quantified modal logics actually only very few theorem provers are available. In our approach even challenge combinations of logics can be achieved: as an example we outline a combination of spatial and epistemic reasoning. Moreover, our approach even supports the automated analysis and verification of meta-properties of combined logics. It can thus serve as a useful tool for engineers of logic combinations.

In Sect. 2 we outline our embedding of quantified multimodal logics in $\mathcal{STT}$. Further logic embeddings in $\mathcal{STT}$ are discussed in Sect. 3; our examples comprise intuitionistic logic, access control logics and the region connection calculus. In Sect. 4 we illustrate how the reasoning *about* logics and their combinations is facilitated in our approach, and in Sect. 5 we employ simple examples to demonstrate the application of our approach for reasoning *within* combined logics. The performance results of our experiments with off-the-shelf, TPTP THF compliant higher-order automated reasoning systems are presented in Sect. 6.

2 (Normal) Quantified Multimodal Logics in $\mathcal{STT}$

$\mathcal{STT}$ [18] is based on the simply typed λ-calculus. The set $\mathcal{T}$ of simple types is usually freely generated from a set of basic types $\{o, \iota\}$ (where o is the type of Booleans and ι is the type of individuals) using the right-associative function type constructor $\rightarrow$. Instead of $\{o, \iota\}$ we here consider a set of base types $\{o, \iota, \mu\}$, providing an additional base type μ (the type of possible worlds).

The simple type theory language $\mathcal{STT}$ is defined by (where α, β, $o \in \mathcal{T}$):

$$\begin{aligned} s,t ::= p_\alpha \mid X_\alpha \mid (\lambda X_\alpha \mathbin{\scriptstyle\blacksquare} s_\beta)_{\alpha\rightarrow\beta} \mid (s_{\alpha\rightarrow\beta}\, t_\alpha)_\beta \mid (\neg_{o\rightarrow o}\, s_o)_o \mid \\ (s_o \vee_{o\rightarrow o\rightarrow o} t_o)_o \mid (s_\alpha =_{\alpha\rightarrow\alpha\rightarrow o} t_\alpha)_o \mid (\Pi_{(\alpha\rightarrow o)\rightarrow o}\, s_{\alpha\rightarrow o})_o \end{aligned}$$

p_α denotes typed constants and X_α typed variables (distinct from p_α). Complex typed terms are constructed via abstraction and application. Our logical connectives of choice are $\neg_{o\rightarrow o}$, $\vee_{o\rightarrow o\rightarrow o}$, $=_{\alpha\rightarrow\alpha\rightarrow o}$ and $\Pi_{(\alpha\rightarrow o)\rightarrow o}$ (for each type α).[1] From these connectives, other logical connectives can be defined in the usual way (e.g., $\wedge$ and $\Rightarrow$). We often use binder notation $\forall X_\alpha \mathbin{\scriptstyle\blacksquare} s$ for $\Pi_{(\alpha\rightarrow o)\rightarrow o}(\lambda X_\alpha \mathbin{\scriptstyle\blacksquare} s_o)$. We assume familiarity with α-conversion, β- and η-reduction, and the existence of β- and $\beta\eta$-normal forms. Moreover, we obey the usual definitions of free variable occurrences and substitutions.

The semantics of $\mathcal{STT}$ is well understood and thoroughly documented in the literature [1,2,8,26]. The semantics of choice for our work is Henkin semantics.

Quantified modal logics have been studied by Fitting [19] (further related work is available by Blackburn and Marx [16] and Braüner [17]). In contrast to Fitting we are here not interested only in **S5** structures but in the more general case of **K** from which more constrained structures (such as **S5**) can be easily obtained. First-order quantification can be constant domain or varying domain.

[1] This choice is not minimal (from $=_{\alpha\rightarrow\alpha\rightarrow o}$ all other logical constants can already be defined [3]). It useful though in the context of resolution based theorem proving.

Below we only consider the constant domain case: every possible world has the same domain. Like Fitting, we keep our definitions simple by not having function or constant symbols. While Fitting [19] studies quantified monomodal logic, we are interested in quantified multimodal logic. Hence, we introduce multiple $\Box_r$ operators for symbols r from an index set S. The grammar for our quantified multimodal logic $\mathcal{QML}$ hence is

$$s, t ::= P \mid k(X^1, \ldots, X^n) \mid \neg\, s \mid s \,\vee\, t \mid \forall X_\centerdot s \mid \forall P_\centerdot s \mid \Box_r\, s$$

where $P \in \mathcal{PV}$ denotes propositional variables, $X, X^i \in \mathcal{IV}$ denote first-order (individual) variables, and $k \in \mathcal{SYM}$ denotes predicate symbols of any arity. Further connectives, quantifiers, and modal operators can be defined as usual. We also obey the usual definitions of free variable occurrences and substitutions.

Fitting introduces three different notions of Kripke semantics for $\mathcal{QML}$: $\mathbf{QS5}\pi^-$, $\mathbf{QS5}\pi$, and $\mathbf{QS5}\pi^+$. In our work [10] we study related notions $\mathbf{QK}\pi^-$, $\mathbf{QK}\pi$, and $\mathbf{QK}\pi^+$ for a modal context $\mathbf{K}$, and we support multiple modalities.

$\mathcal{STT}$ is an expressive logic and it is thus not surprising that $\mathcal{QML}$ can be elegantly modeled and even automated as a fragment of $\mathcal{STT}$. The idea of the encoding, called $\mathcal{QML}^{STT}$, is simple. Choose type ι to denote the (non-empty) set of individuals and we choose the second base type μ to denote the (non-empty) set of possible worlds. As usual, the type o denotes the set of truth values. Certain formulas of type $\mu \to o$ then correspond to multimodal logic expressions. The multimodal connectives $\neg$, $\vee$, and $\Box$, become λ-terms of types $(\mu \to o) \to (\mu \to o)$, $(\mu \to o) \to (\mu \to o) \to (\mu \to o)$, and $(\mu \to \mu \to o) \to (\mu \to o) \to (\mu \to o)$ respectively.

Quantification is handled as in $\mathcal{STT}$ by modeling $\forall X_\centerdot p$ as $\Pi(\lambda X_\centerdot .p)$ for a suitably chosen connective Π. Here we are interested in defining two particular modal $\boldsymbol{\Pi}$-connectives: $\boldsymbol{\Pi}^\iota$, for quantification over individual variables, and $\boldsymbol{\Pi}^{\mu \to o}$, for quantification over modal propositional variables that depend on worlds. They become terms of type $(\iota \to (\mu \to o)) \to (\mu \to o)$ and $((\mu \to o) \to (\mu \to o)) \to (\mu \to o)$ respectively.

The $\mathcal{QML}^{STT}$ modal operators $\neg, \vee, \Box, \boldsymbol{\Pi}^\iota$, and $\boldsymbol{\Pi}^{\mu \to o}$ are now simply defined as follows:

$$\begin{aligned}
\neg_{\,(\mu\to o)\to(\mu\to o)} &= \lambda\phi_{\mu\to o\centerdot}\lambda W_{\mu\centerdot}\neg\phi\, W \\
\vee_{\,(\mu\to o)\to(\mu\to o)\to(\mu\to o)} &= \lambda\phi_{\mu\to o\centerdot}\lambda\psi_{\mu\to o\centerdot}\lambda W_{\mu\centerdot}\phi\, W \vee \psi\, W \\
\Box_{\,(\mu\to\mu\to o)\to(\mu\to o)\to(\mu\to o)} &= \lambda R_{\mu\to\mu\to o\centerdot}\lambda\phi_{\mu\to o\centerdot}\lambda W_{\mu\centerdot}\forall V_{\mu\centerdot}\neg R\, W\, V \vee \phi\, V \\
\boldsymbol{\Pi}^\iota_{(\iota\to(\mu\to o))\to(\mu\to o)} &= \lambda\phi_{\iota\to(\mu\to o)\centerdot}\lambda W_{\mu\centerdot}\forall X_{\iota\centerdot}\phi\, X\, W \\
\boldsymbol{\Pi}^{\mu\to o}_{((\mu\to o)\to(\mu\to o))\to(\mu\to o)} &= \lambda\phi_{(\mu\to o)\to(\mu\to o)\centerdot}\lambda W_{\mu\centerdot}\forall P_{\mu\to o\centerdot}\phi\, P\, W
\end{aligned}$$

Note that our encoding actually only employs the second-order fragment of $\mathcal{STT}$ enhanced with lambda-abstraction.

Further operators can be introduced as usual, for example, $\top = \lambda W_{\mu\centerdot}\top$, $\bot = \neg\,\top$, $\wedge = \lambda\phi,\psi_\centerdot\neg\,(\neg\,\phi \,\vee\, \neg\,\psi)$, $\supset = \lambda\phi,\psi_\centerdot\neg\,\phi \,\vee\, \psi$, $\Leftrightarrow = \lambda\phi,\psi_\centerdot(\phi \,\supset\, \psi) \wedge (\psi \,\supset\, \phi)$, $\Diamond = \lambda R,\phi_\centerdot\neg\,(\Box\, R\,(\neg\,\phi))$, $\boldsymbol{\Sigma}^\iota = \lambda\phi_\centerdot\neg\,\boldsymbol{\Pi}^\iota(\lambda X_\centerdot\neg\,\phi\, X)$, $\boldsymbol{\Sigma}^{\mu\to o} = \lambda\phi_\centerdot\neg\,\boldsymbol{\Pi}^{\mu\to o}(\lambda P_\centerdot\neg\,\phi\, P)$.

For defining $\mathcal{QML}^{STT}$-propositions we fix a set $\mathcal{IV}^{STT}$ of individual variables of type ι, a set $\mathcal{PV}^{STT}$ of propositional variables[2] of type $\mu \rightarrow o$, and a set $\mathcal{SYM}^{STT}$ of n-ary (curried) predicate symbols of types $\underbrace{\iota \rightarrow \ldots \rightarrow \iota}_{n} \rightarrow (\mu \rightarrow o)$. Moreover, we fix a set $\mathcal{S}^{STT}$ of accessibility relation constants of type $\mu \rightarrow \mu \rightarrow o$. $\mathcal{QML}^{STT}$-propositions are now defined as the smallest set of $\mathcal{STT}$-terms for which the following hold:

– if $P \in \mathcal{PV}^{STT}$, then $P \in \mathcal{QML}^{STT}$
– if $X^j \in \mathcal{IV}^{STT}$ $(j = 1, \ldots, n)$ and $k \in \mathcal{SYM}^{STT}$, then $(k\, X^1 \ldots X^n) \in \mathcal{QML}^{STT}$
– if $\phi, \psi \in \mathcal{QML}^{STT}$, then $\neg\, \phi \in \mathcal{QML}^{STT}$ and $\phi \vee \psi \in \mathcal{QML}^{STT}$
– if $r \in \mathcal{S}^{STT}$ and $\phi \in \mathcal{QML}^{STT}$, then $\Box\, r\, \phi \in \mathcal{QML}^{STT}$
– if $X \in \mathcal{IV}^{STT}$ and $\phi \in \mathcal{QML}^{STT}$, then $\boldsymbol{\Pi}^{\iota}(\lambda X_{\blacksquare} \phi) \in \mathcal{QML}^{STT}$
– if $P \in \mathcal{PV}^{STT}$ and $\phi \in \mathcal{QML}^{STT}$, then $\boldsymbol{\Pi}^{\mu\rightarrow o}(\lambda P_{\blacksquare} \phi) \in \mathcal{QML}^{STT}$

We write $\Box_r\, \phi$ for $\Box\, r\, \phi$, $\forall X_{\iota \blacksquare} \phi$ for $\boldsymbol{\Pi}^{\iota}(\lambda X_{\iota \blacksquare} \phi)$, and $\forall P_{\mu\rightarrow o \blacksquare} \phi$ for $\boldsymbol{\Pi}^{\mu\rightarrow o}(\lambda P_{\mu\rightarrow o \blacksquare} \phi)$.

Note that the defining equations for our $\mathcal{QML}$ modal operators are themselves formulas in $\mathcal{STT}$. Hence, we can express $\mathcal{QML}$ formulas in a higher-order reasoner elegantly in the usual syntax. For example, $\Box_r\, \exists P_{\mu\rightarrow o \blacksquare}\, P$ is a $\mathcal{QML}^{STT}$ proposition; it has type $\mu \rightarrow o$.

Validity of $\mathcal{QML}^{STT}$ propositions is defined in the obvious way: a $\mathcal{QML}$-proposition $\phi_{\mu\rightarrow o}$ is valid if and only if for all possible worlds w_μ we have $w \in \phi_{\mu\rightarrow o}$, that is, if and only if $\phi_{\mu\rightarrow o}\, w_\mu$ holds. Hence, the notion of validity is modeled via the following equation (alternatively we could define valid simply as $\Pi_{(\mu\rightarrow o)\rightarrow o}$):

$$\text{valid} = \lambda \phi_{\mu\rightarrow o \blacksquare} \forall W_{\mu \blacksquare}\, \phi\, W$$

Now we can formulate proof problems in $\mathcal{QML}^{STT}$, e.g., valid $\Box_r\, \exists P_{\mu\rightarrow o \blacksquare}\, P$. Using rewriting or definition expanding, we can reduce such proof problems to corresponding statements containing only the basic connectives $\neg$, $\vee$, $=$, Π^{ι}, and $\Pi^{\mu\rightarrow o}$ of $\mathcal{STT}$. In contrast to the many other approaches no external transformation mechanism is required. For our example formula valid $\Box_r\, \exists P_{\mu\rightarrow o \blacksquare}\, P$ unfolding and $\beta\eta$-reduction leads to $\forall W_{\mu \blacksquare} \forall Y_{\mu \blacksquare} \neg r\, W\, Y \vee (\neg \forall X_{\mu\rightarrow o \blacksquare} \neg (X\, Y))$. It is easy to check that this formula is valid in Henkin semantics: put $X = \lambda Y_{\mu \blacksquare} \top$.

We have proved soundness and completeness for this embedding [10], that is, for $s \in \mathcal{QML}$ and the corresponding $s_{\mu\rightarrow o} \in \mathcal{QML}^{STT} \subset \mathcal{STT}$ we have:

Theorem 1. $\models^{\mathcal{STT}}$ *(valid* $s_{\mu\rightarrow o}$*) if and only if* $\models^{\mathbf{QK}\pi}$ s*.*

This result also illustrates the correspondence between $\mathbf{QK}\pi$ models and Henkin models; for more details see [10].

Obviously, the reduction of our embedding to first-order multimodal logics (which only allow quantification over individual variables), to propositional quantified multimodal logics (which only allow quantification over propositional variables) and to propositional multimodal logics (no quantifiers) is sound and

[2] Note that the denotation of propositional variables depends on worlds.

complete. Extending our embedding for hybrid logics is straightforward [27]; note in particular that denomination of individual worlds using constant symbols of type μ is easily possible.

In the remainder we will often omit type information. It is sufficient to remember that worlds are of type μ, multimodal propositions of type $\mu \to o$, and accessibility relations of type $\mu \to \mu \to o$. Individuals are of type ι.

3 Embeddings of Other Logics in $\mathcal{STT}$

We have studied several other logic embeddings in $\mathcal{STT}$, some of which will be mentioned in this section.

Intuitionistic Logics. Gödels interpretation of propositional intuitionistic logic in propositional modal logic $S4$ [23] can be combined with our results from the previous section in order to provide a sound and complete embedding of propositional intuitionistic logic into $\mathcal{STT}$ [10].

Gödel studies the propositional intuitionistic logic $\mathcal{IPL}$ defined by

$$s, t ::= p \mid \dot{\neg}\, s \mid s \mathbin{\dot{\supset}} t \mid s \mathbin{\dot{\vee}} t \mid p \mathbin{\dot{\wedge}} t$$

He introduces the a mapping from $\mathcal{IPL}$ into propositional modal logic $S4$ which maps $\dot{\neg}\, s$ to $\neg \Box_r s$, $s \mathbin{\dot{\supset}} t$ to $\Box_r s \supset \Box_r t$, $s \mathbin{\dot{\vee}} t$ to $\Box_r s \vee \Box_r t$, and $s \mathbin{\dot{\wedge}} t$ to $s \wedge t$.[3] By simply combining Gödel's mapping with our mapping from before we obtain the following embedding of $\mathcal{IPL}$ in $\mathcal{STT}$.

Let $\mathcal{IPL}$ be a propositional intuitionistic logic with atomic primitives p^1, ..., p^m $(m \geq 1)$. We define the set $\mathcal{IPL}^{\mathcal{STT}}$ of corresponding propositional intuitionistic logic propositions in $\mathcal{STT}$ as follows.

1. For the atomic $\mathcal{IPL}$ primitives $p^1, \ldots, p^m$ we introduce corresponding $\mathcal{IPL}^{\mathcal{STT}}$ predicate constants $p^1_{\mu\to o}, \ldots, p^m_{\mu\to o}$. Moreover, we provide the single accessibility relation constant $r_{\mu\to\mu\to o}$.
2. Corresponding to Gödel's mapping we introduce the logical connectives of $\mathcal{IPL}^{\mathcal{STT}}$ as abbreviations for the following λ-terms (we omit the types here):

$$\begin{aligned}
\dot{\neg} &= \lambda\phi_{\bullet} \lambda W_{\bullet} \neg\forall V_{\bullet} \neg r\, W\, V \vee \phi\, V \\
\dot{\supset} &= \lambda\phi_{\bullet} \lambda\psi_{\bullet} \lambda W_{\bullet} \neg(\forall V_{\bullet} \neg r\, W\, V \vee \phi\, V) \vee (\forall V_{\bullet} \neg r\, W\, V \vee \psi\, V) \\
\dot{\vee} &= \lambda\phi_{\bullet} \lambda\psi_{\bullet} \lambda W_{\bullet} (\forall V_{\bullet} \neg r\, W\, V \vee \phi\, V) \vee (\forall V_{\bullet} \neg r\, W\, V \vee \psi\, V) \\
\dot{\wedge} &= \lambda\phi_{\bullet} \lambda\psi_{\bullet} \lambda W_{\bullet} \neg(\neg\phi\, W \vee \neg\psi\, W)
\end{aligned}$$

3. We define the set of $\mathcal{IPL}^{\mathcal{STT}}$-propositions as the smallest set of simply typed λ-terms for which the following hold:
   - $p^1_{\mu\to o}, \ldots, p^m_{\mu\to o}$ define the atomic $\mathcal{IPL}^{\mathcal{STT}}$-propositions.
   - If ϕ and ψ are $\mathcal{IPL}^{\mathcal{STT}}$-propositions, then so are $\dot{\neg}\, \phi$, $\phi \mathbin{\dot{\supset}} \psi$, $\phi \mathbin{\dot{\vee}} \psi$, and $\phi \mathbin{\dot{\wedge}} \psi$.

[3] Alternative mappings have been proposed and studied in the literature which we could employ here equally as well.

The notion of validity we adopt is the same as for $\mathcal{QML}^{\mathcal{STT}}$. However, since Gödel connects $\mathcal{IPL}$ with modal logic $S4$, we transform each proof problem $t \in \mathcal{IPL}$ into a corresponding proof problem t' in $\mathcal{STT}$ of the following form

$$t' := ((\text{valid}\,\forall\phi_{\mu\to o}\text{.}\,\Box_r\,\phi \supset \phi) \wedge (\text{valid}\,\forall\phi_{\mu\to o}\text{.}\,\Box_r\,\phi \supset \Box_r\,\Box_r\,\phi)) \Rightarrow (\text{valid}\,t_{\mu\to o})$$

where $t_{\mu\to o}$ is the $\mathcal{IPL}^{\mathcal{STT}}$ term for t according to our definition above. Alternatively we may translate t into $t'' := ((\text{reflexive}\,r) \wedge (\text{transitive}\,r)) \Rightarrow (\text{valid}\,t_{\mu\to o})$.

Combining soundness [23] and completeness [28] of Gödel's embedding with Theorem 1 we obtain the following soundness and completeness result: Let $t \in \mathcal{IPL}$ and let $t' \in \mathcal{STT}$ as constructed above. t is valid in propositional intuitionistic logic if and only if t' is valid in $\mathcal{STT}$.

Example problems in intuitionistic logic have been encoded in THF syntax [15] and added to the TPTP THF library[4] and are accessible under identifiers SYO058^4 – SYO074^4.

Access Control Logics. Garg and Abadi recently translated several prominent access control logics into modal logic S4 and proved these translations sound and complete [21]. We have combined this work with our above results in order to obtain a sound and complete embedding of these access control logics in $\mathcal{STT}$ and we have carried out experiments with the prover LEO-II [7]. Example problems have been added to the TPTP THF library and are accessible under identifiers SWV425^x – SWV436^x (for $x \in \{1, \ldots, 4\}$).

Logics for Spatial Reasoning. Evidently, the region connection calculus [30] is a fragment of $\mathcal{STT}$: choose a base type r ('region') and a reflexive and symmetric relation c ('connected') of type $r \to r \to o$ and define (where X, Y, and Z are variables of type r):

$$\begin{aligned}
\text{disconnected}: \; & dc && = \lambda X, Y\text{.}\,\neg(c\;X\;Y)\\
\text{part of}: \; & p && = \lambda X, Y\text{.}\,\forall Z\text{.}\,((c\;Z\;X) \Rightarrow (c\;Z\;Y))\\
\text{identical with}: \; & eq && = \lambda X, Y\text{.}\,((p\;X\;Y) \wedge (p\;Y\;X))\\
\text{overlaps}: \; & o && = \lambda X, Y\text{.}\,\exists Z\text{.}\,((p\;Z\;X) \wedge (p\;Z\;Y))\\
\text{partially overlaps}: \; & po && = \lambda X, Y\text{.}\,((o\;X\;Y) \wedge \neg(p\;X\;Y) \wedge \neg(p\;Y\;X))\\
\text{externally connected}: \; & ec && = \lambda X, Y\text{.}\,((c\;X\;Y) \wedge \neg(o\;X\;Y))\\
\text{proper part}: \; & pp && = \lambda X, Y\text{.}\,((p\;X\;Y) \wedge \neg(p\;Y\;X))\\
\text{tangential proper part}: \; & tpp && = \lambda X, Y\text{.}\,((pp\;X\;Y) \wedge \exists Z\text{.}\,((ec\;Z\;X) \wedge (ec\;Z\;Y)))\\
\text{nontang. proper part}: \; & ntpp && = \lambda X, Y\text{.}\,((pp\;X\;Y) \wedge \neg\exists Z\text{.}\,((ec\;Z\;X) \wedge (ec\;Z\;Y)))
\end{aligned}$$

An example problem for the region connection calculus will be discussed below.

[4] TPTP THF problems for various problem categories are available at http://www.cs.miami.edu/~tptp/cgi-bin/SeeTPTP?Category=Problems; all problem identifiers with an '^' in their name refer to higher-order THF problems. The TPTP library meanwhile contains more than 2700 example problems in THF syntax.

4 Reasoning about Logics and Combinations of Logics

We illustrate how our approach supports reasoning about logics and their combinations. First, we focus on modal logics and their well known relationships between properties of accessibility relations and corresponding modal axioms (respectively axiom schemata) [25]. Such meta-theoretic insights can be elegantly encoded (and, as we will later see, automatically proved) in our approach. First we encode various accessibility relation properties in $\mathcal{STT}$:

$$\begin{aligned}
\text{reflexive} &= \lambda R_\centerdot \forall S_\centerdot R\,S\,S && (1)\\
\text{symmetric} &= \lambda R_\centerdot \forall S, T_\centerdot ((R\,S\,T) \Rightarrow (R\,T\,S)) && (2)\\
\text{serial} &= \lambda R_\centerdot \forall S_\centerdot \exists T_\centerdot (R\,S\,T) && (3)\\
\text{transitive} &= \lambda R_\centerdot \forall S, T, U_\centerdot ((R\,S\,T) \wedge (R\,T\,U) \Rightarrow (R\,S\,U)) && (4)\\
\text{euclidean} &= \lambda R_\centerdot \forall S, T, U_\centerdot ((R\,S\,T) \wedge (R\,S\,U) \Rightarrow (R\,T\,U)) && (5)\\
\text{partially\_functional} &= \lambda R_\centerdot \forall S, T, U_\centerdot ((R\,S\,T) \wedge (R\,S\,U) \Rightarrow T = U) && (6)\\
\text{functional} &= \lambda R_\centerdot \forall S_\centerdot \exists T_\centerdot ((R\,S\,T) \wedge \forall U_\centerdot ((R\,S\,U) \Rightarrow T = U)) && (7)\\
\text{weakly\_dense} &= \lambda R_\centerdot \forall S, T_\centerdot ((R\,S\,T) \Rightarrow \exists U_\centerdot ((R\,S\,U) \wedge (R\,U\,T))) && (8)\\
\text{weakly\_connected} &= \lambda R_\centerdot \forall S, T, U_\centerdot (((R\,S\,T) \wedge (R\,S\,U)) \Rightarrow \\
&\qquad ((R\,T\,U) \vee T = U \vee (R\,U\,T))) && (9)\\
\text{weakly\_directed} &= \lambda R_\centerdot \forall S, T, U_\centerdot (((R\,S\,T) \wedge (R\,S\,U)) \Rightarrow \\
&\qquad \exists V_\centerdot ((R\,T\,V) \wedge (R\,U\,V))) && (10)
\end{aligned}$$

Remember, that R is of type $\mu \rightarrow \mu \rightarrow o$ and S, T, U are of type μ. The corresponding axioms are given next.

$$\begin{aligned}
M &: \forall \phi_\centerdot \Box_r\, \phi \supset \phi && (11)\\
B &: \forall \phi_\centerdot \phi \supset \Box_r \Diamond_r\, \phi && (12)\\
D &: \forall \phi_\centerdot \Box_r\, \phi \supset \Diamond_r\, \phi && (13)\\
4 &: \forall \phi_\centerdot \Box_r\, \phi \supset \Box_r \Box_r\, \phi && (14)\\
5 &: \forall \phi_\centerdot \Diamond_r\, \phi \supset \Box_r \Diamond_r\, \phi && (15)
\end{aligned}$$

$$\begin{aligned}
&\forall \phi_\centerdot \Diamond_r\, \phi \supset \Box_r\, \phi && (16)\\
&\forall \phi_\centerdot \Diamond_r\, \phi \Leftrightarrow \Box_r\, \phi && (17)\\
&\forall \phi_\centerdot \Box_r \Box_r\, \phi \supset \Box_r\, \phi && (18)\\
&\forall \phi, \psi_\centerdot \Box_r\, ((\phi \wedge \Box_r\, \phi) \supset \psi) \vee \\
&\qquad \Box_r\, ((\psi \wedge \Box_r\, \psi) \supset \phi) && (19)\\
&\forall \phi_\centerdot \Diamond_r \Box_r\, \phi \supset \Box_r \Diamond_r\, \phi && (20)
\end{aligned}$$

Example 1. For k ($k =$ (1), ..., (10)) we can now easily formulate the well known correspondence theorems $(k) \Rightarrow (k+10)$ and $(k) \Leftarrow (k+10)$. For example,

$$(1) \Rightarrow (11) : \quad \forall R_\centerdot (\text{reflexive } R) \Rightarrow (\text{valid } \forall \phi_\centerdot \Box_R\, \phi \supset \phi)$$

Example 2. There are well known relationships between different modal logics and there exist alternatives for their axiomatization (cf. the relationship map in [22]). For example, for modal logic S5 we may choose axioms M and 5 as standard axioms. Respectively for logic KB5 we may choose B and 5. We may then want to investigate the following conjectures (the only one that does not hold is (31)):

$$\begin{aligned} \text{S5} = \text{M5} &\Leftrightarrow \text{MB5} && (21)\\ &\Leftrightarrow \text{M4B5} && (22)\\ &\Leftrightarrow \text{M45} && (23)\\ &\Leftrightarrow \text{M4B} && (24)\\ &\Leftrightarrow \text{D4B} && (25)\\ &\Leftrightarrow \text{D4B5} && (26)\\ &\Leftrightarrow \text{DB5} && (27) \end{aligned}$$

$$\begin{aligned} \text{KB5} &\Leftrightarrow \text{K4B5} && (28)\\ &\Leftrightarrow \text{K4B} && (29) \end{aligned}$$

$$\begin{aligned} \text{M5} &\Rightarrow \text{D45} && (30)\\ \text{D45} &\Rightarrow \text{M5} && (31) \end{aligned}$$

Exploiting the correlations $(k) \Leftrightarrow (k+10)$ from before these problems can be formulated as follows; we give the case for M5 $\Leftrightarrow$ D4B:

$$\forall R_\blacksquare (((\text{reflexive } R) \wedge (\text{euclidean } R)) \Leftrightarrow ((\text{serial } R) \wedge (\text{transitive } R) \wedge (\text{symmetric } R)))$$

Example 3. We can also encode the Barcan formula and its converse. (They are theorems in our approach, which confirms that we are 'constant domain'.)

$$\begin{aligned} BF: &\quad \text{valid } \forall X_{\iota}{}_\blacksquare\, \Box_r\, (p_{\iota\to(\mu\to o)}\, X) \supset \Box_r\, \forall X_{\iota}{}_\blacksquare\, (p_{\iota\to(\mu\to o)}\, X) && (32)\\ BF^{-1}: &\quad \text{valid } \Box_r\, \forall X_{\iota}{}_\blacksquare\, (p_{\iota\to(\mu\to o)}\, X) \supset \forall X_{\iota}{}_\blacksquare\, \Box_r\, (p_{\iota\to(\mu\to o)}\, X) && (33) \end{aligned}$$

Example 4. An interesting meta property for combined logics with modalities $\Diamond_i, \Box_j, \Box_k$, and $\Diamond_l$ is the correspondence between the following axiom and the (i,j,k,l)-confluence property

$$\begin{aligned} &(\text{valid } \forall \phi_\blacksquare\, (\Diamond_i\, \Box_j\, \phi) \supset \Box_k\, \Diamond_l\, \phi)\\ \Leftrightarrow\ &(\forall A_\blacksquare \forall B_\blacksquare \forall C_\blacksquare (((i\, A\, B) \wedge (k\, A\, C)) \Rightarrow \exists D_\blacksquare ((j\, B\, D) \wedge (l\, C\, D)))) \qquad (34) \end{aligned}$$

Example 5. Segerberg [31] discusses a 2-dimensional logic providing two S5 modalities $\Box_a$ and $\Box_b$. He adds further axioms stating that these modalities are commutative and orthogonal. It actually turns out that orthogonality is already implied in this context. This statement can be encoded in our framework as follows:

$$\begin{aligned} &(\text{reflexive } a), (\text{transitive } a), (\text{euclid. } a), (\text{reflexive } b), (\text{transitive } b), (\text{euclid. } b),\\ &(valid\, \forall \phi_\blacksquare\, \Box_a\, \Box_b\, \phi \Leftrightarrow \Box_b\, \Box_a\, \phi)\\ &\models^{\mathcal{STT}}\ (valid\, \forall \phi, \psi_\blacksquare\, \Box_a\, (\Box_a\, \phi \vee \Box_b\, \psi) \supset (\Box_a\, \phi \vee \Box_a\, \psi)) \wedge\\ &\qquad\quad (valid\, \forall \phi, \psi_\blacksquare\, \Box_b\, (\Box_a\, \phi \vee \Box_b\, \psi) \supset (\Box_b\, \phi \vee \Box_b\, \psi)) \qquad (35) \end{aligned}$$

Example 6. Suppose we want to work with a 2-dimensional logic combining a modality $\Box_k$ of knowledge with a modality $\Box_b$ of belief. Moreover, suppose we model $\Box_k$ as an S5 modality and $\Box_b$ as an D45 modality and let us furthermore add two axioms characterizing their relationship. We may then want to check whether or not $\Box_k$ and $\Box_b$ coincide, i.e., whether $\Box_k$ includes $\Box_b$:

$$\begin{aligned} &(\text{reflexive } k), (\text{transitive } k), (\text{euclid. } k), (\text{serial } b), (\text{transitive } b), (\text{euclid. } b),\\ &(valid\, \forall \phi_\blacksquare\, \Box_k\, \phi \supset \Box_b\, \phi), (valid\, \forall \phi_\blacksquare\, \Box_b\, \phi \supset \Box_b\, \Box_k\, \phi)\\ &\models^{\mathcal{STT}}\ (valid\, \forall \phi_\blacksquare\, \Box_b\, \phi \supset \Box_k\, \phi) \qquad (36) \end{aligned}$$

5 Reasoning within Combined Logics

We illustrate how our approach supports reasoning within combined logics. First we present two examples in epistemic reasoning. In this examples we model the individual and common knowledge of different persons respectively agents by combining different modalities. Our formulation in both cases adapts Baldoni's modeling [6].

Example 7 (Epistemic reasoning: The friends puzzle). (i) Peter is a friend of John, so if Peter knows that John knows something then John knows that Peter knows the same thing. (ii) Peter is married, so if Peter's wife knows something, then Peter knows the same thing. John and Peter have an appointment, let us consider the following situation: (a) Peter knows the time of their appointment. (b) Peter also knows that John knows the place of their appointment. Moreover, (c) Peter's wife knows that if Peter knows the time of their appointment, then John knows that too (since John and Peter are friends). Finally, (d) Peter knows that if John knows the place and the time of their appointment, then John knows that he has an appointment. From this situation we want to prove (e) that each of the two friends knows that the other one knows that he has an appointment.

For modeling the knowledge of Peter, Peter's wife, and John we consider a 3-dimensional logic combining the modalities $\Box_{\mathrm{p}}$, $\Box_{(\mathrm{w\,p})}$, and $\Box_{\mathrm{j}}$. Actually modeling them as S4 modalities turns out to be sufficient for this example. Hence, we introduce three corresponding accessibility relations j, p, and (w p). The S4 axioms for $x \in \{\mathrm{j}, \mathrm{p}, (\mathrm{w\,p})\}$ are

$$\text{valid}\ \forall\phi\text{.}\ \Box_x\,\phi \supset \phi \quad (37) \qquad\qquad \text{valid}\ \forall\phi\text{.}\ \Box_x\,\phi \supset \Box_x\,\Box_x\,\phi \quad (38)$$

As done before, we could alternatively postulate that the accessibility relations are reflexive and transitive.

Next, we encode the facts from the puzzle. For (i) we provide a persistence axiom and for (ii) an inclusion axiom:

$$\text{valid}\ \forall\phi\text{.}\ \Box_{\mathrm{p}}\,\Box_{\mathrm{j}}\,\phi \supset \Box_{\mathrm{j}}\,\Box_{\mathrm{p}}\,\phi \quad (39) \qquad\qquad \text{valid}\ \forall\phi\text{.}\ \Box_{(\mathrm{w\,p})}\,\phi \supset \Box_{\mathrm{p}}\,\phi \quad (40)$$

Finally, the facts (a)-(d) and the conclusion (e) are encoded as follows (time, place, and appointment are propositional constants, that is, constants of type $\mu \to o$ in our framework):

$$\text{valid}\ \Box_{\mathrm{p}}\,\text{time} \qquad (41)$$

$$\text{valid}\ \Box_{\mathrm{p}}\,\Box_{\mathrm{j}}\,\text{place} \qquad (42)$$

$$\text{valid}\ \Box_{(\mathrm{w\,p})}\,(\Box_{\mathrm{p}}\,\text{time} \supset \Box_{\mathrm{j}}\,\text{time}) \qquad (43)$$

$$\text{valid}\ \Box_{\mathrm{p}}\,\Box_{\mathrm{j}}\,(\text{place} \wedge \text{time} \supset \text{appointment}) \qquad (44)$$

$$\text{valid}\ \Box_{\mathrm{j}}\,\Box_{\mathrm{p}}\,\text{appointment} \wedge \Box_{\mathrm{p}}\,\Box_{\mathrm{j}}\,\text{appointment} \qquad (45)$$

The combined proof problem for Example 8 is

$$(37), \ldots, (44) \models^{\mathcal{STT}} (45) \qquad (46)$$

Example 8 (Wise men puzzle). Once upon a time, a king wanted to find the wisest out of his three wisest men. He arranged them in a circle and told them that he would put a white or a black spot on their foreheads and that one of the three spots would certainly be white. The three wise men could see and hear each other but, of course, they could not see their faces reflected anywhere. The king, then, asked to each of them to find out the color of his own spot. After a while, the wisest correctly answered that his spot was white.

We employ a 4-dimensional logic combining the modalities $\Box_{\mathrm{a}}$, $\Box_{\mathrm{b}}$, and $\Box_{\mathrm{c}}$, for encoding the individual knowledge of the three wise men, and a box operator $\Box_{\mathrm{fool}}$, for encoding the knowledge that is common to all of them. The entire encoding consists now of the following axioms for $X, Y, Z \in \{a, b, c\}$ and $X \neq Y \neq Z$:

$$\text{valid } \Box_{\mathrm{fool}}\,((\text{ws a}) \lor (\text{ws b}) \lor (\text{ws c})) \tag{47}$$
$$\text{valid } \Box_{\mathrm{fool}}\,((\text{ws } X) \supset \Box_{Y}\,(\text{ws } X)) \tag{48}$$
$$\text{valid } \Box_{\mathrm{fool}}\,(\neg\,(\text{ws } X) \supset \Box_{Y}\,\neg\,(\text{ws } X)) \tag{49}$$
$$\text{valid } \forall\phi_{\blacksquare}\; \Box_{\mathrm{fool}}\,\phi \supset \phi \tag{50}$$
$$\text{valid } \forall\phi_{\blacksquare}\; \Box_{\mathrm{fool}}\,\phi \supset \Box_{\mathrm{fool}}\,\Box_{\mathrm{fool}}\,\phi \tag{51}$$
$$\text{valid } \forall\phi_{\blacksquare}\; \Box_{\mathrm{fool}}\,\phi \supset \Box_{\mathrm{a}}\,\phi \tag{52}$$
$$\text{valid } \forall\phi_{\blacksquare}\; \Box_{\mathrm{fool}}\,\phi \supset \Box_{\mathrm{b}}\,\phi \tag{53}$$
$$\text{valid } \forall\phi_{\blacksquare}\; \Box_{\mathrm{fool}}\,\phi \supset \Box_{\mathrm{c}}\,\phi \tag{54}$$
$$\text{valid } \forall\phi_{\blacksquare}\; \neg\,\Box_{\mathrm{X}}\,\phi \supset \Box_{\mathrm{Y}}\,\neg\,\Box_{\mathrm{X}}\,\phi \tag{55}$$
$$\text{valid } \forall\phi_{\blacksquare}\; \Box_{\mathrm{X}}\,\phi \supset \Box_{\mathrm{Y}}\,\Box_{\mathrm{X}}\,\phi \tag{56}$$
$$\text{valid } \neg\,\Box_{\mathrm{a}}\,(\text{ws a}) \tag{57}$$
$$\text{valid } \neg\,\Box_{\mathrm{b}}\,(\text{ws b}) \tag{58}$$

From these assumptions we want to conclude that

$$\text{valid } \Box_{\mathrm{c}}\,(\text{ws c}) \tag{59}$$

Axiom (47) says that a, b, or c must have a white spot and that this information is known to everybody. Axioms (48) and (49) express that it is generally known that if someone has a white spot (or not) then the others see and hence know this. $\Box_{\mathrm{fool}}$ is axiomatized as an S4 modality in axioms (50) and (51). For $\Box_{\mathrm{a}}$, $\Box_{\mathrm{b}}$, and $\Box_{\mathrm{c}}$ it is sufficient to consider K modalities. The relation between those and common knowledge ($\Box_{\mathrm{fool}}$ modality) is axiomatized in inclusion axioms (52)–(55). Axioms (55) and (56) encode that whenever a wise man does (not) know something the others know that he does (not) know this. Axioms (57) and (58) say that a and b do not know whether they have a white spot. Finally, conjecture (59) states that that c knows he has a white spot. The combined proof problem for Example 7 is

$$(47), \ldots, (58) \models^{\mathcal{STT}} (59) \tag{60}$$

Example 9. A trivial example problem for the region connection calculus is (adapted from [20], p. 80):

$$\begin{aligned}&(\mathit{tpp}\ \text{catalunya spain}),\\&(\mathit{ec}\ \text{spain france}),\\&(\mathit{ntpp}\ \text{paris france}),\\&\models^{\mathcal{STT}} (\mathit{dc}\ \text{catalunya paris}) \wedge (\mathit{dc}\ \text{spain paris})\end{aligned} \tag{61}$$

The assumptions express that (i) Catalunya is a border region of Spain, (ii) Spain and France are two different countries sharing a common border, and (iii) Paris is a proper part of France. The conjecture is that (iv) Catalunya and Paris are disconnected as well as Spain and Paris.

Example 10. Within our $\mathcal{STT}$ framework we can easily put such spatial reasoning examples in an epistemic context, that is, we can model the individual spatial knowledge of different agents. Similar to before we here distinguish between common knowledge (fool) and the knowledge of person bob:

$$\begin{aligned}&\text{valid}\ \forall\phi_{\blacksquare}\ \Box_{\text{fool}}\,\phi \supset \Box_{\text{bob}}\,\phi,\\&\text{valid}\ \Box_{\text{bob}}\,(\lambda W_{\blacksquare}\,(\mathit{tpp}\ \text{catalunya spain})),\\&\text{valid}\ \Box_{\text{fool}}\,(\lambda W_{\blacksquare}\,(\mathit{ec}\ \text{spain france})),\\&\text{valid}\ \Box_{\text{bob}}\,(\lambda W_{\blacksquare}\,(\mathit{ntpp}\ \text{paris france}))\\&\models^{\mathcal{STT}}\\&\text{valid}\ \Box_{\text{bob}}\,(\lambda W_{\blacksquare}\,((\mathit{dc}\ \text{catalunya paris}) \wedge (\mathit{dc}\ \text{spain paris})))\end{aligned} \tag{62}$$

We here express that (ii) from above is commonly known, while (i) and (iii) are not. (i) and (iii) are known to the educated person bob though. In this situation, conjecture (iv) still follows for bob. However, it does not follow when replacing bob by common knowledge (hence, the following problem is not provable):

$$\ldots \models^{\mathcal{STT}} \text{valid}\ \Box_{\text{fool}}\,(\lambda W_{\blacksquare}\,((\mathit{dc}\ \text{catalunya paris}) \wedge (\mathit{dc}\ \text{spain paris}))) \tag{63}$$

In order to facilitate the combination of logics we have here lifted the region connection calculus propositions of type o to modal propositions of type $\mu \to o$ by λ-abstraction. Thus, the region connection calculus statements can now be applied to possible worlds; they evaluate statically though for all possible worlds since the λ-abstracted variable W is fresh for the encapsulated region connection calculus proposition.

$$\underbrace{(\mathit{tpp}\ \text{catalunya spain})}_{\text{type}\,o} \longrightarrow \underbrace{(\lambda W_{\blacksquare}\,(\mathit{tpp}\ \text{catalunya spain}))}_{\text{type}\,\iota\to o}$$

6 Experiments

In our case studies, we have employed the $\mathcal{STT}$ automated reasoners LEO-II—v1.1 [12], TPS—3.080227G1d [4], IsabelleP—2009-1, IsabelleM—2009-1, and

Table 1. Performance results of $\mathcal{STT}$ provers for problems in paper

Problem	TPTP id	LEO-II	TPS	IsabelleP
Reasoning about Logics and Combined Logics				
(1) ⇒ (11)	LCL699^1.p	0.0	0.3	3.6
(2) ⇒ (12)	LCL700^1.p	0.0	0.3	13.9
(3) ⇒ (13)	LCL701^1.p	0.0	0.3	4.0
(4) ⇒ (14)	LCL702^1.p	0.0	0.3	15.9
(5) ⇒ (15)	LCL703^1.p	0.1	0.3	16.0
(6) ⇒ (16)	LCL704^1.p	0.0	0.3	3.6
(7) ⇒ (17)	LCL705^1.p	0.1	51.2	3.9
(8) ⇒ (18)	LCL706^1.p	0.1	0.3	3.9
(9) ⇒ (19)	LCL707^1.p	0.1	0.3	3.6
(10) ⇒ (20)	LCL708^1.p	0.1	0.3	4.1
(1) ⇐ (11)	LCL709^1.p	0.0	0.3	3.7
(2) ⇐ (12)	LCL710^1.p	—	0.3	53.8
(3) ⇐ (13)	LCL711^1.p	0.0	0.3	3.7
(4) ⇐ (14)	LCL712^1.p	0.0	0.3	3.8
(5) ⇐ (15)	LCL713^1.p	—	0.8	67.0
(6) ⇐ (16)	LCL714^1.p	1.6	0.3	29.3
(7) ⇐ (17)	LCL715^1.p	37.9	—	—
(8) ⇐ (18)	LCL716^1.p	—	6.6	—
(9) ⇐ (19)	LCL717^1.p	—	—	—
(10) ⇐ (20)	LCL718^1.p	0.1	0.4	8.1
(21)		0.1	0.4	4.3
(22)		0.2	27.4	4.0
(23)		0.1	8.9	4.0
(24)		0.1	1.2	3.7
(25)		0.1	1.7	4.2
(26)		0.2	14.8	5.4
(27)		0.1	0.6	3.7
(28)		0.2	2.3	4.0
(29)		0.1	0.9	3.9
(30)		0.1	12.8	16.5
(31)$^{\text{Countersatisfiable}}$		—	—	—
(32)		0.0	0.3	3.6
(33)		0.0	0.3	3.6
(34)		0.1	0.4	3.6
(35)		0.2	35.5	—
(36)		0.4	—	—
Reasoning within Combined Logics				
(46)	PUZ086^1.p	0.1	—	102.4
(60)	PUZ087^1.p	0.3	—	—
(61)		2.3	—	112.7
(62)		20.4	—	—
(63)$^{\text{Countersatisfiable}}$		—	—	—

IsabelleN—2009-1.[5] These systems are available online via the SystemOnTPTP tool [32] and they support the new TPTP THF infrastructure for typed higher-order logic [15]. Their reasoning power is currently improving rapidly.

The axiomatizations of $\mathcal{QML}^{\mathcal{STT}}$ and $\mathcal{IPL}^{\mathcal{STT}}$ are available as LCL013^0.ax and LCL010^0.ax in the TPTP library.[6] The example problems LCL698^1.p and LCL695^1.p ask about the satisfiability of these axiomatizations. Both questions are answered positively by the Satallax prover [5] in less than a second.

Table 1 presents the results of our experiments; the timeout was set to 120 seconds and the entries in the table are reported in seconds. Those examples which have already entered the new higher-order TPTP library are presented with their respective TPTP identifiers in the second column and the others have meanwhile been submitted and will appear in a forthcoming TPTP release.

As expected, (31) and (63) cannot be proved by any prover and IsabelleN reports a counterexample for (31) in 34.4 seconds and for (63) in 39.7 seconds.

In summary, all but one of our example problems can be solved effectively by at least one of the reasoners. In fact, most of our example problems require only milliseconds. LEO-II solves most problems and it is the fastest prover in our experiment.

As mentioned before, we are not aware of any other running system that can handle all of the above problems.

7 Conclusion

Our overall goal is to show that various interesting classical and non-classical logics and their combinations can be elegantly mechanized and partly automated in modern higher-order reasoning systems with the help of our logic embeddings.

Our experiments are encouraging and they provide first evidence for our claim that $\mathcal{STT}$ is suited as a framework for combining classical and non-classical logics. It is obvious, however, that $\mathcal{STT}$ reasoners should be significantly improved for fruitful application to more challenge problems in practice. The author is convinced that significant improvements — in particular for fragments of $\mathcal{STT}$ as illustrated in this paper — are possible and that they will be fostered by the new TPTP infrastructure and the new yearly higher-order CASC competitions.

Note that when working with our reasoners from within a proof assistant such as Isabelle/HOL the user may also provide interactive help if the reasoning tasks are still to challenging, for example, by formulating some lemmas or by splitting proof tasks in simpler subtasks.

An advantage of our approach also is that provers such as our LEO-II are generally capable of producing verifiable proof output, though much further

[5] IsabelleM and IsabelleN are model finder in the Isabelle proof assistant [29] that have been made available in batch mode, while IsabelleP applies a series of Isabelle proof tactics in batch mode.

[6] Note that the types μ and ι are unfortunately switched in the encodings available in the TPTP: the former is used for individuals and the latter for worlds. This syntactic switch is completely unproblematic.

work is needed to make these proof protocols exchangeable between systems or to explain them to humans. Furthermore, it may be possible to formally verify the entire theory of our embedding(s) within a proof assistant.

The work presented in this paper has its roots in the LEO-II project (in 2006/2007 at University of Cambridge, UK) in which we first studied and employed the presented embedding of quantified and propositional multimodal logics in $\mathcal{STT}$ [9,11]. This research, amongst others, is currently continued in the DFG project ONTOLEO (BE 2501/6-1). In ONTOLEO we study whether our approach can be applied to automate modalities in ontology reasoning [14,13]. However, our work is relevant also for other application directions, including multi-agent systems. Studying the scalability of our approach for a range of applications is thus important future work.

Acknowledgment. The author is indebted to Larry Paulson, Geoff Sutcliffe, and Chad Brown. Larry Paulson, together with the author, initiated the LEO-II project at Cambridge University (EPRSC grant LEO-II EP/D070511/1). Geoff Sutcliffe, in collaboration with the author and supported by several further contributors, developed the new higher-order TPTP THF infrastructure (EU FP7 grant THFTPTP PIIF-GA-2008-219982). Both projects had a significant impact on the work presented in this article. Moreover, Chad Brown originally inspired this research with a presentation at Loria, Nancy in 2005.

References

1. Andrews, P.B.: General models and extensionality. Journal of Symbolic Logic 37, 395–397 (1972)
2. Andrews, P.B.: General models, descriptions, and choice in type theory. Journal of Symbolic Logic 37, 385–394 (1972)
3. Andrews, P.B.: An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof, 2nd edn. Kluwer Academic Publishers, Dordrecht (2002)
4. Andrews, P.B., Brown, C.E.: TPS: A hybrid automatic-interactive system for developing proofs. Journal of Applied Logic 4(4), 367–395 (2006)
5. Backes, J., Brown, C.E.: Analytic tableaux for higher-order logic with choice. In: Giesl, J., Haehnle, R. (eds.) IJCAR 2010 - 5th International Joint Conference on Automated Reasoning, Edinburgh, UK. LNCS (LNAI). Springer, Heidelberg (2010)
6. Baldoni, M.: Normal Multimodal Logics: Automatic Deduction and Logic Programming Extension. PhD thesis, Universita degli studi di Torino (1998)
7. Benzmüller, C.: Automating access control logic in simple type theory with LEO-II. In: Gritzalis, D., López, J. (eds.) Proceedings of 24th IFIP TC 11 International Information Security Conference, SEC 2009, Emerging Challenges for Security, Privacy and Trust, Pafos, Cyprus, May 18-20. IFIP, vol. 297, pp. 387–398. Springer, Heidelberg (2009)
8. Benzmüller, C., Brown, C.E., Kohlhase, M.: Higher order semantics and extensionality. Journal of Symbolic Logic 69, 1027–1088 (2004)
9. Benzmüller, C., Paulson, L.: Exploring Properties of Normal Multimodal Logics in Simple Type Theory with LEO-II. In: Festschrift in Honor of Peter B. Andrews on His 70th Birthday. Studies in Logic, Mathematical Logic and Foundations. College Publications (2008)

10. Benzmüller, C., Paulson, L.C.: Quantified Multimodal Logics in Simple Type Theory. SEKI Report SR–2009–02, SEKI Publications, DFKI Bremen GmbH, Safe and Secure Cognitive Systems, Cartesium, Enrique Schmidt Str. 5, D–28359 Bremen, Germany (2009), ISSN 1437-4447, http://arxiv.org/abs/0905.2435
11. Benzmüller, C., Paulson, L.C.: Multimodal and intuitionistic logics in simple type theory. The Logic Journal of the IGPL (2010)
12. Benzmüller, C., Paulson, L.C., Theiss, F., Fietzke, A.: LEO-II — A Cooperative Automatic Theorem Prover for Higher-Order Logic. In: Baumgartner, P., Armando, A., Gilles, D. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 162–170. Springer, Heidelberg (2008)
13. Benzmüller, C., Pease, A.: Progress in automating higher-order ontology reasoning. In: Konev, B., Schmidt, R., Schulz, S. (eds.) Workshop on Practical Aspects of Automated Reasoning (PAAR 2010), Edinburgh, UK, July 14. CEUR Workshop Proceedings (2010)
14. Benzmüller, C., Pease, A.: Reasoning with embedded formulas and modalities in SUMO. In: Bundy, A., Lehmann, J., Qi, G., Varzinczak, I.J. (eds.) The ECAI 2010 Workshop on Automated Reasoning about Context and Ontology Evolution (ARCOE 2010), Lisbon, Portugal, August 16-17 (2010)
15. Benzmüller, C., Rabe, F., Sutcliffe, G.: THF0 — The Core TPTP Language for Classical Higher-Order Logic. In: Baumgartner, P., Armando, A., Gilles, D. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 491–506. Springer, Heidelberg (2008)
16. Blackburn, P., Marx, M.: Tableaux for quantified hybrid logic. In: Egly, U., Fermüller, C.G. (eds.) TABLEAUX 2002. LNCS (LNAI), vol. 2381, pp. 38–52. Springer, Heidelberg (2002)
17. Braüner, T.: Natural deduction for first-order hybrid logic. Journal of Logic, Language and Information 14(2), 173–198 (2005)
18. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic 5, 56–68 (1940)
19. Fitting, M.: Interpolation for first order S5. Journal of Symbolic Logic 67(2), 621–634 (2002)
20. Gabbay, D., Kurucz, A., Wolter, F., Zakharyaschev, M.: Many-dimensional modal logics: theory and applications. Studies in Logic, vol. 148. Elsevier Science, Amsterdam (2003)
21. Garg, D., Abadi, M.: A Modal Deconstruction of Access Control Logics. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 216–230. Springer, Heidelberg (2008)
22. Garson, J.: Modal logic. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy (Winter 2009)
23. Gödel, K.: Eine interpretation des intuitionistischen aussagenkalküls. Ergebnisse eines Mathematischen Kolloquiums 8, 39–40 (1933); Also published in Gödel [24], pp. 296–302
24. Gödel, K.: Collected Works, vol. I. Oxford University Press, Oxford (1986)
25. Goldblatt, R.: Logics of Time and Computation. Center for the Study of Language and Information - Lecture Notes, vol. 7. Leland Stanford Junior University (1992)
26. Henkin, L.: Completeness in the theory of types. Journal of Symbolic Logic 15, 81–91 (1950)
27. Kaminski, M., Smolka, G.: Terminating tableau systems for hybrid logic with difference and converse. Journal of Logic, Language and Information 18(4), 437–464 (2009)
28. McKinsey, J.C.C., Tarski, A.: Some theorems about the sentential calculi of lewis and heyting. Journal of Symbolic Logic 13, 1–15 (1948)

29. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic. LNCS, vol. 2283. Springer, Heidelberg (2002)
30. Randell, D.A., Cui, Z., Cohn, A.G.: A spatial logic based on regions and connection. In: Proceedings 3rd International Conference on Knowledge Representation and Reasoning, pp. 165–176 (1992)
31. Segerberg, K.: Two-dimensional modal logic. Journal of Philosophical Logic 2(1), 77–96 (1973)
32. Sutcliffe, G.: TPTP, TSTP, CASC, etc. In: Diekert, V., Volkov, M.V., Voronkov, A. (eds.) CSR 2007. LNCS, vol. 4649, pp. 6–22. Springer, Heidelberg (2007)
33. Sutcliffe, G.: The TPTP problem library and associated infrastructure. J. Autom. Reasoning 43(4), 337–362 (2009)
34. Sutcliffe, G., Benzmüller, C.: Automated reasoning in higher-order logic using the TPTP THF infrastructure. Journal of Formalized Reasoning 3(1), 1–27 (2010)
35. Sutcliffe, G., Benzmüller, C., Brown, C., Theiss, F.: Progress in the development of automated theorem proving for higher-order logic. In: Schmidt, R.A. (ed.) Automated Deduction – CADE-22. LNCS, vol. 5663, pp. 116–130. Springer, Heidelberg (2009)

Speculative Abductive Reasoning for Hierarchical Agent Systems

Jiefei Ma[1], Krysia Broda[1], Randy Goebel[2], Hiroshi Hosobe[3], Alessandra Russo[1], and Ken Satoh[3]

[1] Department of Computing, Imperial College London, United Kingdom
[2] Department of Computer Science, University of Alberta, Canada
[3] National Institute of Informatics, Japan

Abstract. Answer sharing is a key element in multi-agent systems as it allows agents to collaborate towards achieving a global goal. However exogenous knowledge of the world can influence each agent's local computation, and communication channels may introduce delays, creating multiple partial answers at different times. Agent's answers may, therefore, be incomplete and revisable, giving rise to the concept of speculative reasoning, which provides a framework for managing multiple revisable answers within the context of multi-agent systems. This paper extends existing work on speculative reasoning by introducing a new abductive framework to hierarchical speculative reasoning. This allows speculative reasoning in the presence of both negation and constraints, enables agents to receive conditional answers and to continue their local reasoning using default answers, thus increasing the parallelism of agents collaboration. The paper describes the framework and its operational model, illustrates the main features with an example and states soundness and completeness results.

1 Introduction

A Multi-agent System (MAS) is comprised of multiple intelligent agents, which collaborate in distributed problem solving. Each agent has local (and partial) knowledge of the world and cannot solve a given problem alone. The restricted range of an agent's capability could be due to lack of computational power/resources, or to incomplete knowledge. Hence, agents must be able to interact and exchange tasks or knowledge. For example, during a reasoning task, when an agent has a goal for which it has no knowledge, it may query another agent and attempt to use an answer to continue the reasoning process. However, two important issues arise. First, physical communication channels often introduce delays, and the queried agent may incur an unacceptable delay to process the query (e.g., it may ask another agent too). So a timely query-answer round trip cannot be guaranteed. Second, the queried agent may not return all the answers at once (i.e., an answer is returned as soon as it is found), and the queried agent may also revise previous replies. For example, when a change of the world is detected, an agent needs to revise its local knowledge accordingly, which may subsequently affect the answers already computed for its received query. Such *belief revision* must be propagated to the querying agent through *answer revision*.

While awaiting answers, the querying agent may suspend and wait. Alternatively, if the query has a finite number of possible answers, the agent may continue reasoning in parallel with each of them. When the final answers are returned, the querying agent can discard the computations that used uncorroborated answers. However, these two extreme approaches have limitations. The former leaves CPU resources idle, whereas the latter may expend CPU cycles doing unnecessary computations. Even worse, if the answers are not returned concurrently, the latter is unable to determine which computation(s) to discard. But there is a compromise: if the agent has access to some *default answers* (*default* in brief) of the query, e.g., the answers that are somehow most likely, then it may continue the reasoning using only the defaults. When an answer (either *new* or *revised*) is available, the agent will revise the current computations and begin new computations entailed by the new answer beyond defaults. This compromise method is known as *speculative computation*. The defaults can be obtained based on an agent's previous partial knowledge, and can be considered as heuristics used for prioritising parallel computation by the querying agent, which has an impact on the performance of speculative computation.

Speculative computation in MAS was first proposed in [14] for master-slave structured systems where an answer to a (ground) query is simply either *yes* or *no*. Since then it has been extended for hierarchical MAS [15], and for constraint processing [12,2,7] where the answers can be a set of *constraints* over the variables in the (non-ground) query. The second extension allows a much wider range of problems to be modelled and solved. For example, an agent can query another about $available\_day(X)$ and receive answer $X \in \{sat, sun, wed\}$. However, in many real world applications, support for *conditional answers* and *negations* is also desired. For example, an answer to the above query could be $sunny(X) \wedge \neg teaching(X)$ instead of a simple finite domain constraint.

Abduction [9] is a powerful inference mechanism that enables the generation of conditional proofs (answers). The conditions (i.e. assumptions) of a reasoning process, together with the given background knowledge, imply the conclusion of the proof (i.e. the query). In this paper, we extend existing results by presenting a very general speculative reasoning framework for hierarchical agent systems, where each agent can perform abductive reasoning and can accept revisable conditional answers from other agents. The operational model of the framework is a non-trivial extension to the speculative constraint processing model [7] and the well-known abductive proof procedure [18], which supports *constructive negation* [19].

The rest of the paper is organised as follows. Section 2 discusses related work. Section 3 and Section 4 present the speculative abductive framework and its operational model respectively, by using a planning example. Open issues, such as the requirement for agent hierarchy and the usage of defaults, are discussed in Section 5. Finally, Section 6 concludes the paper and describes future work.

2 Related Work

Speculative computation has originally been investigated in the context of parallel computing to speed computation [1], and has been applied to areas such as optimistic transactions in database and efficient execution mechanisms for parallel logic programming [4].

Our work, although inspired by some of these techniques, provides a more general formalization of the computational process, appropriate for multi-agent systems.

Agent-based belief revision approaches have been proposed in the context of planning in *dynamic* environments (e.g. [16,5,6,13]). [16] and [5] do not consider speculative computation – in [16] the reasoning restarts every time a new piece of information leads to contradiction, and in [5] the *Dynamic SLDNF* procedure, which supports dynamic addition and deletion of rules, allows previous planning fragments to be reused, but it does not support abduction and conditional answers. In [6], each plan has a protected condition as a set of "primitive fluents". When a primitive fluent is added or deleted, new valid plans are generated or invalid plans are removed. This is similar to the idea proposed in [15]. [13] is the first work to apply speculative computation in abduction. However, its operational model does not support constructive negation (although it supports *negation as failure* [20]) and constraints processing, and is designed for master-slave agent systems where only the master can perform speculative reasoning. In contrast, our framework is very general. It can be used for hypothetical multi-agent reasoning with exogenous knowledge, including but not limited to hierarchical planning with constraints (e.g., cost or time) in a dynamic environment. Our system differs from existing systems for decentralised abduction [3,11]. These systems are concerned with finding consistent hypotheses among collaborative agents, where the total knowledge is distributed (not necessarily in a hierarchical way). But, they do not handle revisable answers, which is the case of our system.

3 Formulation

Here we present basic definitions using conventional concepts and notation from Logic Programming [10]. Given a multi-agent system, we assume each agent has a unique constant identifier, and the system of agents is organized in a *hierarchical* structure. An *atom* is either $p(\boldsymbol{t})$ (called *non-askable atom* or *non-askable* in brief) or $p(\boldsymbol{t})@S$ (called *askable atom* or *askable* in brief), where p is its predicate (name), $\boldsymbol{t}$ is a shorthand for its arguments as a vector of terms $t_1, \ldots, t_n$ ($t \geq 0$), and S is an agent identifier. A *literal* can be an atom, the negation of an atom A (denoted with $\neg A$), or a constraint in CLP [8] over finite domains. A *clause* is either a *rule* $H \leftarrow L_1 \wedge \cdots \wedge L_n$ ($n \geq 0$), or a *denial* $\leftarrow L_1 \wedge \cdots \wedge L_n$ ($n > 0$), where H (called the *head*) is an atom, and each L_i in $L_1 \wedge \cdots \wedge L_n$ (called the *body*) is a literal. Any variable in a clause, unless stated otherwise, is implicitly universally quantified with the scope the whole clause.

Definition 1. (Agent Hierarchy) *An agent hierarchy $\mathcal{H}$ is a tree consisting of a set of nodes, each of which is an agent. The set of all agents in $\mathcal{H}$ is denoted $ags(\mathcal{H})$. The root of $\mathcal{H}$, denoted $root(\mathcal{H})$, is called the* root agent. *A non-leaf agent is called an* internal agent*, and a leaf agent is called an* external agent*. Let $int(\mathcal{H})$ and $ext(\mathcal{H})$ denote the set of internal agents and the set of external agents respectively. Given an internal agent M, $chi(M, \mathcal{H})$ denotes the set of its children nodes, each of which is called a* child agent *of M. Given a non-root agent S, $par(S, \mathcal{H})$ denotes its parent node, called the* parent agent *of S.*

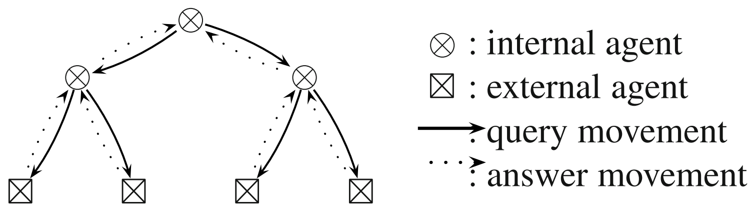


Fig. 1. Agent Hierarchy

Intuitively, the hierarchical relationship amongst agents is such that a parent agent can receive partial answers from children, and then consolidate partial or "local" answers into a global answer. The speculative nature of computation arises because partial answers can be delivered at different times with different "answers", and the parent agent's job is to consolidate answers as required. In Figure 1, the idea is that external agents at the leaves are working in an environment and passing information back up the tree, and that parent agents exercise some kind of heuristic discretion about how to combine answers to build the "best" answer.

Definition 2. **(Agent Specification)** *Given an agent hierarchy $\mathcal{H}$, the agent specification of an internal agent $i \in int(\mathcal{H})$ is $T_i = \langle F_i, \mathcal{D}_i \rangle$, where:*

- *F_i is an* abductive framework *$\langle \Pi_i, \mathcal{AB}_i, \mathcal{I}_i \rangle$, such that:*
  - *$\mathcal{AB}_i$ is a set of special non-askable predicates called* abducible predicates. *A non-askable with abducible predicate is called an* abducible atom *(*abducible *in brief). A non-askable without abducible predicate is called a* non-abducible atom *(*non-abducible *in brief).*
  - *Π_i is called the* background knowledge*, and is a set of rules whose heads are non-abducibles;*
  - *$\mathcal{I}_i$ is called the* integrity constraints*, and is a set of denials, each of which must contain at least one abducible as body literal.*
- *for every askable $p(\boldsymbol{t})@S$ appearing in F_i, p must not be an abducible predicate and S must be a child agent of i, i.e. $p \notin \mathcal{AB}_i$ and $S \in chi(i, \mathcal{H})$.*
- *$\mathcal{D}_i$ is a set of rules whose heads are askables and whose bodies do not contain askables. $\mathcal{D}_i$ is called the* default answers *for the askables in F_i.*

Definition 3. **(Speculative Framework)** *The speculative framework for a hierarchical abductive agent system is a pair $\mathcal{M} = \langle \mathcal{H}, \mathcal{T} \rangle$, where $\mathcal{H}$ is an agent hierarchy, and $\mathcal{T}$ is a set of internal agent specifications, i.e. $\mathcal{T} = \{T_i | i \in int(\mathcal{H})\}$.*

We use the following delivery problem as a running example:

Example 1. A logistics company (l) takes delivery orders from customers, and outsources the subtasks to its partners, e.g. a transport company (d), who has contracts with flight pilots (e.g. a') and lorry drivers (e.g., b' and c'). The logistics company also needs real-time information, such as the weather from a public service (e.g. s'), to help with planning. These entities can be abstracted as a hierarchy $\mathcal{H}$ of agents, where l (the root) and d are internal agents, and a', b', c', s' are external agents, i.e.

$$\mathcal{H} = \{l \rightarrowtail d, l \rightarrowtail s', d \rightarrowtail a', d \rightarrowtail b', d \rightarrowtail c'\}^{1}$$

The agent specifications for l and d are as follows:

$$\Pi_l = \left\{ \begin{array}{l} deliver(Src, Des, Cost, Mtd) \leftarrow \\ \quad covers(Src, Des, Who), \\ \quad quote(Src, Des, Cost, Mtd)@Who. \\ covers(Src, Des, d) \leftarrow \\ \quad Src \in \{1,2,3\}, Des \in \{1,2,3\}. \\ safe\_weather(Area) \leftarrow \neg(storm(Area)@s'). \end{array} \right\}$$

$$\mathcal{D}_l = \left\{ \begin{array}{l} storm(Area)@s' \leftarrow Area = 3. \\ quote(Src, Des, 5, air)@d. \end{array} \right\}$$

$$\Pi_d = \left\{ \begin{array}{l} quote(Src, Des, TotalCost, air) \leftarrow \\ \quad flight(Src, Des, Cost)@a', \\ \quad safe\_weather(Src), safe\_weather(Des), \\ \quad TotalCost = Cost + 1. \\ quote(Src, Des, TotalCost, land) \leftarrow \\ \quad lorry(Src, Mid, Cost1)@b', \\ \quad lorry(Mid, Des, Cost2)@c', \\ \quad TotalCost = Cost1 + Cost2 + 2. \end{array} \right\}$$

$$\mathcal{D}_d = \left\{ \begin{array}{l} flight(Src, Des, 3)@a' \leftarrow \\ \quad Src \in \{1,2,3\}, Des \in \{1,2,3\}. \\ lorry(Src, Des, 2)@b' \leftarrow Src \in \{1,3\}, Des \in \{1,3\}. \\ lorry(Src, Des, 2)@c' \leftarrow Src \in \{2,3\}, Des \in \{2,3\}. \end{array} \right\}$$

and $\mathcal{AB}_d = \{safe\_weather/1\}$, $\mathcal{AB}_l = \mathcal{I}_l = \mathcal{I}_d = \emptyset$.

Semantics

A *global goal* is a conjunction of literals $G_1 \wedge \cdots \wedge G_n$ that can be issued to the root agent. During hierarchical speculative computation, internal agents can send *queries* to and receive *replies* from their children agents. A query is an askable $Q@S$, where S is the recipient's identifier. A reply for $Q@S$ is "$Q@S \leftarrow B$" by S, where B is a (possibly empty) conjunction of non-askable literals. A query/reply is *internal* if S is an internal agent; otherwise it is *external*.

Definition 4. (Agent Belief State) *Given a speculative framework $\mathcal{M} = \langle \mathcal{H}, \mathcal{T} \rangle$, let $T_A = \langle F, \mathcal{D} \rangle$ be the specification of an internal agent A where $F = \langle \Pi, \mathcal{AB}, \mathcal{I} \rangle$, and let $\mathcal{R}$ be the set of the latest replies received by A (called* reply set*), then the current agent belief state of A w.r.t. $\mathcal{R}$ is $\langle \Pi^{bel}, \mathcal{AB}, \mathcal{I} \rangle$, where*

$$\begin{array}{l} \Pi^{bel} = \Pi \cup \mathcal{R} \cup \{\text{“}Q@S \leftarrow B\text{”} \mid \\ \quad \text{“}Q@S \leftarrow B\text{”} \in \mathcal{D} \wedge \neg\exists B'.[\text{“}Q@S \leftarrow B'\text{”} \in \mathcal{R}]\} \end{array}$$

and is denoted $BEL(T_A, \mathcal{R})$.

[1] "$Parent \rightarrowtail Child$" represents an arc of the tree.

Upon receiving a query, the internal agent initiates a *local abductive task* to compute *local abductive answers* for the query:

Definition 5. (Local Abductive Answer) *Given an internal query $Q@S$, let $BEL(T, \mathcal{R}) = \langle \Pi^{bel}, \mathcal{AB}, \mathcal{I} \rangle$ be the current belief state of agent S. A local abductive answer by S for Q w.r.t. $BEL(T, \mathcal{R})$, is a pair $\langle \Delta, \mathcal{C} \rangle$ such that:*

- *Δ is a set of abducibles from $\mathcal{AB}$, and $\mathcal{C}$ is a set of constraints;*
- *$\Pi^{bel} \cup \Delta \models_{sm} Q\theta$, where θ is a set of variable substitutions induced by $\mathcal{C}$;*
- *$\Pi^{bel} \cup \Delta \models_{sm} \mathcal{I}$.*

where $\models_{sm}$ is the logical entailment under a user selected semantics.

During the hierarchical computation by the agents, each abductive answer $\langle \Delta, \mathcal{C} \rangle$ computed by agent S for the query $Q@S$ is returned as an internal reply of the form "$Q@S \leftarrow \bigwedge_{C \in \mathcal{C}} C \wedge \bigwedge_{A \in \Delta} A$". External replies, on the other hand, are not computed (by the internal agents) and are the exogenous knowledge of the agent system, which may be incomplete and are revisable at any time by external agents. Thus, given an internal agent with specification $\langle \mathcal{F}, \mathcal{D} \rangle$, we separate its reply set $\mathcal{R}$ into two disjoint sets $\mathcal{R}_{int}$ and $\mathcal{R}_{ext}$, which contain the received internal and external replies respectively. The agent's belief state is said to be *stable* if and only if $\mathcal{R}_{int}$ is the set of all the answers for all the askables appearing in $\mathcal{F}$. Finally, an agent system is *stable* if and only if all of its internal agents are stable.

Definition 6. (Global Abductive Answer) *Let $\mathcal{M} = \langle \mathcal{H}, \mathcal{T} \rangle$ be the speculative framework of a stable agent system, and let $BEL(T, \mathcal{R})$ be the belief state of the root agent $root(\mathcal{H})$. Given a global goal $\mathcal{G}$, a global abductive answer for $\mathcal{G}$ is a local abductive answer by $root(\mathcal{H})$ w.r.t. $BEL(T, \mathcal{R})$.*

4 Operational Model

Given a query, an internal agent will compute all answers with a local inference procedure, which consists of two phases: the *abductive reduction phase* and the *answer arrival phase*. The former is a process of reducing the set of local goals derived from the query, during which abducibles and constraints are collected, and new queries are sent out to children agents when askables are reduced. If no local goal is pending, an answer (for the received query) is extracted from the set of collected abducibles and constraints, and is returned to the querying parent. Whenever an answer (either new or revised) arrives from a child agent, the abductive reduction phase is interrupted and the answer arrival phase takes over, which revises the current computations accordingly. Instead of simply discarding any completed but outdated computation (i.e., one using a default answer or an old answer being revised), the revision process tries to reuse it as much as possible by adding to it a set of revision goals. This potential reuse is a major novelty of speculative reasoning and will be described in detail below.

The inference procedure is best described as a *state rewriting* process with a set of *inference rules*, where the current computations are represented as a set of *states*. Each state has a unique ID and has six components $\langle \mathcal{G}, \Delta, \mathcal{C}, \mathcal{N}, WA, AA \rangle$, where $\mathcal{G}$ is a set of local goals, Δ and $\mathcal{C}$ are the (collected) abducible and constraint stores respectively, $\mathcal{N}$ is a set of *dynamic denials*, WA and AA are the set of *waiting answers* and the set of *assumed answers* respectively. A local goal is either a literal, or a *failure goal* of the form $\forall \boldsymbol{X}.\leftarrow L_1 \wedge \cdots \wedge L_n$ ($n > 0$), where each L_i ($0 < i \leq n$) is a literal and $\boldsymbol{X}$ is the set of variables in $L_1 \wedge \cdots \wedge L_n$ that are universally quantified with the scope the whole failure goal. Each dynamic denial in $\mathcal{N}$ is a denial with the first body literal being an abducible. Each element in WA or AA has the form (AID, G), where G is either an askable $Q@S$ or a failure goal $\forall \boldsymbol{X}.\leftarrow Q@S \wedge L_1 \wedge \cdots \wedge L_n$ ($n \geq 0$), and AID is the unique ID of an *answer entry* (described shortly) for $Q@S$. WA and AA record what answers the current state is using for the askables it has reduced, so that it can be revised when a relevant answer arrives. Any free variable appearing in the state is considered to be existentially quantified with the scope the whole state.

There are three types of states: *active state*, *choice state* and *finished state*. At each abductive reduction (phase) step, an active state, with $\mathcal{G} \neq \emptyset$ and $WA = \emptyset$, is replaced by its child(ren) state(s) generated by an applicable inference rule on a local goal chosen according to some goal selection strategy. When a finished state, with $\mathcal{G} = \emptyset$ and $WA = \emptyset$, is generated, the answer extracted from its Δ and $\mathcal{C}$ is returned. Choice states, with $WA \neq \emptyset$, are generated when an askable is reduced or a new answer arrives.

To keep track of which state to revise and how, we need to maintain a store of *answer entries*. Each answer entry has a unique ID and has three components $\langle Type, Q@S, Ans \rangle$, where $Type$ indicates the entry's type, and Ans is an answer for the askable $Q@S$ associated with the entry. There are three answer entry types: *default*, *returned* and *status*. Each default answer entry corresponds to a default in the agent's specification, and each returned answer entry corresponds to an answer which arrived from a child agent. Each reduced askable has five status answer entries, all of which have empty Ans (i.e. $true$): *positive speculated original* (PSO), *positive non-speculated original* (PNO), *negative original* (NO), *negative speculated* (NS) and *negative non-speculated* (NN). They are used for recording how a choice state is created after the associated askable is reduced, or how an active state has reduced the askable. For example, PSO and PNO are for the positive reduction of askable atoms; whereas NO, NS and NN are for the negative reduction of askable atoms. The word "original" indicates that any state using the answer entry (PSO, PNO or NO) is a *choice state*. Their usage will be described in detail in Section 4.1.

4.1 State Rewriting

For any received query $Q_0@ag$, internal agent ag creates an (active) initial state $\langle \{Q_0\}, \Delta_0, \mathcal{C}_0, \mathcal{N}_0, WA_0, AA_0 \rangle$, where $\Delta_0 = \mathcal{C}_0 = \mathcal{N}_0 = WA_0 = AA_0 = \emptyset$. Let the agent's specification be $\langle \langle \Pi, \mathcal{AB}, \mathcal{I} \rangle, \mathcal{D} \rangle$; the state rewriting process begins as follows. At each step, an active state $St = \langle \mathcal{G}, \Delta, \mathcal{C}, WA, AA \rangle$ is selected, and a local goal L is chosen from $\mathcal{G}$ (let $\mathcal{G}^- = \mathcal{G} \setminus \{L\}$, only the differences between St and its children states are described):

1. [Reduce Non-abducible] if L is a non-abducible $p(\boldsymbol{t})$, St is replaced by states St_i with new goals $B_i[\boldsymbol{X}_i/\boldsymbol{t}]$[2] $\cup \mathcal{G}^-$ for each rule "$p(\boldsymbol{X}_i) \leftarrow B_i$" $\in \Pi$;
2. [Reduce Abducible] if L is an abducible $a(\boldsymbol{t})$, St is replaced by:
   - a state St_i with new goals $\mathcal{G}^-$ and consistent constraints $\{\boldsymbol{t} = \boldsymbol{v}_i\} \cup \mathcal{C}$ (i.e. *reuse abducible*) for each $a(\boldsymbol{v}_i) \in \Delta$, and
   - a state St' with new goals $\underline{E} \cup \underline{I} \cup \underline{N} \cup \mathcal{G}^-$ and new abducibles $\{a(\boldsymbol{t})\} \cup \Delta$ (i.e. *collect new abducible*), where $\underline{E} = \{\boldsymbol{t} \neq \boldsymbol{v}_i | a(\boldsymbol{v}_i) \in \Delta\}$, $\underline{I} = \{$"$\forall \boldsymbol{X}. \leftarrow \boldsymbol{t} = \boldsymbol{v}_i \wedge B_i^-$"$|$"$\leftarrow B_i$" $\in \mathcal{I}$ and $\boldsymbol{X} = vars(B_i)$[3] and $B_i^- = B_i \setminus \{a(\boldsymbol{v}_i)\}\}$, and $\underline{N} = \{$"$\forall \boldsymbol{X}. \leftarrow \boldsymbol{t} = \boldsymbol{v}_i \wedge B_i$"$|$"$\forall \boldsymbol{X}. \leftarrow a(\boldsymbol{v}_i) \wedge B_i$" $\in \mathcal{N}\}$;
3. [Reduce Constraint] if L is a constraint, St is updated with new goals $\mathcal{G}^-$ and new constraints $\{L\} \cup \mathcal{C}$ if they are consistent; otherwise, St is removed if they are inconsistent;
4. [Rewrite Negation] if L is a negation $\neg L'$, St is updated with new goals $\{\leftarrow L'\} \cup \mathcal{G}^-$;
5. [Reduce Askable] if L is an askable $Q@S$ where S is ground, see subsection **handling askable as a positive goal**;
6. [Fail Non-abducible] if $L =$ "$\forall \boldsymbol{X}. \leftarrow p(\boldsymbol{t}) \wedge B$" where $p(\boldsymbol{t})$ is a non-abducible, then St is updated with new goals $\mathcal{G}_{add} \cup \mathcal{G}^-$, where $\mathcal{G}_{add} = \{$"$\forall \boldsymbol{X}. \leftarrow \boldsymbol{t} = \boldsymbol{v}_i \wedge B_i' \wedge B$"$|$"$p(\boldsymbol{v}_i) \leftarrow B_i'$" $\in \Pi\}$;
7. [Fail Abducible] if $L =$ "$\forall \boldsymbol{X}. \leftarrow a(\boldsymbol{t}) \wedge B$" where $a(\boldsymbol{t})$ is an abducible and $vars(a(\boldsymbol{t})) \cap \boldsymbol{X} = \emptyset$, then St is updated with new goals $\mathcal{G}_{add} \cup \mathcal{G}^-$ and new dynamic denials $\{L\} \cup \mathcal{N}$, where $\mathcal{G}_{add} = \{$"$\forall \boldsymbol{X}. \leftarrow \boldsymbol{t} = \boldsymbol{v}_i \wedge B$"$|a(\boldsymbol{v}_i) \in \Delta\}$;
8. [Fail Constraint] if L="$\forall \boldsymbol{X}. \leftarrow C \wedge B$" where C is a constraint and $vars(C) \cap \boldsymbol{X} = \emptyset$:
   - if $\overline{C}$ is a negated constraint[4] of C and $\{\overline{C}\} \cup \mathcal{C}$ is consistent, St is updated with new goals $\mathcal{G}^-$ and new constraints $\{\overline{C}\} \cup \mathcal{C}$;
   - otherwise St is updated with new goals $\{$"$\forall \boldsymbol{X}. \leftarrow B$"$\} \cup \mathcal{G}^-$ and new constraints $\{C\} \cup \mathcal{C}$;
9. [Fail Negation] if $L =$ "$\forall \boldsymbol{X}. \leftarrow \neg L' \wedge B$", where $vars(L') \cap \boldsymbol{X} = \emptyset$, then St is replaced by two states with new goals $\{L'\} \cup \mathcal{G}^-$ and new goals $\{$"$\leftarrow L'$", "$\forall \boldsymbol{X}. \leftarrow B$"$\} \cup \mathcal{G}^-$ respectively;
10. [Fail Askable] if $L =$ "$\forall \boldsymbol{X}. \leftarrow Q@S \wedge B''$ where $vars(Q) \cap \boldsymbol{X} = \emptyset$ and S is ground, see subsection **handling askable in a failure goal**;

In inference rules (7)-(10), if any variable appearing in the selected literal of a failure goal is universally quantified, the selected literal is not *safe* and will cause *floundering*. In this case, the inference aborts and reports an error. Inference rules (6)-(10) apply to failure goals. If the selected failure goal has empty body, then none of the rules can be applied and the inference fails.

Handling Askable as a Positive Goal. Here we describe how an askable as a positive goal is reduced, and how its subsequent computations can be revised when a new or

[2] $B[\boldsymbol{X}/\boldsymbol{t}]$ is a formula obtained by substituting variables $\boldsymbol{X}$ in formula B with terms $\boldsymbol{t}$.

[3] $vars(B)$ denotes the set of variables appearing in formula B.

[4] The negation of a constraint C is $\neg C$, which can also be rewritten from C by switching the operator, e.g. $\neg(X > 5)$ is equivalent to $X \leq 5$.

revised answer arrives. We will use *derivation trees* to illustrate the steps. In a derivation tree, each node represents a state and is labelled with its local goals. The children of each node represent the new states generated from it after a reduction/revision step. Let $Q@S \leftarrow A$ be a default or returned answer, and "$\forall \boldsymbol{X}. \leftarrow A$" be its *negated answer*, where $\boldsymbol{X} = vars(A) \setminus vars(Q)$.

Positive Reduction: Let $Q@S$ (S is ground) be the positive goal selected from an active state $St = \langle \{Q@S\} \cup \mathcal{G}, \Delta, \mathcal{C}, \mathcal{N}, WA, AA \rangle$, $Q@S$ is sent to agent S as a query if not already sent, and then (see Figure 2):

1. If no answer has arrived yet, and there exist default answers, St is replaced by a set of new states, each of which uses a default answer entry and hence has new goals $\{A_d\} \cup \mathcal{G}$ and new assumed answers $\{(AID_d, Q@S)\} \cup AA$, where AID_d and A_d are the answer entry ID and answer respectively. Otherwise, St is replaced by a set of new states, each of which uses a returned answer entry with ID AID_{rtn} and answer A_{rtn} in a similar way.
2. An *original* choice state with new goals $\mathcal{G}$ and waiting answers WA=$\{(AID, Q@S)\}$ is created, where AID is the PSO answer entry ID for $Q@S$ if defaults are used (i.e. speculation exists), and is the PNO answer entry ID otherwise (i.e. no speculation).

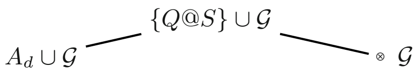


Fig. 2. Reduce $Q@S$ positively using defaults (assuming there is only one): let $Q@S \leftarrow A_d$ be the default answer, and let $\otimes$ denote a choice state

First Answer and Alternative Answers. Suppose defaults are used at the reduction of $Q@S$, and suppose the first answer $Q@S \leftarrow A_f$ has just arrived. One way of revising the current computation is to create a new active state, which make use of A_f, from the original choice state, and then to discard all the states that are using a default of $Q@S$. However, this means that all the computations completed after the reduction will be lost. The advantage of speculative reasoning arises from the attempt to reuse the previous computation as much as possible, as follows (see Figure 3):

1. [*Answer Revision*] for each state St_d using a default answer entry of $Q@S$ with ID AID_d:
   (a) a new state is created with A_f added to the goals and with $(AID_d, Q@S)$ replaced by $(AID_f, Q@S)$ in AA, where AID_f is the answer entry ID for the first answer;
   (b) St_d is updated by moving $(AID_d, Q@S)$ from AA to WA (i.e. it becomes a choice state);
2. [*Maintain Completeness*] a new active state is created from the original choice state, with $(AID_f, Q@S)$ added to AA, and with $\{A_f\} \cup \mathcal{G}_{rev}$ added to the goals,

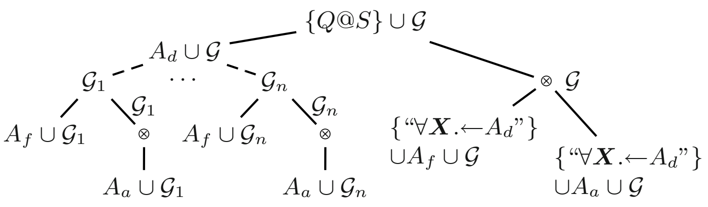
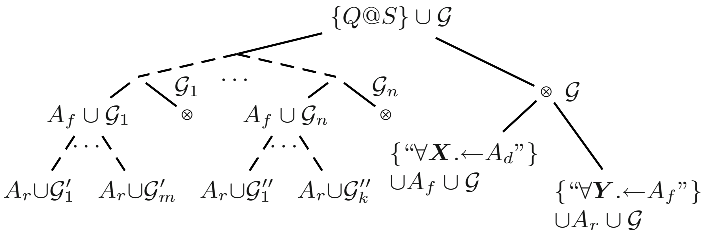


Fig. 3. New answers arrive: let A_f and A_a be the first and second (i.e. alternative) returned answer of $Q@S$ respectively, $\mathcal{G}_1, \ldots, \mathcal{G}_n$ be the goals of n states derived from $A_d \cup \mathcal{G}$, and $\boldsymbol{X} = vars(A_d) \backslash vars(Q)$



$\{Q@S\} \cup \mathcal{G}$
$\cdots$
$\mathcal{G}_1$
$\otimes$
$\mathcal{G}_n$
$\otimes$
$\otimes\ \mathcal{G}$
$A_f \cup \mathcal{G}_1$
$A_f \cup \mathcal{G}_n$
$A_r \cup \mathcal{G}'_1$
$A_r \cup \mathcal{G}'_m$
$A_r \cup \mathcal{G}''_1$
$A_r \cup \mathcal{G}''_k$
$\{“\forall \boldsymbol{X}.\leftarrow A_d”\} \cup A_f \cup \mathcal{G}$
$\{“\forall \boldsymbol{Y}.\leftarrow A_f”\} \cup A_r \cup \mathcal{G}$

Fig. 4. Revised answer arrives: let A_r be the revised answer for A_f, and $\boldsymbol{Y} = vars(A_f) \backslash vars(Q)$, let $\mathcal{G}'_1, \ldots, \mathcal{G}'_m$ (and $\mathcal{G}''_1, \ldots, \mathcal{G}''_k$) be the goals of m (and k) states derived from $A_f \cup \mathcal{G}_1$ (and $A_f \cup \mathcal{G}_n$)

where $\mathcal{G}_{rev}$ is the set of all the negated default answers (e.g. it is $\{“\forall \boldsymbol{X}. \leftarrow A_d”\}$ in Figure 3, assuming there is only one default).

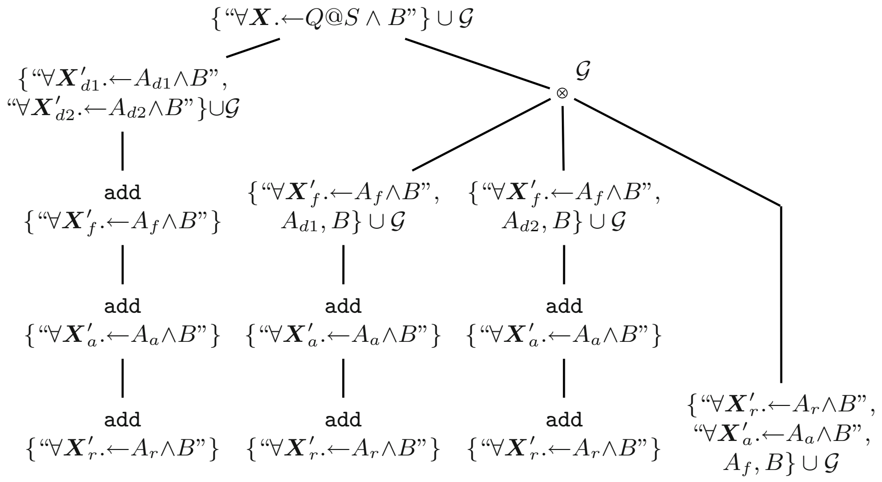
For any subsequently arriving alternative answer, we repeat step 1(a) and (2) only (see Figure 3). Note that if no default is used at the reduction, then in step (2) only A_f (or A_a) is added to the goals. If defaults are used at the reduction (i.e. default answers are available but no returned answer at the time of reduction), then $\mathcal{G}_{rev}$ is added to the new state in the attempt to avoid the re-computation of the search space covered by the states using defaults.

Revised Answer. Suppose a revised answer[5] $Q@S \leftarrow A_r$ arrives for the previous answer $Q@S \leftarrow A_f$. Instead of simply discarding the computations using A_f, we do the following (see Figure 4):

1. [*Answer Revision*] for each state using A_f, we add A_r to its goals and update its AA;
2. [*Maintain Completeness*] a new active state is created from the original choice state, with $(AID_r, Q@S)$ added to AA and with $\{A_r\} \cup \mathcal{G}_{rev}$ added to the goals, where $\mathcal{G}_{rev}$ is the negated previous answer (e.g. it is $\{“\forall \boldsymbol{Y}. \leftarrow A_f”\}$ in Figure 4).

[5] Revised answers can be sent by an internal agent, e.g. a finished state is revised and succeeds again.

Handling Askable in a Failure Goal. An askable in a failure goal can be reduced if none of its variables is universally quantified. Defaults can also be used at the reduction step. However, the first answer, alternative answers, and revised answers must be handled carefully. Figure 5 illustrates these important steps. Let $F = \forall \boldsymbol{X}. \leftarrow Q@S \wedge B$ be a failure goal and $Q@S \leftarrow A$ be an answer for $Q@S$, then $\forall \boldsymbol{X}'. \leftarrow A \wedge B$ is the resolvent of F with answer A, where $\boldsymbol{X}' = \boldsymbol{X} \cup vars(A)$.



$\{$"$\forall \boldsymbol{X}.\leftarrow Q@S \wedge B$"$\} \cup \mathcal{G}$
$\{$"$\forall \boldsymbol{X}'_{d1}.\leftarrow A_{d1} \wedge B$", "$\forall \boldsymbol{X}'_{d2}.\leftarrow A_{d2} \wedge B$"$\} \cup \mathcal{G}$
$\mathcal{G}$
$\otimes$
add $\{$"$\forall \boldsymbol{X}'_f.\leftarrow A_f \wedge B$"$\}$
$\{$"$\forall \boldsymbol{X}'_f.\leftarrow A_f \wedge B$", $A_{d1}, B\} \cup \mathcal{G}$
$\{$"$\forall \boldsymbol{X}'_f.\leftarrow A_f \wedge B$", $A_{d2}, B\} \cup \mathcal{G}$
add $\{$"$\forall \boldsymbol{X}'_a.\leftarrow A_a \wedge B$"$\}$
add $\{$"$\forall \boldsymbol{X}'_a.\leftarrow A_a \wedge B$"$\}$
add $\{$"$\forall \boldsymbol{X}'_a.\leftarrow A_a \wedge B$"$\}$
add $\{$"$\forall \boldsymbol{X}'_r.\leftarrow A_r \wedge B$"$\}$
add $\{$"$\forall \boldsymbol{X}'_r.\leftarrow A_r \wedge B$"$\}$
add $\{$"$\forall \boldsymbol{X}'_r.\leftarrow A_r \wedge B$"$\}$
$\{$"$\forall \boldsymbol{X}'_r.\leftarrow A_r \wedge B$", "$\forall \boldsymbol{X}'_a.\leftarrow A_a \wedge B$", $A_f, B\} \cup \mathcal{G}$

Fig. 5. Handle $Q@S$ negatively using defaults (assuming there are two): let A_{d1}, A_{d2}, A_f, and A_a be the two default answers, the first answer and an alternative answer for $Q@S$, respectively, and A_r revises A_f.

Negative Reduction Let $F =$ "$\forall \boldsymbol{X}.\leftarrow Q@S \wedge B$" be the failure goal selected from state $St = \langle(\{F\} \cup \mathcal{G}), \Delta, \mathcal{C}, \mathcal{N}, WA, AA\rangle$, $Q@S$ is sent to agent S as a query if not already sent, and then (see Figure 5):

1. a *negative original* state is created with new goals $\mathcal{G}$ and waiting answers $\{(AID_{no}, F)\}$, where AID_{no} is the NO answer entry ID for $Q@S$;
2. St is replaced by a state with new goals $\mathcal{G}_{add} \cup \mathcal{G}$ and with new assumed answers $\{(AID, F)\} \cup AA$, such that:
   - if no answer has arrived yet and there exist default answers, then $\mathcal{G}_{add}$ is the set of resolvents of F with the defaults; otherwise $\mathcal{G}_{add}$ is the set of resolvents with returned answers.
   - AID is the NS answer entry ID if defaults are used, and is the NN answer entry ID otherwise.

First Answer, Alternative Answers, and Revised Answers. Suppose the first answer $Q@S \leftarrow A_f$ arrives. Let R_f be the resolvent of the failure goal F with A_f. We do the following:

1. [*Maintain Completeness*] for each default answer A_d, a new active state is created from the negative original state, with (AID_{nn}, F) added to AA and with $\{R_f, A_d, B\}$ added to the goals (e.g. $\{R_f, A_{d_i}, B\}$ for the ith default A_{d_i} is $\{$"$\forall \boldsymbol{X}'_f. \leftarrow A_f \wedge B$", $A_{d_i}, B\}$ in Figure 5);
2. [*Answer Revision*] for each state assuming the NS answer for $Q@S$, it is updated by having R_f added to the goals and replacing (AID_{ns}, F) with (AID_{nn}, F) in AA, where AID_{ns} and AID_{nn} are the NS and NN answer entries, respectively.

Note that in Step 1 (Maintain Completeness), by letting each new active state contain one default A_d, we avoid the re-computation of the search space covered by the existing states that are using the defaults negatively.

For every alternative answer $Q@S \leftarrow A_a$ arriving subsequently, we just add R_a to the goals of every state that is assuming the NN answer, where R_a is the resolvent of F with A_a.

Suppose a revised answer $Q@S \leftarrow A_r$ arrives for the previous answer $Q@S \leftarrow A_f$. We (1) update the states like we do for A_a, and (2) create a new active state from the negative original state, with (AID_{nn}, F) added to AA and with $\mathcal{R} \cup \{A_f, B\}$ added to the goals, where $\mathcal{R}$ is the set of the resolvents of F with all the returned answers except A_f.

4.2 Example Trace

Here we use a (simplified) execution trace of Example 1 to illustrate the answer revision process. Suppose that **agent** l initiated the computation with a global goal $\mathcal{G}_0 = \{deliver(1, 2, Cost, Mtd)\}$:

1. **agent** l: the initial state with $\mathcal{G}_0$ was created. After a few local reduction steps (by Rules 1 and 3), the only pending goal was $quote(1, 2, Cost, Mtd)@d$. One default $quote(Src, Des, 5, air)@d$ was available and the askable was reduced positively – the goal was sent to **agent** d and two states were created: a finished state assuming the default and with constraints $Cost = 5 \wedge Mtd = air$ (i.e. the final answer), and a PSO choice state waiting for the askable's PSO answer.
2. **agent** d: similarly, local inference was performed for query $quote(1, 2, Cost, Mtd)$ (with Rules 1–3,5). Two answers were returned: "$quote(1, 2, Cost, Mtd)@d \leftarrow Cost{=}4, Mtd{=}air, safe\_weather(1), safe\_weather(2)$" and "$quote(1, 2, Cost, Mtd)@d \leftarrow Cost{=}6, Mtd{=}land$".
3. **agent** l: suppose that the two answers by **agent** d arrived in order,
   – revisions for the first answer: (i) a new choice state was created from the finished state, waiting for alternative answers; (ii) the first answer's body literals were added as the goals to the finished state (i.e. it became *active*). However, the new goal $Cost = 4$ is inconsistent with $Cost = 5$ (from the default), and hence the state was deleted (so was the previous answer extracted from it); (iii) a new active state with the first answer's body literals as goals was created from the PSO choice state. Subsequently, two askables $storm(1)@s'$ and $storm(2)@s'$ were derived and negatively reduced (also sent to agent s'). The same default answer was used twice, with two NO choice states created and two new goals $\{$"$\forall Area. \leftarrow 1 = Area, Area = 3$", "$\forall Area. \leftarrow 2 = Area, Area = 3$"$\}$ derived. The failure goals trivially hold, and hence a new final answer $Cost = 4, Mtd = air$ was obtained.

   - for the second answer, two revisions similar to (ii) and (iii) took place, and another new final answer $Cost$=6, Mtd=$land$ was obtained.
4. suppose an answer $storm(2)@s' \leftarrow true$ arrived at **agent** l: (a) the finished state in (iii) was revised with new goals $\{$"$\forall Area. \leftarrow 1 = Area, Area = 2$", "$\forall Area. \leftarrow 2 = Area, Area = 2$"$\}$. The second goal cannot hold, so the final state and its answer were removed; (b) two new states created from the two NO choice states in (iii), one with new goals $\{$"$\forall Area. \leftarrow 1 = Area, Area = 2$"$, 1 = 3\}$ and the other with $\{$"$\forall Area. \leftarrow 2 = Area, Area = 2$"$, 2 = 3\}$. Both of them had unsatisfiable goals and hence were removed eventually.

Thus, the only final answer for $deliver(1, 2, Cost, Mtd)$ is $Cost = 6, Mtd = land$.

4.3 Correctness

Our operational model is sound and complete with respect to the three-valued semantics [17] only, mainly because the adopted top-down abductive proof procedure cannot detect and handle loops occurring in an agent's local specification. However, the results can be strengthen to the two-valued semantics, if the logic program formed from the agent's specification satisfies certain conditions, such as "overall consistent" and "abductive acyclic"[18].

Theorem 1. (Soundness of Local Inference) *Given a query $Q@S$ for an internal agent S with current belief state $BEL(T, \mathcal{R}) = \langle \Pi^{bel}, \mathcal{AB}, \mathcal{I} \rangle$, if there exists a finished state St_f computed by S for $Q@S$, and $Ans = \langle \Delta, \mathcal{C} \rangle$ is extracted from St_f, then Ans is a local abductive answer for $Q@S$, such that let the* interpretation of $\mathcal{AB}$ *be $I_{\Delta\theta} = \{A^t | A \in \Delta\theta \wedge A \in \mathcal{AB}\} \cup \{A^f | A \notin \Delta\theta \wedge A \in \mathcal{AB}\}$ and θ be the variable substitutions induced by $\mathcal{C}$:*

- *$\Pi^{bel} \cup I_{\Delta\theta} \models_{3comp} Q\theta$, and*
- *$\Pi^{bel} \cup I_{\Delta\theta} \models_{3comp} \mathcal{I}$*

where $\models_{3comp}$ is the logical entailment under the three-valued semantics for abductive logic programs [17].

Theorem 2. (Completeness of Local Inference) *Given a query $Q@S$ for an internal agent S with current belief state $BEL(T, \mathcal{R})$. If the local inference finishes (i.e. no more active state):*

- *if there exists no finished state, then there is no local abductive answer for $Q@S$ w.r.t. $BEL(T, \mathcal{R})$;*
- *if there is a local abductive answer for $Q@S$ w.r.t. $BEL(T, \mathcal{R})$, then there must exist a finished state, in which the answer can be extracted.*

The proofs for Theorems 1 and 2 are based on the correctness (both soundness and completeness) of each inference rule. The correctness for rules reducing non-askables (i.e. Rule 1–4, 6–9) are proven in [18]. To show the correctness of Rule 5 and Rule 10, we need to show that, after handling a returned answer, the search space (represented by the active states and the finished states) for the query w.r.t. the agent's old belief state is transformed into the search space w.r.t. the agent's new belief state, similar to the proofs in [2]. We omit proofs here in favour of a more detailed discussion of the overall summary of the abductive speculative process.

Assuming that no query or answer is lost during communications, Theorem 2 guarantees that each internal agent eventually receives all the internal answers from its children, and hence reaches a stable state. This further implies that the whole system will become stable eventually. Then, applying Theorems 1 and 2 to the root agent's local inferences gives the **correctness of global inference**.

4.4 Implementation

A system prototype has been implemented in YAP Prolog [6]. In the prototype, agents run on different hosts of a local network and exchange queries/answers through TCP messages. Each agent maintains its own answer entry store, and has a multi-threaded reasoning engine executing the inference procedure: each active state is processed by a *worker thread*, and the finished and choice states are managed by a persistent *manager thread*, which revises the finished/choice states and notifies relevant worker threads upon receiving an answer. We have run Example 1 with increased complexity (e.g., more agents and more rules), and details are available from the authors.

5 Discussion

Reuse of Computation: One of the main advantages of speculative computation is the possibility of reusing parts of the computation already done when new answers arrive and answer revision is required. If the new answer is consistent with the existing one (or with the default in the case of the first answer), then all of the previous computation can be reused. On the other hand, if the new answer is inconsistent with the existing one (or with the default), then none of the existing computation can be reused. In this case, however, the computation between the use of the defaults and the first new answer, consumes CPU cycles of the agent only, which would be in idle if no defaults were used. The computation between the use of a returned answer and its new revising answer is unavoidable as the agent has no idea of whether or when the revising answer would arrive. In either case, the main overhead is the extra computation required to decide, during answer revision, whether the new and existing answers are consistent. A more detailed study on the impact that reuse of computations has over computational time will be subject of our future work.

Agent Hierarchy: The problem of *cyclic answer dependency* may arise if agents are allowed to arbitrarily send queries to each other, e.g. the answer A_1 computed by agent a for a query $Q_1@a$ depends on an answer A_2 computed by b for the query $Q_2@b$, but the computation of A_2 assumes A_1. This is usually caused by cycles in the union of the agents' background knowledge (as in normal logic programs). Enforcing an agent hierarchy can eliminate such problem. This requirement can be removed if we record and propagate the answer usages during agent collaboration, as in a variety of methods sometimes called dependency-directed backtracking. However, in doing so, there is a trade-off between efficiency and flexibility.

[6] http://www.dcc.fc.up.pt/~vsc/Yap/

The Growth of States: Let an askable be reduced P times positively among all the computations. The revision for each new answer creates $P + Q$ new states and revises Q existing states, where Q is the number of states using a default answer. The revision for each revised answer creates P new states and revises O existing states, where O is the number of states using the previous answer. On the other hand, let an askable be reduced N times negatively among the computations. The revision for each new answer creates $N \times D$ new states and revises M existing states, where D is the number of available default answers and M is the number of states using the defaults negatively. The revision of each revised answer creates N new states and revises K states, where K is the number of states using returned answers negatively. In practice, the actual increase of total states is much smaller as the new states created and the existing states revised often have complementary goals.

Usage of Default Answers: Similar to the situation with growth of states during answer revision, the quality and the number of available defaults may impact the computational performance: so the generation of quality defaults is important. In addition, in some applications it may be better to ignore defaults even if they are available. For example, if a travel booking system will charge a penalty fee for any plan being revised, then the agent speculatively computing a plans should try to use as few defaults as possible. Hence in practice, agents should be allowed to decide whether defaults are used during local inference, using some application-dependent heuristics. Note that the notion of an agent, or set of agents, using an application-dependent economic or utility model would be appropriate here.

6 Conclusion

In practice, exogenous knowledge will exist and will influence multi-agent reasoning. The speculative abductive reasoning framework presented here allows agents in a hierarchical system to efficiently handle incomplete and revisable conditional answers returned by others. Our operational model supports negation and constraints, and hence can be applied to a wide range of real world applications.

Good default answers can increase the benefits provided by speculative computation, such as more highly parallel reasoning among the agents. In addition, it is important to have flexible control of default use. As future work, we intend to use the existing prototype with more complex examples, in order to investigate the scalability of the system, and try to find good heuristics for controlling the usage of defaults.

Acknowledgment

This research was partially supported by the Ministry of Education, Science, Sports and Culture, Japan, Grant-in-Aid for Scientific Research (B), 19300053, and is continuing through participation in the International Technology Alliance sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence.

References

1. Burton, F.W.: Speculative computation, parallelism, and functional programming. IEEE Trans. Computers 34(12), 1190–1193 (1985)
2. Ceberio, M., Hosobe, H., Satoh, K.: Speculative constraint processing with iterative revision for disjunctive answers. In: Toni, F., Torroni, P. (eds.) CLIMA 2005. LNCS (LNAI), vol. 3900, pp. 340–357. Springer, Heidelberg (2006)
3. Ciampolini, A., Lamma, E., Mello, P., Toni, F., Torroni, P.: Cooperation and competition in alias: a logic framework for agents that negotiate. Annals of Math. and AI, 65–91 (2003)
4. Gregory, S.: Experiments with speculative parallelism in parlog. In: ILPS 1993: Proceedings of the 1993 International Symposium on Logic Programming, pp. 370–387. MIT Press, Cambridge (1993)
5. Hayashi, H.: Replanning in robotics by dynamic SLDNF. In: Proc. Workshop on Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World (1999)
6. 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)
7. Hosobe, H., Satoh, K., Ma, J., Russo, A., Broda, K.: Speculative constraint processing for hierarchical agents. In: Proc. of 7th European Workshop on MAS, EUMAS 2009 (2009)
8. Jaffar, J., Maher, M.J., Marriott, K., Stuckey, P.J.: The semantics of constraint logic programs. J. Log. Prog., 1–46 (1998)
9. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive logic programming. Journal of Logic and Comp. 2(6), 719–770 (1992)
10. Lloyd, J.W.: Foundations of logic programming, 2nd extended edn. Springer, New York (1987)
11. Ma, J., Russo, A., Broda, K., Clark, K.: DARE: a system for distributed abductive reasoning. Autonomous Agents and Multi-Agent Systems 16(3), 271–297 (2008)
12. Nabeshima, H., Iwanuma, K., Inoue, K.: Effective sat planning by speculative computation. In: Proc. of the 15th Australian Joint Conf. on AI, pp. 726–728. Springer, London (2002)
13. Satoh, K.: Speculative computation and abduction for an autonomous agent. IEICE Transactions 88-D(9), 2031–2038 (2005)
14. Satoh, K., Inoue, K., Iwanuma, K., Sakama, C.: Speculative computation by abduction under incomplete communication environments. In: ICMAS, pp. 263–270 (2000)
15. Satoh, K., Yamamoto, K.: Speculative computation with multi-agent belief revision. In: AAMAS, pp. 897–904 (2002)
16. Shanahan, M.: Reinventing shakey. In: Logic-based artificial intelligence, pp. 233–253 (2000)
17. Teusink, F.: Three-valued completion for abductive logic programs. In: Proc. of the 4th Int. Conf. on Algebraic and Logic Programming, pp. 171–200. Elsevier Science Inc., New York (1996)
18. Van Nuffelen, B.: Abductive constraint logic programming: implementation and applications, Phd, Department of Computer Science, K.U.Leuven, Belgium (June 2004)
19. Wallace, M.: Negation by constraints: A sound and efficient implementation of negation in deductive databases. In: SLP, pp. 253–263 (1987)
20. Clark, K.: 1Negation as failure. In: Logic and Data Bases, pp. 293–322 (1978)

Formal Semantics of a Dynamic Epistemic Logic for Describing Knowledge Properties of π-Calculus Processes

Pedro Arturo Góngora[1], Eric Ufferman[2], and Francisco Hernández-Quiroz[2]

[1] Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas
Universidad Nacional Autónoma de México
Circuito escolar, Ciudad Universitaria, C.P. 04510, México D.F., México
pedro.gongora@gmail.com

[2] Departamento de Matemáticas, Facultad de Ciencias
Universidad Nacional Autónoma de México
Circuito exterior, Ciudad Universitaria, C.P. 04510, México D.F., México
ufferman@gmail.com, fhq@ciencias.unam.mx

Abstract. The π-calculus process algebra describes the interaction of concurrent and communicating processes. The π-calculus, however, has neither explicit agency nor epistemic capabilities. In this paper, we present the formal syntax and semantics of a multi-agent dynamic epistemic logic. In this logic, the epistemic actions of agents are π-calculus processes. A process of the language is translated to a class of model updating functions reflecting the epistemic changes after the execution of such processes. Our proposal combines the capabilities of two approaches: it is possible to model structured interaction among agents as elaborated π-calculus programs, and it is also possible to describe the dynamic knowledge implications of such programs. We show the utility of our language by encoding the Dining Cryptographers protocol.

1 Introduction

On the one hand, multi-agent dynamic epistemic logic describes and reasons about the epistemic dynamics in a multi-agent setting. On the other hand, π-calculus process algebra describes and reasons about the dynamics of concurrent and communicating systems. By using dynamic epistemic logic, we can represent communicating actions as instantaneous events, but without describing the internal dynamics of such events. By contrast, with π-calculus we can finely describe the dynamics of concurrent and communicating systems. In this paper, our goal is to combine both approaches into a single epistemic logic. Specifically, we introduce the syntax and semantics for a multi-agent dynamic epistemic logic that uses π-calculus processes as the actions of the language.

For the Action Model Logic of [1], the authors propose using relational structures for describing communication events. By using such relational structures, it is possible to capture many forms of communication: public announcement, private communication, private communication with outsiders, etc. Nevertheless, the internal dynamics of the communication cannot be described.

By contrast, with π-calculus we can finely describe the dynamics of concurrent and communicating processes, but not the epistemic updates within the execution of such processes. The method we propose translates a π-calculus process into a class of model updating functions. These functions are sequential compositions of epistemic updates, reflecting the interactions of the concurrent processes. We will build upon the semantic framework of [1] by using relational structures for describing these epistemic updates, and the π-calculus abstract machine of [8] for doing the translation.

The organization of the paper is as follows. Section 2 presents the syntax and semantics for the dynamic epistemic logic of [1], and also introduces the semantic framework that we will use in the rest of the paper. Section 3 surveys the formal syntax and semantics of the π-calculus. Sections 4 and 5 contain our main contribution. Section 4 is devoted to present the formal syntax and semantics of the dynamic epistemic logic $\mathcal{L}_{E\pi}$, that uses π-calculus for describing epistemic actions. In Sect. 5 we present a codification of the Dining Cryptographers protocol [3] in $\mathcal{L}_{E\pi}$ as an example of its usefulness.

2 Logics for Epistemic Dynamics

In this section, we will briefly introduce the syntax and semantics of the dynamic epistemic logic of [1]. The language presented here is one of the many approaches to epistemic dynamics. For a more thorough introduction to this language, and others as well, we direct the reader to [4].

We first introduce the syntax and semantics of basic multi-agent epistemic logic. This logic is an instance of propositional multi-modal logic. In this particular instance, the modalities K_i are interpreted as "agent i knows".

Definition 1 (Multi-Agent Epistemic Logic Syntax). *Let $\mathcal{P}$ be a countable set of atomic proposition symbols and $\mathcal{A}$ be a finite set of agents. The set $\mathcal{L}_{EL}$ of all formulas of Multi-Agent Epistemic Logic is the least set generated by the grammar:*

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_i\varphi$$

where $p \in \mathcal{P}$ and $i \in \mathcal{A}$.

The models we will use for all the languages presented in this paper are the usual *Kripke* or *possible-world* models. In such models, it is assumed that in addition to the actual state of affairs (valuations of propositions), there are other possible worlds that the agents may consider possible. Then, we say an agent i *knows* some proposition p (viz. K_ip), if such proposition holds on every world that i considers possible from the actual world.

Definition 2 (Models and Satisfaction). *A model $\mathfrak{M}$ for the formulas in $\mathcal{L}_{EL}$ is the tuple:*

$$\mathfrak{M} \overset{\text{def}}{=} \langle W, \{R_i\}_{i\in\mathcal{A}}, V\rangle$$

where W is a finite set of possible worlds, each $R_i \subseteq (W \times W)$ is the accessibility relation of agent i, and $V : (W \times \mathcal{P}) \to \{\mathbf{T}, \mathbf{F}\}$ is a valuation for atomic propositions. A pointed model is a pair $(\mathfrak{M}, w)$ where w is a possible world in $\mathfrak{M}$. We denote as $\mathbf{K}$ the class of all models and as $\overrightarrow{\mathbf{K}}$ the class of all pointed models. The satisfaction relation $\models$ between pointed models and formulas in $\mathcal{L}_{EL}$ is the least relation such that:

1. $(\mathfrak{M}, w) \models p$ *iff* $V(w, p) = \mathbf{T}$ $(p \in \mathcal{P})$;
2. $(\mathfrak{M}, w) \models \neg\varphi$ *iff* $(\mathfrak{M}, w) \not\models \varphi$;
3. $(\mathfrak{M}, w) \models (\varphi \wedge \psi)$ *iff* $(\mathfrak{M}, w) \models \varphi$ *and also* $(\mathfrak{M}, w) \models \psi$;
4. $(\mathfrak{M}, w) \models K_i\varphi$ *iff for all* $w' \in W$, $(w, w') \in R_i$, *implies* $(\mathfrak{M}, w') \models \varphi$.

The logical connectives $\neg$ and $\wedge$ preserve their intuitive meaning of *negation* and *conjunction*, respectively. For any agent i and formula φ, the formula $K_i\varphi$ (i knows that φ) holds only when φ is true at every world that i considers possible (by R_i) from the actual world. Also, we can define the other logical connectives ($\vee$, $\Rightarrow$ and $\Leftrightarrow$) in terms of $\neg$ and $\wedge$ with the usual definitions. Finally, for some fixed $p \in \mathcal{P}$, we define the *true constant* $\top \overset{\text{def}}{=} (p \vee \neg p)$ and the *false constant* $\bot \overset{\text{def}}{=} \neg\top$, such that $(\mathfrak{M}, w) \models \top$ and $(\mathfrak{M}, w) \not\models \bot$ for any pointed model $(\mathfrak{M}, w)$.

For different purposes we may choose different axiomatizations. For example, the **S5** and **KD45** systems are common choices for describing idealized forms *knowledge* and *belief*, respectively (see [4]).

The basic epistemic language is expressive enough to describe higher order knowledge. For example, $K_iK_j\varphi$: "i knows that j knows", or i's introspection $K_i\varphi \Rightarrow K_iK_i\varphi$: "$i$ knows that she knows". Also, it is desirable to describe change in agents' knowledge as a result of *communication*. For describing dynamic knowledge, we extend the language with dynamic-logic-like modalities. The extended language has formulas like $[\alpha]\,\varphi$, standing for "φ holds after the communication event α took place".

In this paper, we build upon the approach by Baltag, Moss and Solecki [1], by using *action models* for describing communication events. An action model is a relational structure, the possible worlds stand for the possible epistemic updates, and the accessibility relations model the uncertainty of agents about such epistemic updates. We first present the definitions of action models and the *model product*. The model product is an operation for updating a model according to the communication event described by an action model.

Definition 3 (Action Model and Model Product). *Let $\mathcal{A}$ be a finite set of agents. An action model $\mathfrak{A}$ is the tuple:*

$$\mathfrak{A} \overset{\text{def}}{=} \langle E, \{\to_i\}_{i \in \mathcal{A}}, \textsc{Pre} \rangle$$

where E is a finite set of events, there is a binary accessibility relation $\to_i \subseteq E \times E$ for each agent $i \in \mathcal{A}$, and the total function $\textsc{Pre} : E \to \mathcal{L}_{EL}$ assigns precondition formulas to events. A pointed action model is a pair $(\mathfrak{A}, e)$, where e is an event in $\mathfrak{A}$. Let $\mathfrak{M} = \langle W, \{R_i\}_{i \in \mathcal{A}}, V \rangle$ be a model and $\mathfrak{A} = \langle E, \{\to_i\}_{i \in \mathcal{A}}, \textsc{Pre} \rangle$ be an action model. The product $\otimes$ between the pointed models $(\mathfrak{M}, w)$ and $(\mathfrak{A}, e)$ is defined as follows:

$$(\mathfrak{M}, w) \otimes (\mathfrak{A}, e) \stackrel{\text{def}}{=} (\mathfrak{M}', (w, e))$$

whenever $(\mathfrak{M}, w) \models \textsc{Pre}(e)$*, and where the model* $\mathfrak{M}' \stackrel{\text{def}}{=} \langle W', \{R'_i\}_{i \in \mathcal{A}}, V' \rangle$ *consist of:*

– $W' \stackrel{\text{def}}{=} \{(u, f) \mid u \in W, f \in E,$ *and* $(\mathfrak{M}, u) \models \textsc{Pre}(f)\}$*;*
– $R'_i \stackrel{\text{def}}{=} \{((u, f), (u', f')) \mid (u, u') \in R_i$ *and* $(f, f') \in \rightarrow_i\}$*;*
– $V'((u, f), p) \stackrel{\text{def}}{=} V(u, p)$ *for all* $u \in W$*,* $f \in E$*, and* $p \in \mathcal{P}$*.*

We extend the basic epistemic language with a new construct $[(\mathfrak{A}, e)]\varphi$. This new formula holds whenever the updated model, according to the product by $(\mathfrak{A}, e)$, satisfies φ. Next, we formally define the syntax and semantics of the extended language.

Definition 4 (Action Model Logic Syntax and Satisfaction). *Let* $\mathcal{P}$ *be a countable set of atomic proposition symbols and* $\mathcal{A}$ *be a finite set of agents. The set* $\mathcal{L}_{Act}$ *of all formulas of Action Model Logic is the least set generated by the grammar:*

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_i\varphi \mid [(\mathfrak{A}, e)]\,\varphi$$

where $p \in \mathcal{P}$*,* $i \in \mathcal{A}$*, and* $(\mathfrak{A}, e)$ *denotes a pointed action model. The satisfaction relation* $\models$ *between pointed models and formulas in* $\mathcal{L}_{Act}$ *is the least relation satisfying rules* 1 *to* 4 *of Def.* 2 *plus the following additional rule:*

5. $(\mathfrak{M}, w) \models [(\mathfrak{A}, e)]\varphi$ *iff* $(\mathfrak{M}, w) \models \textsc{Pre}(e)$ *implies* $((\mathfrak{M}, w) \otimes (\mathfrak{A}, e)) \models \varphi$*.*

Note that $(\mathfrak{M}, w) \otimes (\mathfrak{A}, e)$ is defined only when w satisfies e's precondition. Hence, in the other case, w vacuously satisfies any formula $[(\mathfrak{A}, e)]\varphi$.

Finally, to finish this section, consider the two action models depicted in Fig. 1. The action model in panel (a) models a *public announcement*. In this public announcement, the only possible epistemic update for every agent is that of φ being told. The action model in panel (b) models a *private communication* between agents 1 and 2. The agents 1 and 2 consider φ the only possible update, however, the other agents *believe* that no update occurred (or, equivalently, that the trivial update $\top$ took place).

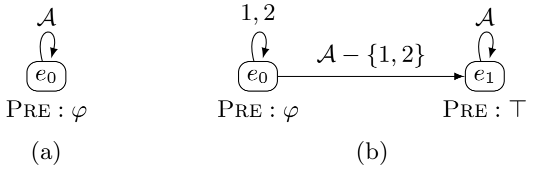


Fig. 1. (a) public announcement, (b) private communication

3 The π-Calculus

The π-*calculus* [10] is a process algebra for describing concurrent and communicating processes. In π-calculus processes, all communication is done through *channels* or *names*. For example, any process listening on some channel is capable of receiving any message sent through the same channel. A distinctive feature of π-calculus is that of *mobility*. Mobility refers to the processes capability to exchange communication channels. In this section, we will introduce the synchronous variant of the π-calculus. We refer the reader to [10] for more details on the material presented here.

Definition 5 (π-Calculus Syntax). *Let $\mathcal{N} = \{x, y, \ldots\}$ be an infinitely countable set of name symbols. The set of all processes of the π-calculus is the least set generated by the grammar:*

$$\begin{aligned} P &::= \mathbf{0} \mid \pi.P \mid \pi.P + \cdots + \pi.P \mid P|P \mid (\boldsymbol{\nu} x)\ P \mid\ !\pi.P \\ \pi &::= \overline{x}y \mid x(z) \mid \tau \mid \textbf{if}\ x = y\ \textbf{then}\ \pi \end{aligned}$$

where $x, y, z \in \mathcal{N}$ and $\tau, \boldsymbol{\nu} \notin \mathcal{N}$.

We will refer to the elements of $\mathcal{N}$ both as names and as channels. The basic entity of π-calculus is the *process*. The most elementary process is the *nil process* $\mathbf{0}$, standing for a finished computation. The *guarded* or *prefixed* process $\pi.P$ is capable of doing the action specified by π and then continuing as P. The *output* prefix $\overline{x}y$ represents the capability of sending y through x. The *input* prefix $x(z)$ represents the capability of receiving z through x. The *silent action* τ represents the capability of doing some internal computation. The *match* prefix $\textbf{if}\ x = y\ \textbf{then}\ \cdot$ is a guard that performs the check of whether x and y are the same name. The sum $\pi.P + \pi'.Q$ stands for the *non-deterministic choice* between prefixed processes $\pi.P$ and $\pi'.Q$. The process $P \mid Q$ stands for the *parallel composition* of processes P and Q. The construct $(\boldsymbol{\nu} z)\ P$ is a *name restriction*, indicating that every *free* occurrence of the name z in P (see Def. 6) is different from other z outside the scope of such restriction. The construct $!\pi.P$ is the *process replication*, stating that there are as many parallel copies of $\pi.P$ as needed.

Input and output prefixes represent complementary actions, and may interact when they are present in a parallel composition. For example, the parallel processes:

$$\overline{x}y.\mathbf{0} \mid x(z).\overline{w}z.\mathbf{0}$$

may interact, evolving to the new process:

$$\mathbf{0} \mid \overline{w}y.\mathbf{0}$$

Note that the output and input processes must use the same communication channel to interact with each other. Also, observe that in the input process $x(z).\overline{w}z.\mathbf{0}$ the name z is a placeholder. Hence, z has to be replaced with the actual name y received.

By using the sum operation we can model non-deterministic computations. For example, the process:

$$(\overline{x}a.P + \overline{x}b.Q) \mid x(z).R$$

may evolve into one of the following two processes:

$$P \mid R\{a/z\} \qquad\qquad Q \mid R\{b/z\}$$

where $R\{a/z\}$ is the syntactic substitution replacing all the free instances of z by a in the process R.

With parallel composition we can also introduce non-deterministic behavior:

$$\overline{x}a.P \mid \overline{x}b.Q \mid x(z).R$$

this process may evolve into one of the following two processes:

$$P \mid \overline{x}b.Q \mid R\{a/z\} \qquad\qquad \overline{x}a.P \mid Q \mid R\{b/z\}$$

The name restrictions confine the use of a name to a given process. For example, the following two parallel processes:

$$\overline{x}a.P \mid ((\boldsymbol{\nu}x)\ x(z).Q)$$

cannot interact. The restriction above indicates that the x in process $x(z).Q$ is not the same x in process $\overline{x}a.P$.

With name restrictions we can also specify more complex interactions, as the exchange of *private* communication channels. For example, consider the following process:

$$((\boldsymbol{\nu}a)\ \overline{x}a.P) \mid x(z).Q$$

In the previous processes, the restriction is on the message, not on the channel the processes use for communicating. Because of this, if the name a does not occur free in Q, then the processes can interact, and the scope of the restriction is *expanded*:

$$(\boldsymbol{\nu}a)\ (P \mid Q\{a/z\})$$

After this expansion, the name a may be used for further private communication.

We can use the process replication for describing repetitive computations. We can regard replication as satisfying the equation $!\pi.P = \pi.P \mid !\pi.P$. For example, the following processes interact infinitely often:

$$!\overline{x}y.\mathbf{0} \mid !x(z).\mathbf{0}$$

Formally, all of the above processes interactions are described by the notion of *reduction*. A reduction is a transition $P \longrightarrow P'$ stating that process P can be reduced to process P'. In order to define reductions, we need to formalize the notions of free names and α-equivalence.

Definition 6 (Free Names, α-Equivalence). *We say that a name z occurs free in a process P iff z occurs in P and it is not under the scope of an input prefix $x(z)$ or a name restriction $(\boldsymbol{\nu} z)$. We denote as $\texttt{fn}(P)$ the set of all the free names of process P. We say that two processes P and Q are α-equivalent ($P =_\alpha Q$), if Q can be obtained from P through a finite number of bound-name changes such that:*

1. *If a name w does not occur in P, then the process $P\{w/x\}$ is obtained by replacing with w each free occurrence of x in P.*
2. *A bound-name change is the exchange of a term $x(z).P$ or a term $(\boldsymbol{\nu} z)\ P$, with terms $x(w).(P\{w/z\})$ or $(\boldsymbol{\nu} w)\ (P\{w/z\})$, respectively, given that $w \notin \texttt{fn}(P)$.*

The α-equivalence indicates that, for example, the following two processes describe the same computation:

$$(\boldsymbol{\nu} y)\ (x(z).\overline{u}z.\mathbf{0} \mid u(s).\overline{s}y.\mathbf{0}) \qquad\qquad (\boldsymbol{\nu} z)\ (x(a).\overline{u}a.\mathbf{0} \mid u(b).\overline{b}z.\mathbf{0})$$

Finally, we state the formal semantics of the π-calculus with the following definition.

Definition 7 (Reductions). *The reduction transition relation $\longrightarrow$ between processes is the least relation defined according to the following rules:*

$$\textsc{Reduct-Struct}\ \frac{P \equiv P', Q \equiv Q', P \longrightarrow Q}{P' \longrightarrow Q'}$$

$$\textsc{Reduct-Inter}\ \frac{}{(\overline{x}y.P_1 + Q_1) \mid (x(z).P_2 + Q_2) \longrightarrow P_1 \mid P_2\{y/z\}}$$

$$\textsc{Reduct-Tau}\ \frac{}{\tau.P + Q \longrightarrow P}$$

$$\textsc{Reduct-Par}\ \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}$$

$$\textsc{Reduct-Res}\ \frac{P \longrightarrow P'}{(\boldsymbol{\nu} z)\ P \longrightarrow (\boldsymbol{\nu} z)\ P'}$$

where the structural congruence relation $\equiv$ between processes is the least symmetric relation satisfying:

1. *$P =_\alpha Q$ implies $P \equiv Q$;*
2. *$P + \mathbf{0} \equiv P$, $P + Q \equiv Q + P$, and $P + (Q + R) \equiv (P + Q) + R$;*
3. *$P \mid \mathbf{0} \equiv P$, $P \mid Q \equiv Q \mid P$, and $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$;*
4. *$(\mathbf{if}\ x = x\ \mathbf{then}\ P) \equiv P$;*
5. *$(\boldsymbol{\nu} z)\ \mathbf{0} \equiv \mathbf{0}$, $(\boldsymbol{\nu} z)\ (\boldsymbol{\nu} s)\ P \equiv (\boldsymbol{\nu} s)\ (\boldsymbol{\nu} z)\ P$, $(\boldsymbol{\nu} z)\ (P \mid Q) \equiv P \mid ((\boldsymbol{\nu} z)\ Q)$ if $z \notin \texttt{fn}(P)$, and $(\boldsymbol{\nu} z)\ \mathbf{if}\ x = y\ \mathbf{then}\ P \equiv \mathbf{if}\ x = y\ \mathbf{then}\ (\boldsymbol{\nu} z)\ P$ if $x, y \neq z$;*
6. *$!P \equiv P \mid !P$.*

The π-calculus is a rich language for describing concurrent and communicating computations. This language, however, is not able to express neither agent-based communication nor the epistemic-state changes based in such communication. In the next section we will propose a language for describing agency and epistemic updates with π-calculus processes.

4 Epistemic Dynamics with π-Calculus

In this section, we present a language that combines dynamic epistemic logic and π-calculus. Syntactically, this language is a dynamic epistemic logic such that the actions are π-calculus processes executed by agents. The processes of this language explicitly state the agent responsible for each primitive action.

The semantics of our language is based on the action models of [1] and on the π-calculus abstract machine of [8]. Briefly, the processes of this language are translated to sequential compositions of *model updating functions*. This model updating functions are applications of the model product by some predefined action models. Such action models define the calculus communication nature. In this case, the communication is the private and synchronous message exchanging between a sender and a receiver. Also, because of the non-deterministic nature of π-calculus, the meaning of a process is a class of this sequential compositions, as each process may evolve with several possible executions.

Definition 8 (Dynamic Epistemic Logic with Processes Syntax). *Let $\mathcal{P}$ be a countable set of atomic proposition symbols, $\mathcal{A}$ be a finite set of agents, and $\mathcal{N}$ be an infinitely countable set of name symbols. The set $\mathcal{L}_{E\pi}$ of all formulas of Dynamic Epistemic Logic with Processes is the least set generated by the grammar:*

$$\begin{aligned} \varphi &::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_i\varphi \mid [P]\,\varphi \\ P &::= \mathbf{0} \mid \pi.P \mid \pi.P + \cdots + \pi.P \mid P|P \mid (\boldsymbol{\nu}x)\ P \mid !\pi.P \\ \pi &::= \overline{x}_i y \mid \overline{x}_i \varphi \mid x_i(z) \mid \tau \mid \textbf{if}\ \varphi\ \textbf{then}\ \pi \end{aligned}$$

where $p \in \mathcal{P}$, $i \in \mathcal{A}$, $x, y, z \in \mathcal{N}$ and $\tau, \boldsymbol{\nu} \notin \mathcal{N}$.

The basic structure of the language logical part is the same as the dynamic epistemic logic presented in Sect. 2. Also, all of the process constructs preserve their intuitive meaning, as described in the previous section.

There are, however, three main differences between the processes of this language and the processes of the pure π-calculus. The first difference is the specification of the agent executing the action in the input and output prefixes. For example, the process $x_i(z).P$ states that agent i is expecting to receive a message z through channel x, and then, after receiving such a message, the execution continues as process P. The second difference is the ability to send formulas as well as names. With this addition, we preserve the mobility capabilities of the calculus plus the ability for updating the agents' epistemic state. Finally, the last difference is that we parametrize the match prefix with arbitrary formulas. This addition allows modeling richer guards for processes.

The semantics for the static part of this language is not different to that of basic epistemic logic. For the dynamic part, the semantics is built upon three main components: model updating functions, process environments and a process interpretation relation.

Definition 9 (Action Models for Processes and Model Updating Functions). *Given a formula $\varphi \in \mathcal{L}_{E\pi}$ and two agents, a sender s and a receiver r, we define the action model $\mathfrak{A}^{\varphi}_{s,r}$ as the tuple:*

$$\mathfrak{A}^{\varphi}_{s,r} \overset{\text{def}}{=} \langle E, \{\rightarrow_i\}_{i \in \mathcal{A}}, \textsc{Pre} \rangle$$

where:

$$\begin{aligned} E &\overset{\text{def}}{=} \{e_0, e_1\} & \rightarrow_s &\overset{\text{def}}{=} \{(e_0,e_0),(e_1,e_1)\} \\ \textsc{Pre}(e_0) &\overset{\text{def}}{=} K_s\varphi & \rightarrow_r &\overset{\text{def}}{=} \{(e_0,e_0),(e_1,e_1)\} \\ \textsc{Pre}(e_1) &\overset{\text{def}}{=} \top & \rightarrow_i &\overset{\text{def}}{=} \{(e_0,e_1),(e_1,e_1)\} \quad (i \in \mathcal{A} - \{s,r\}) \end{aligned}$$

A model updating function is a function $f : \overrightarrow{\mathbf{K}} \rightarrow \overrightarrow{\mathbf{K}}$. We define the model updating functions id and com as the following λ-terms:

$$\begin{aligned} id &\overset{\text{def}}{=} \lambda M.M \\ com(s,r,\eta) &\overset{\text{def}}{=} \begin{cases} id & \text{if } \eta \in \mathcal{N} \\ \lambda M.\left(M \otimes (\mathfrak{A}^{\eta}_{s,r}, e)\right) & \text{otherwise} \end{cases} \end{aligned}$$

where η is either a name or a formula, and s and r are agents (a sender and a receiver), and the variable M stands for any pointed model.

These model updating functions represent the primitive epistemic updates caused by process reductions. The model updating function *com* describes a reduction involving the exchange of a formula, and the function *id* describes the other cases (where there are no epistemic updates). As we are using synchronous π-calculus, the action model $\mathfrak{A}^{\varphi}_{s,r}$ represents the synchronous communication between sender s and receiver r (cf. panel (b) in Fig. 1). Note that this communication is also private. This is because communication occurs through a channel shared only between the sender and the receiver at the time of the message exchange.

Definition 10 (Process Environments). *We define a process environment as a pair (ρ, X), where ρ is a multiset of (prefixed-) process summations and the set $X \subseteq \mathcal{N}$ contains the free names of the processes in ρ. We define the insertion operation $\oplus$ for inserting processes into process environments as follows:*

$$\begin{aligned} \mathbf{0} \oplus (\rho, X) &\overset{\text{def}}{=} (\rho, X) \\ (P_1 + \cdots + P_k) \oplus (\rho, X) &\overset{\text{def}}{=} (\rho \uplus \{\!| P_1 + \cdots + P_k |\!\}, X \cup \mathtt{fn}(P_1 + \cdots + P_k)) \\ P \mid Q \oplus (\rho, X) &\overset{\text{def}}{=} P \oplus (Q \oplus (\rho, X)) \\ !\pi.P \oplus (\rho, X) &\overset{\text{def}}{=} (\rho \uplus \{\!| \pi.(P \mid !\pi.P) |\!\}, X \cup \mathtt{fn}(\pi.P)) \\ (\boldsymbol{\nu} z)\, P \oplus (\rho, X) &\overset{\text{def}}{=} P\{z^n/z\} \oplus (\rho, X) \end{aligned}$$

where $k \geq 1$, $n = \min\{n \mid n \in \mathbb{N} \land z^n \notin X \cup \mathtt{fn}(P)\}$ (assuming that $z^0, z^1, \ldots$ is an enumeration of $\mathcal{N}$), and $\uplus$ is the standard multiset union operation.

Our translation from processes into functions is based on the π-calculus abstract machine of Phillips and Cardelli [8]. The translation procedure has two main steps. The first step consist of creating a process environment containing all the processes running in parallel. In this step, we also eliminate name restrictions by replacing the restricted names with new and unique names (we use the set X in (ρ, X) for tracking the free names in ρ). Also, $\oplus$ operates in such a way that the resulting environments contain only summations of one or more prefixed processes. The summation of processes should be understood as a list-type structure. This is important if, for instance, the semantics would be implemented. With such environments, it is now possible to define the second step of the translation procedure.

Definition 11 (Process Interpretation). *Let* $(\mathfrak{M}, w)$ *be a pointed model,* (ρ, X) *a process environment, and* $f_0 : \overrightarrow{\mathbf{K}} \to \overrightarrow{\mathbf{K}}$ *a model updating function. We define the process interpretation function* $[\![(\rho, X)]\!]_{(\mathfrak{M},w)}\, f_0 \subseteq \{f \mid f : \overrightarrow{\mathbf{K}} \to \overrightarrow{\mathbf{K}}\}$ *as follows:*

$$[\![(\rho, X)]\!]_{(\mathfrak{M},w)}\, f_0 \overset{\text{def}}{=} \begin{cases} I & \textit{if } I \neq \emptyset \\ \{f_0\} & \textit{otherwise} \end{cases}$$

where:

$$I \overset{\text{def}}{=} \bigcup_{\Sigma \in \rho} \; \bigcup_{\pi.P \text{ of } \Sigma} \textit{interleave}(\pi.P, (\rho - \{\!|\Sigma|\!\}, X), (\mathfrak{M}, w), f_0)$$

and we define the above interleaving function for each type of prefixed process as follows:

$$\textit{interleave}(\tau.P, (\rho, X), (\mathfrak{M}, w), f_0) \overset{\text{def}}{=} [\![P \oplus (\rho - \{\!|\tau.P|\!\}, X)]\!]_{(\mathfrak{M},w)}\, (id \circ f_0)$$

$$\textit{interleave}(\textbf{if } \varphi \textbf{ then } \pi'.P, (\rho, X), (\mathfrak{M}, w), f_0)$$
$$\overset{\text{def}}{=} \begin{cases} [\![\pi'.P \oplus (\rho - \{\!|\textbf{if } \varphi \textbf{ then } \pi' P|\!\}, X)]\!]_{(\mathfrak{M},w)}\, (id \circ f_0) & \textit{if } f_0(\mathfrak{M}, w) \models \varphi \\ \emptyset & \textit{otherwise} \end{cases}$$

$$\textit{interleave}(\overline{x}_s \eta.P, (\rho, X), (\mathfrak{M}, w), f_0)$$
$$\overset{\text{def}}{=} \bigcup_{\Sigma \in \rho} \; \bigcup_{x_r(z).Q \text{ of } \Sigma} [\![P \oplus (Q\{\eta/z\} \oplus (\rho - \{\!|\Sigma|\!\}, X))]\!]_{(\mathfrak{M},w)}\, (com(s, r, \eta) \circ f_0)$$

$$\textit{interleave}(x_r(z).P, (\rho, X), (\mathfrak{M}, w), f_0)$$
$$\overset{\text{def}}{=} \bigcup_{\Sigma \in \rho} \; \bigcup_{\overline{x}_s \eta.Q \text{ of } \Sigma} [\![P\{\eta/z\} \oplus (Q \oplus (\rho - \{\!|\Sigma|\!\}, X))]\!]_{(\mathfrak{M},w)}\, (com(s, r, \eta) \circ f_0)$$

The second step consist of interpreting the result of combining every compatible input/output prefixes as appropriate model updating functions. We use $[\![\cdot]\!]$ to interpret a process environment (i.e., the parallel composition of some summations). The interpretation $[\![\cdot]\!]$ has two additional parameters: the original model $(\mathfrak{M}, w)$ to be updated, and an initial model updating function f_0 representing the updates made so far.

In this second step, we need to resolve the non-deterministic choice for each summation running in parallel, and $[\![\cdot]\!]$ must contain all the possible updates. Hence, we consider all the terms of a summation. For each term $\pi.P$, the function $interleave(\pi.P, (\rho, X), (\mathfrak{M}, w), f_0)$ translates the interaction of $\pi.P$ with the processes in (ρ, X) into the appropriated model updating functions. The function *interleave* operates according with the prefix of $\pi.P$. This function translates τ prefixes as the model identity, performs the checking for the match prefixes, and translates the compatible input and output prefixes interactions as *com* functions. We sequentially compose the result of *interleave* with the initial function f_0 and recursively proceed until no further interpretations are possible.

Definition 12 (Satisfaction for $\mathcal{L}_{E\pi}$). *The satisfaction relation* $\models$ *between pointed models and formulas in* $\mathcal{L}_{E\pi}$ *is the least relation satisfying the rules* 1 *to* 4 *from Def.* 2 *plus the following rule:*

5. $(\mathfrak{M}, w) \models [P]\varphi$ *iff for every model updating function* f,

$$f \in [\![(P \oplus \{\!|\,|\!\}\,, \mathtt{fn}\,(P))]\!]_{(\mathfrak{M},w)}\,(id)$$

implies $f(\mathfrak{M}, w) \models \varphi$.

Recall that $[\![\cdot]\!]$ receives as an argument an initial model updating function f_0, representing the epistemic updates made so far. Thus, for starting computations, we initialize the interpretation with the identity function. Finally, we say that a formula $[P]\varphi$ is true at some possible world w, only when φ holds in every possible updated model by the *finished* computation of P.

For example, consider the following process:

$$P \stackrel{\text{def}}{=} (\overline{x}_1 p.\mathbf{0} + \overline{x}_2 q.\mathbf{0}) \mid x_3(z).\mathbf{0}$$

We need to first insert the above process into an empty environment, obtaining the the environment (ρ, X) such that:

$$\rho = \{\!|\overline{x}_1 p.\mathbf{0} + \overline{x}_2 q.\mathbf{0}, x_3(z).\mathbf{0}|\!\} \qquad X = \{x\}$$

Then, we have to translate each summation in ρ. In this case, we obtain the same results regardless of the summation we choose. Let choose the left summation, then we apply *interleave* to both summation terms:

$$\begin{aligned}&interleave(\overline{x}_1 p.\mathbf{0}, (\{\!|x_3(z).\mathbf{0}|\!\}\,, X), (\mathfrak{M}, w), id)\\ &\qquad = [\![(\{\!|\,|\!\}\,, X)]\!]_{(\mathfrak{M},w)}\,(com(1,3,p) \circ id) = (com(1,3,p) \circ id)\end{aligned}$$

$$\begin{aligned}&interleave(\overline{x}_2 q.\mathbf{0}, (\{\!|x_3(z).\mathbf{0}|\!\}\,, X), (\mathfrak{M}, w), id)\\ &\qquad = [\![(\{\!|\,|\!\}\,, X)]\!]_{(\mathfrak{M},w)}\,(com(2,3,q) \circ id) = (com(2,3,q) \circ id)\end{aligned}$$

Thus, for this example:

$$[\![(P \oplus \{\!|\!\}\,, \mathtt{fn}\,(P))]\!]_{(\mathfrak{M},w)}\,(id) = \{(com(1,3,p) \circ id), (com(2,3,q) \circ id)\}$$

We say that $(\mathfrak{M}, w) \models [P]\varphi$ iff the following two statements hold:

$$(com(1,3,p) \circ id)(\mathfrak{M}, w) \models \varphi$$
$$(com(2,3,q) \circ id)(\mathfrak{M}, w) \models \varphi$$

We finish this section with two remarks. First, the function $com(s, r, \varphi)$ is defined only when $K_s\varphi$ (for φ a formula). This is because we only model *truthful* agents. If an agent emits an untruthful message in a process P, then the translation f of that execution will cause $f(\mathfrak{M}, w) \models \psi$ to be vacuously satisfied for any ψ (being consistent with Action Model Logic). Finally, observe that we can write a process having an infinite chain of reductions. The translation f of such an infinite chain is also an infinite sequential composition. Thus, $f(\mathfrak{M}, w)$ is undefined, vacuously satisfying any formula ψ (being consistent with the notion that "ψ holds *after* the update f").

5 The Dining Cryptographers Protocol

In this section we show the utility of our language by encoding Chaum's Dinining Cryptographers Protolcol [3]. Quoting Chaum's paper:

> Three cryptographers are sitting down to dinner at their favorite three-star restaurant. Their waiter informs them that arrangements have been made with the maitre d'hotel for the bill to be paid anonymously. One of the cryptographers might be paying for the dinner, or it might have been NSA (U.S. National Security Agency). The three cryptographers respect each other's right to make an anonymous payment, but they wonder if NSA is paying. They resolve their uncertainty fairly by carrying out the following protocol:
>
> Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer on his right, so that only the two of them can see the outcome. Each cryptographer then states aloud whether the two coins he can see–the one he flipped and the one his left-hand neighbor flipped–fell on the same side or on different sides. If one of the cryptographers is the payer, he states the opposite of what he sees. An odd number of differences uttered at the table indicates that a cryptographer is paying; an even number indicates that NSA is paying (assuming that the dinner was paid for only once). Yet if a cryptographer is paying, neither of the other two learns anything from the utterances about which cryptographer it is.

For capturing the Dining Cryptographers problem, we need to add a new feature to the language $\mathcal{L}_{E\pi}$. We would like, for example, an agent A to receive information from agent B, and be able to act differently depending on what information is received. For this reason, we allow names to appear in the formulas in the conditionals. We allow this feature under the assumption that if a name has not been replaced in a formula by the time the conditional is checked, then the formula is automatically false.

We represent the three cryptographers with the following set of agents:

$$\mathcal{A} = \{A, B, C\}$$

We will use the following atomic propositions:

- atoms p_i stand for *"agent i paid"*;
- atoms b_{ij} stand for the shared bit (coin toss) between i and j;
- atoms C_i represent the bits that the agents will exchange depending on the results of the coin flips and whether or not they paid.

We will only consider the case when some agent is paying. We are after some formula like:

$$\mathit{precondition} \Rightarrow [P]\, \mathit{postcondition}$$

where P is a process modeling the protocol.

The postcondition is easiest to describe, we simply want it to say that all agents know that some agent paid, but that no agent other than the payer knows which agent paid:

$$\begin{aligned} \mathit{postcondition} &\overset{\text{def}}{=} K_A(p_A \vee p_B \vee p_C) \wedge K_B(p_A \vee p_B \vee p_C) \wedge K_C(p_A \vee p_B \vee p_C) \\ &\wedge \neg K_A p_B \wedge \neg K_A p_C \wedge \neg K_B p_A \wedge \neg K_B p_C \wedge \neg K_C p_A \wedge \neg K_C p_B \end{aligned}$$

The formula *precondition* will say the following:

- One of the agents paid the bill;
- The agents who did not pay the bill don't know who paid the bill (ie., they consider it possible that either of the other two agents or the NSA paid);
- After acquiring information from the execution protocol, the agents can infer that an agent had paid the bill, without knowing which agent.

$$\begin{aligned} \mathit{precondition} &\overset{\text{def}}{=} \alpha \wedge \beta \wedge \gamma \\ \alpha &\overset{\text{def}}{=} (p_A \wedge \neg p_B \wedge \neg p_C) \vee (\neg p_A \wedge p_B \wedge \neg p_C) \vee (\neg p_A \wedge \neg p_B \wedge p_C) \\ \beta &\overset{\text{def}}{=} (\neg p_A \Rightarrow (\neg K_A p_B \wedge \neg K_A p_C)) \\ &\wedge (\neg p_B \Rightarrow (\neg K_B p_A \wedge \neg K_B p_C)) \\ &\wedge (\neg p_C \Rightarrow (\neg K_C p_A \wedge \neg K_C p_B)) \\ \gamma &\overset{\text{def}}{=} K_A((C_A \oplus C_B \oplus C_C) \wedge \neg p_A \Rightarrow (p_B \vee p_C)) \\ &\wedge K_B((C_A \oplus C_B \oplus C_C) \wedge \neg p_B \Rightarrow (p_A \vee p_C)) \\ &\wedge K_C((C_A \oplus C_B \oplus C_C) \wedge \neg p_C \Rightarrow (p_A \vee p_B)) \end{aligned}$$

The formula α says that someone paid; β says that non-payers do not know who paid; γ says that after learning all C_i's everyone knows that some agent has paid. Also, observe that the $\oplus$ above stands for the XOR logical connective (not the semantic operation defined in the previous section).

We now describe the protocol as a π-calculus process. The protocol will be represented by the process P, which has three components acting in parallel:

$$P \overset{\text{def}}{=} P_{AB} \mid P_{BC} \mid P_{CA}$$

Where:

$$\begin{aligned} P_{AB} &\overset{\text{def}}{=} \overline{x}_A b_{AB}.\mathbf{0} \mid x_B(z).Q_B \\ P_{BC} &\overset{\text{def}}{=} \overline{y}_B b_{BC}.\mathbf{0} \mid y_C(z).Q_C \\ P_{CA} &\overset{\text{def}}{=} \overline{w}_C b_{CA}.\mathbf{0} \mid w_A(z).Q_A \end{aligned}$$

Process P_{ij} dictates i to share her bit with the agent to her right, and wait for the bit from the agent to her left. After this bit exchange, i continues the execution with the process Q_i.

We show Q_i for agent A:

$$\begin{aligned} Q_A \overset{\text{def}}{=} &\ \mathbf{if}\ (p_A \wedge (z \oplus b_{AB}))\ \mathbf{then}\ \overline{u}_A \neg C_A.\mathbf{0} \\ &+ \mathbf{if}\ (p_A \wedge \neg(z \oplus b_{AB}))\ \mathbf{then}\ \overline{u}_A C_A.\mathbf{0} \\ &+ \mathbf{if}\ (\neg p_A \wedge (z \oplus b_{AB}))\ \mathbf{then}\ \overline{u}_A C_A.\mathbf{0} \\ &+ \mathbf{if}\ (\neg p_A \wedge \neg(z \oplus b_{AB}))\ \mathbf{then}\ \overline{u}_A \neg C_A.\mathbf{0} \end{aligned}$$

After learning the shared bit z with the agent to her left, A announces (through channel v) the result of computing the exclusive-or of her two known bits. If A paid, then the opposite result is announced.

The Model

We now describe an appropriate Kripke model in which the formulas can be interpreted, giving a correct result. We describe the states of the Kripke model according to the propositions which hold in the states. First, at most one of the propositions p_A, p_B, p_C holds in any given state. Therefore we may label each state with one of those propositions, or with a $\emptyset$ if it is the case that the NSA paid the bill. For any given payer, any combination of coin-flips is possible, so we have a state corresponding to each combination for each payer. However, given the payer, and the coin-flip bits, the sent bits C_i are determined. So we are left with 32 states in the model. A typical state may be described as:

$$w_0 \overset{\text{def}}{=} (p_A, b_{AB}, \neg b_{BC}, b_{CA}, C_A, C_B, C_C)$$

In this state, agent A has paid the bill, so he sends the message C_A, despite the XOR of the two bits available to him being false. Agents B and C send messages C_B and C_C respectively, because each has seen exactly one positive coin-flip bit, and neither has paid the bill.

Now, agent A knows he paid the bill, and furthermore knows the result of his own coin-flip. So p_A and b_{AB} hold in any state that A considers possible from w_0. However, the results of the other coin-flips are not initially known to A, so there are actually three other states that A considers accessible from w_0, for example:

$$w_1 \overset{\text{def}}{=} (p_A, b_{AB}, b_{BC}, b_{CA}, C_A, \neg C_B, \neg C_C)$$

Things are more interesting from the perspective of a different agent. For example, agent B considers it possible that any one of agent A, agent C or the NSA paid the bill. Therefore there are are 11 other states accessible from state w_0 from the perspective of agent B, including, for example:

$$w_2 \overset{\text{def}}{=} (p_C, b_{AB}, \neg b_{BC}, \neg b_{CA}, C_A, C_B, C_C)$$

And:

$$w_3 \overset{\text{def}}{=} (\emptyset, b_{AB}, \neg b_{BC}, b_{CA}, \neg C_A, C_B, C_C)$$

It is easy to check that the precondition is satisfied in any state of the model in which an agent paid. After the execution of the process P, all the agents know the valuations of the bits C_i, but the key observation is that any non-payer considers two states – with two different payers – possible. For example, after the execution of processes P in state w_0, agent B considers the updated state corresponding to w_2 possible, showing that B does not know the identity of the payer. Analogous remarks may be made about C, showing the A's anonymity has been preserved.

6 Related and Future Work

Related Work

One of the first approaches to epistemic dynamics was that of public announcements [9]. The action models were introduced in [1]. With action models we can describe several communication scenarios. There are other extensions to this approach. In [13] the authors augment action models with updates to the atomic facts of the world, and use a different base epistemic language similar to propositional dynamic logic [5]. By combining temporal and epistemic logic [12] similar approaches arise. In [12], the temporal component gives a similar interpretation: the knowledge changes according to a sequence of model-product updates. An important difference from our approach is that the sequences of action models in [12] are given a priori. In [14], the authors present a codification of the Dining Cryptographers protocol with action models. In this paper, we only emphasize the convenience of using π-calculus to codify the protocol. Then, by using the semantics in Sect. 4, we can translate such a codification into action models.

There are also modal logics for reasoning about processes [7]. These logics, sometimes called Hennessy-Milner logics, characterize several variants of process bisimulation. In [6], the author extends a Hennessy-Milner logic for π-calculus with an epistemic operator. In this endogenous logic, the processes are not syntactically present in the language, and there is a unique agent: an external observer. In [2], Chadha et al. propose an epistemic logic for the applied π-calculus (a version of the π-calculus including other data than names). They use their logic to verify properties of protocols. In this logic, the epistemic modal operator is not relative to many possible agents, but to only one—the potential intruder in private communications.

Future Work

In this paper, we presented a method for the interpretation and analysis of epistemic dynamics with π-calculus process communication in a multi-agent setting. The dynamic epistemic logic that we propose, uses π-calculus processes as the communicating actions of the agents. We translate these processes as classes of sequential compositions of model updating functions. This functions describe the epistemic updates caused by the interaction of the processes. We base our translation in the π-calculus abstract machine of [8]. It is worth noting that, by using an abstract machine, we obtain the benefit of a method for algorithmically verifying satisfaction.

We propose some routes for further research. First, it is possible to extend the notion of process bisimulation to our language. By making some simple syntactic restrictions to the language, we can obtain modular processes that preserve bisimulation. Secondly, in [11] the authors present a complete proof system for a fragment of the language presented in this paper. Such a proof system is restricted to finite processes. It is desirable to study a proof system capable of describing infinite processes properties. Finally, the communication in our language allows exchanging both formulas and names. In some contexts, receiving a formula instead of a name may cause the process to be unable to continue (e.g., the process $x_i(z).\overline{z_i}y.\mathbf{0}$ cannot continue if a formula is received). It would be interesting to explore a type system to handle such situations.

References

1. Baltag, A., Moss, L.S., Solecki, S.: The logic of public announcements, common knowledge, and private suspicions. Technical Report SEN-R9922, CWI (1999)
2. Chadha, R., Delaune, S., Kremer, S.: Epistemic logic for the applied pi calculus. In: Lee, D., Lopes, A., Poetzsch-Heffter, A. (eds.) FMOODS 2009. LNCS, vol. 5522, pp. 182–197. Springer, Heidelberg (2009)
3. Chaum, D.: The dining cryptographers problem: unconditional sender and recipient untraceability. J. Cryptol. 1(1), 65–75 (1988)

4. van Ditmarsch, H., var der Hoek, W., Kooi, B.: Dynamic Epistemic Logic. Springer, Heidelberg (2007)
5. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge (2000)
6. Mardare, R.: Observing distributed computation. a dynamic-epistemic approach. In: Mossakowski, T., Montanari, U., Haveraaen, M. (eds.) CALCO 2007. LNCS, vol. 4624, pp. 379–393. Springer, Heidelberg (2007)
7. Milner, R., Parrow, J., Walker, D.: Modal logics for mobile processes. Theoretical Computer Science 114(1), 149–171 (1993)
8. Phillips, A., Cardelli, L.: A correct abstract machine for the stochastic pi-calculus. In: Concurrent Models in Molecular Biology (Bioconcur 2004), London (2004)
9. Plaza, J.A.: Logics of public communications. In: Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems, pp. 201–216 (1989)
10. Sangiorgi, D., Walker, D.: The Pi-Calculus — A Theory of Mobile Processes. Cambridge University Press, Cambridge (2001)
11. Ufferman, E., Góngora, P.A., Hernández-Quiroz, F.: A complete proof system for a dynamic epistemic logic based upon finite π-calculus processes. In: Proceedings of Advances in Modal Logics 2010. College Publications (to appear 2010)
12. van Benthem, J., Gerbrandy, J., Hoshi, T., Pacuit, E.: Merging frameworks for interaction. Journal of Philosophical Logic 38(5), 491–526 (2009)
13. van Benthem, J., van Eijck, J., Kooi, B.: Logics of communication and change. Inf. Comput. (2005)
14. van Eijck, J., Orzan, S.: Epistemic verification of anonymity. Electron. Notes Theor. Comput. Sci. 168, 159–174 (2007)

What Happened to My Commitment? Exception Diagnosis among Misalignment and Misbehavior

Özgür Kafalı[1], Federico Chesani[2], and Paolo Torroni[2]

[1] Department of Computer Engineering - Boğaziçi University
TR-34342, Bebek, İstanbul - Turkey
ozgurkafali@gmail.com
[2] DEIS - University of Bologna
V.le Risorgimento, 2, 40136, Bologna - Italy
{federico.chesani,paolo.torroni}@unibo.it

Abstract. This paper studies misalignment of commitments associated with temporal constraints. We propose a diagnosis algorithm where agents reason based on the current states of their commitments. We also provide an alignment policy that can be applied by an agent when the diagnosis algorithm identifies a misalignment. We formalize a delivery process from e-commerce using $\mathcal{REC}$, and present a case study to demonstrate the workings of our approach.

1 Introduction

Agent contracts, on one hand, describe how agents should act given certain situations. On the other hand, they provide clues on whom to blame when a violation occurs. A contract is often represented by a *commitment* [7], which is formed between a debtor and a creditor about a specific property. In a realistic environment, a commitment is associated with a deadline [8], telling that its property has to be brought about within that bound. If not, a violation occurs regarding that commitment. A violation of a commitment means an *exception* for its creditor. In a distributed contract-based setting, each agent keeps track of its own commitments. Thus, the cause of such an exception is often a misalignment between the debtor and the creditor's individual copies of the same commitment. Example 1 presents such a scenario from a real-life delivery process.

Example 1. *The online purchase of a book.* Let us consider a scenario with a *customer* who wishes to buy a book from a *store*. The transaction needs two additional parties: a *bank* and a *deliverer*. The customer selects the book she wishes to purchase from the website of the store, and pays for it using a bank transfer on Monday. The contract between the customer and the store tells that when the customer's payment is verified by the bank, then the book will be delivered within five business days. Assume that the bank verifies the customer's payment on Wednesday. Now, the customer expects delivery until the following Wednesday. However, the bank does not notify the store about the verification of the customer's payment until Friday. When the store receives the notification, he infers the deadline for delivery as the following Friday (two days later than the customer has previously inferred). When the customer does not receive the book on Wednesday, she asks the store about it. The store tells the customer that there are two more days until the deadline. At this point, the customer understands that there is a misalignment of their commitments. She has to decide what to do next. One such possibility is to update the violated contract and to wait for the new deadline.

There is one key point in Example 1 that provides the motivation for our diagnosis approach: the point where the customer becomes aware that the store has a similar commitment about the delivery of the book, but with a different deadline. This is a typical misalignment of commitments which is caused by different observations of the debtor and the creditor [3]. Another possible cause of misalignment is simply a misbehavior of the debtor (e.g., the store delegates its commitment to a deliverer without respecting its deadline with the customer). Moreover, a misalignment may also be caused by misunderstandings among agents (e.g., the customer receives another book instead of the purchased one). Schroeder and Schweimeier [6] study such cases based on negotiation and argumentation theory. However, in this work, we only consider misbehavior and misalignment caused by different observations.

Among the few related work, Chopra and Singh [3] formalize commitment alignment in multiagent systems. They consider misalignment of commitments that arise from different observations of the debtor and the creditor. In their solution, the creditor informs the debtor when the condition of their conditional commitment is satisfied so that debtor and creditor can infer the same base-level commitment. This approach requires extra communication among the participants of the commitment. However, note that exceptions are rare cases in process execution, and forcing extra communication to ensure alignment would be an unnecessary overhead most of the times. We then choose to verify alignment on demand, i.e., when an exception occurs. Another shortcoming of Chopra and Singh's formalization is the lack of deadlines for commitments. Without them, it is hard to capture realistic e-commerce scenarios.

Accordingly, we propose a distributed collaborative process to diagnose exceptions risen by conflicting deadlines of commitments, caused either by misalignment or by misbehavior. When the creditor agent detects that one of its commitments is violated, it initiates the diagnosis process by making a diagnosis request to the commitment's debtor. Diagnosis continues based on the exchange of commitments that are relevant to (i.e., having the same property with) the violated commitment. If we follow Example 1, the customer makes a diagnosis request to the store about her violated commitment. Based on the complexity of the exception, the diagnosis process may involve a set of agents other than the store (e.g., the store may have delegated its commitment to a further agent). In the end, the diagnosis results in one of the following:

1. a failure, where no conclusions can be drawn,
2. a misbehavior diagnosis, with a culprit agent identified as being responsible, or
3. a misalignment diagnosis, with a possible commitment to be aligned with.

In the case of misalignment, the agents can maintain alignment via an alignment policy described by a set of commitment *update* rules. This is often the case for real-life delivery scenarios; the customer may accept to wait a bit longer in the case of a misalignment over the deadlines. Alternatively, agents could start negotiating on what to do next. That could solve both situations of misalignment and misbehavior. However we do not address negotiation in this paper, but only diagnosis and a simple form of automatic realignment.

In order to evaluate our approach, we extend the scenario described in Example 1. We formalize the agents' interactions in $\mathcal{REC}$ [1], a reactive form of *Event Calculus* [5]. Any tool could have been used, for this purpose, that provides run-time monitoring of commitments. However, $\mathcal{REC}$ is the only tool we are aware of, that accommodates commitments with temporal constraints.

Our diagnosis architecture includes coupled knowledge-bases, in which agents store the protocol rules and contracts they agree upon. That is, a protocol rule is contained in the knowledge-base shared between two agents if that rule concerns those two agents. Similarly, a contract is contained in the knowledge-base shared between its participants. In particular, these shared knowledge-bases contain commitment and protocol rules in $\mathcal{REC}$ and the facts agreed upon by the interested parties. They do not contain an extensional description of the commitments, e.g., the current states of the commitments.

Each agent has a separate trace of happened events according to what it observes. Thus, an agent can only track down the status of its own commitments. We assume that agents are always honest and collaborative during diagnosis. That is, when an agent is requested to take part in the diagnosis process, it does so. Note that being identified as a culprit for misbehavior does not necessarily make an agent dishonest or malicious. The agent may have unintentionally caused an exception, but can still help identify the cause of it.

The algorithm we propose always terminates, is sound and, if associated with suitable agent policies, is capable of removing misalignment whenever possible. We discuss these and other results. Eventually, we illustrate the approach by presenting three different traces of events that lead to separate diagnosis results.

The rest of the paper is organized as follows. The next section reviews commitments. Section 3 describes the delivery scenario to be used as our running example. Section 4 describes the similarity relation we propose to verify alignment of commitments. Then, Section 5 describes our diagnosis process which makes use of the similarity relation and an alignment policy to be used in the case of misalignment. Section 6 describes the case study. Finally, Section 7 concludes the paper with further discussion.

2 Commitments in $\mathcal{REC}$

We use commitments [7] to represent agent contracts. A commitment is formed between a debtor and a creditor, in which the debtor is committed to the creditor for bringing about a property. In this paper, we use $\mathcal{REC}$ [1,8] to model *time-aware* commitments (i.e., commitments with temporal constraints). In $\mathcal{REC}$ we can express that an event *initiates* (or *terminates*) a *fluent* or *property* of the system, by way of *initiates(Event, Fluent, Time)* relations.

$\mathcal{REC}$ models two types of temporal constraints on commitments: (1) an existential temporal constraint where the property of the commitment has to be brought about inside a time interval, and (2) a universal temporal constraint where the property of the commitment has to be maintained valid along a time interval. Here, we focus only on base-level commitments with existential temporal constraints.

We use the following syntax to represent an existential base-level commitment throughout the paper:

$$s(c(x, y, property(e(t_1, t_2), p))),$$

where x and y are the debtor and the creditor of the commitment, respectively. The existential temporal constraint $e(t_1, t_2)$ on the property p means that the logic formula p has to become true at a specific time t within t_1 and t_2 ($t_1 \leq t \leq t_2$). If so, the commitment is *satisfied*. Otherwise, the commitment is *violated*. We use s to show the current status of the commitment at a specific time point[1]. When the commitment is first created by the *create* operation [9], the commitment's state is *active*.

We use the following syntax to represent a happened event (e.g., as a *tell* message):

$$hap(event(tell(x, y, e(p_1, ..., p_n))), t),$$

where x and y are the sender and receiver of the message, respectively. The event description is represented by $e(p_1, ..., p_n)$, where p_1 through p_n are the parameters associated with e. The time of occurrence of the event is represented by t.

3 Running Example

Figure 1 shows the delivery process introduced in Example 1. We assume that the customer has already placed the order, by direct or indirect interaction with the store, e.g., via an e-commerce Web site. We thus focus on the subsequent phases (payment & delivery).

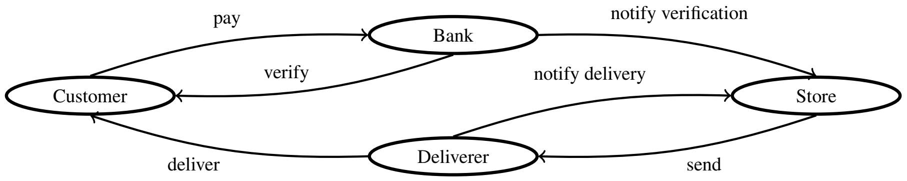


Fig. 1. Delivery Process

In a desired execution, first the customer sends the payment to the bank regarding its purchase from the store (*pay*). Then, the bank verifies the payment of the customer (*verify*), and informs the store about the verification (*notify verification*). Upon receiving the verification, the store requests the delivery of the book from the deliverer (*send*). Finally, the deliverer delivers the book to the customer (*deliver*), and informs the store about the delivery (*notify delivery*). Listings 1 through 5 show how this process is formalized using $\mathcal{REC}$ in coupled knowledge-bases.

Listing 1 shows the $\mathcal{REC}$ rules for the customer and the bank. We model the agents' interactions as *tell* events from a sender towards a receiver. The first two rules (CB_1 and

[1] Note that here we use this notation for presentation purposes only. In $\mathcal{REC}$, the status is also a parameter of the commitment description.

CB_2) describe the effects of such events in terms of fluents *paid* and *verified*. The last rule (CB_3) corresponds to a *create* operation for commitments [9,8]. The semantics is that when the customer sends the payment for an item of the store to the bank, then the bank will be committed to verifying that payment in three time units (without loss of generality, we use days as the time unit from now on). Note that lines starting with % are comments.

```
% CB1: pay
initiates(tell(Customer,Bank,pay(Item)),paid(Item),_).

% CB2: verify payment
initiates(tell(Bank,Customer,verify(Item)),verified(Item),_).

% CB3: pay-verify commitment
create(tell(Customer,Bank,pay(Item)),Bank,
        c(Bank,Customer,property(e(Ts,Te),verified(Item))),Ts):-
        Te is Ts+3.
```

Listing 1. Coupled knowledge-base of the customer and the bank

Listing 2 shows the $\mathcal{REC}$ rules for the bank and the store. Note that the only rule BS_1 is similar to the rule CB_2, in which the only difference is the receiver. That is, the bank notifies the store about the verification of the customer's payment.[2]

```
% BS1: verify payment
initiates(tell(Bank,Store,verify(Item)),verified(Item),_).
```

Listing 2. Coupled knowledge-base of the bank and the store

Listing 3 shows the $\mathcal{REC}$ rules for the customer and the store. The only rule CS_1 describes the commitment between them. The semantics is that when the bank sends the payment verification, then the store will be committed to deliver the item in five days.

```
% CS1: verify-deliver commitment
create(tell(Bank,_,verify(Item)),Store,
        c(Store,Customer,property(e(Ts,Te),delivered(Item))),Ts):-
        Te is Ts+5, holds_at(in_stock(Item,Store),Ts).
```

Listing 3. Coupled knowledge-base of the customer and the store

Listing 4 shows the $\mathcal{REC}$ rules for the store and the deliverer. The first two rules (SD_1 and SD_2) describe the events for the request of a delivery, and the delivery itself. The last rule SD_3 describes the commitment between the two agents. The semantics is

[2] For the sake of simplicity, we do not indicate further conditions on $Bank$, $Store$, and $Item$, which are free variables. A detailed implementation would require to express restrictions on such variables, i.e., to define the "context" [2].

that when the store requests the delivery of an item, then the deliverer will be committed to deliver that item in three days.

```
% SD1: send for delivery
initiates(tell(Store,Deliverer,send(Item)),sent(Item),_).

% SD2: deliver
initiates(tell(Deliverer,Store,deliver(Item)),delivered(Item),_).

% SD3: send-deliver commitment
create(tell(Store,Deliverer,send(Item)),Deliverer,
        c(Deliverer,Store,property(e(Ts,Te),delivered(Item))),Ts):-
        Te is Ts+3.
```

Listing 4. Coupled knowledge-base of the store and the deliverer

Listing 5 shows the $\mathcal{REC}$ rules for the deliverer and the customer. The only rule DC_1 is similar to the rule SD_2, in which the only difference is the receiver.

```
% DC1: deliver
initiates(tell(Deliverer,Customer,deliver(Item)),
            delivered(Item),_).
```

Listing 5. Coupled knowledge-base of the deliverer and the customer

4 Commitment Similarity

Chopra and Singh [3] propose a stronger-weaker relation for commitments using the commitments' conditions and propositions (i.e., properties). However, we do not focus on the properties of the commitments. But, we make comparisons based on the temporal constraints associated with their properties (i.e., deadlines), and the agents involved (i.e., debtor and creditor). Accordingly, we propose the following similarity levels for commitments.

Definition 1. Commitment $c_1 = s_1(c(x_1, y_1, \mathit{property}(e(t_1, t_2), p_1)))$ is relevant to commitment $c_2 = s_2(c(x_2, y_2, \mathit{property}(e(t_3, t_4), p_2)))$ if property p_1 is identical to property p_2. ▮

Example 2. $\mathit{active}(c(\mathit{deliverer}, \mathit{store}, \mathit{property}(e(7.0,10.0), \mathit{delivered}(\mathit{book}))))$ is relevant to $\mathit{violated}(c(\mathit{store}, \mathit{customer}, \mathit{property}(e(3.0,8.0), \mathit{delivered}(\mathit{book}))))$ since their properties are identical.

Definition 2. Commitment $c_1 = s_1(c(x_1, y_1, \mathit{property}(e(t_1, t_2), p_1)))$ is a forward-shift of commitment $c_2 = s_2(c(x_2, y_2, \mathit{property}(e(t_3, t_4), p_2)))$ if c_1 is relevant to c_2, $x_1 = x_2$, $y_1 = y_2$, $t_1 > t_3$, and $t_2 > t_4$. ▮

Example 3. $active(c(store, customer, property(e(5.0,10.0), delivered(book))))$ is a forward-shift of $violated(c(store, customer, property(e(3.0,8.0), delivered(book))))$ since they are relevant, and the deadline of the former is greater than that of the latter.

Note that the forward-shift of a commitment usually has a different status than the commitment itself. But, we do not restrict the definition of forward-shift with constraints on commitment status.

Definition 3. The debtor-creditor couple (x_1, y_1) is a delegatee of the debtor-creditor couple (x_2, y_2) if (1) $x_1 \neq x_2$ and $y_1 = y_2$, or (2) $x_1 \neq y_2$ and $x_2 = y_1$. ▮

Note that [9,3] propose a more restricted definition of delegation which is limited to case (1). There, when a delegation occurs, only the debtor of the commitment changes. Definition 3 extends the notion of delegation by case (2). This provides a way to trace a set of delegated commitments when diagnosing an exception (e.g., identify the sequence of delegations). The first case does not support this by just looking at the commitments themselves. That is, if the commitment is delegated several times, it is not possible to keep track of the delegation sequence.

Example 4. $(deliverer\ , store)$ is a delegatee of $(store\ , customer)$, and $(driver\ , deliverer)$ is a delegatee of $(deliverer\ , store)$. Thus, we have a sequence of two delegations: (1) from $(store, customer)$ to $(deliverer, store)$, and (2) from $(deliverer, store)$ to $(driver, deliverer)$.

However, note that these delegatee relations only make sense when embedded in a commitment as described next.

Definition 4. Commitment $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p_1)))$ is a delegation of commitment $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p_2)))$ if c_1 is relevant to c_2, $t_1 \geq t_3$, $t_2 \leq t_4$, and (x_1, y_1) is a delegatee of (x_2, y_2). ▮

Note that the delegation of a commitment usually has the same status as the commitment itself.

Example 5. $active(c(deliverer, store, property(e(5.0,8.0), delivered(book))))$ is a delegation of $active(c(store, customer, property(e(3.0,8.0), delivered(book))))$ since they are relevant, and the debtor-creditor couple of the former is a delegatee of that of the latter.

Definition 5. Commitment $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p_1)))$ is a forward-shift delegation of commitment $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p_2)))$ if c_1 is relevant to c_2, $t_1 > t_3$, and $t_2 > t_4$, and (x_1, y_1) is a delegatee of (x_2, y_2). ▮

Example 6. $active(c(deliverer, store, property(e(7.0,10.0), delivered(book))))$ is a forward-shift delegation of $violated\ (c\ (store, customer, property\ (\ e\ (3.0, 8.0), delivered\ (book))))$ since they are relevant, the deadline of the former is greater than that of the latter, and the debtor-creditor couple of the former is a delegatee of that of the latter.

Remark 1. Relevance is an equivalence relation, i.e., it is reflexive, symmetric and transitive. Forward-shift instead is asymmetric and thus non reflexive, but it is transitive. Delegation and forward-shift delegation are neither symmetric nor transitive.

Listing 6 shows the $\mathcal{REC}$ rules for the above definitions. Note the syntax for CLP constraints $x \neq y$ (\=(x, y)) and $x > y$ (>(x, y)).

```
% S1: relevant
relevant(c(X1,Y1,property(e(Ts1,Te1),P)),
         c(X2,Y2,property(e(Ts2,Te2),P))).

% S2: forward-shift
fshift(c(X1,Y2,property(e(Ts1,Te1),P1)),
       c(X2,Y2,property(e(Ts2,Te2),P2))):-
       relevant(c(X1,Y1,property(e(Ts1,Te1),P1)),
                c(X2,Y2,property(e(Ts2,Te2),P2))),
       >(Ts1,Ts2), >(Te1,Te2).

% S3: delegatee
delegatee((X1,Y),(X2,Y)):- \=(X1,X2).
delegatee((X1,Y),(Y,Y2)):- \=(X1,Y2).

% S4: forward-shift delegation
fshift_delegation(c(X1,Y1,property(e(Ts1,Te1),P1)),
                  c(X2,Y2,property(e(Ts2,Te2),P2))):-
                  relevant(c(X1,Y1,property(e(Ts1,Te1),P1)),
                           c(X2,Y2,property(e(Ts2,Te2),P2))),
                  >(Ts1,Ts2), >(Te1,Te2),
                  delegatee((X1,Y1),(X2,Y2)).
```

Listing 6. Commitment similarity in $\mathcal{REC}$

5 Diagnosis Process: Architecture, Algorithm, and Properties

The purpose of the diagnosis process is to investigate the status of commitments in the system, and return a possible cause of violation. Let us now specify it more formally. Table 1 summarizes our notation.

Definition 6. Given a set of agents $A \subseteq \mathcal{A}$, and a violated commitment $C \in \mathcal{C}$, we call *diagnosis* of C by A an atom $\delta \in \{$failure, misbehavior(X), misalignment(C')$\}$, where $X \in A$, $C' \in \mathcal{C}$. ∎

A *correct* diagnosis describes the reason of C's violation. It identifies one of the following:

1. a *failure*, when no relevant commitment is found regarding a diagnosis request for the violated commitment,

Table 1. Notation used for diagnosis process

Symbol	Description
$\mathcal{C}$	the domain of commitments
$\mathcal{A}$	the domain of agents
$\mathcal{C}_A$	the set of commitments that agent A is aware of
$\mathcal{C}_\mathcal{A}$	the set of all commitments ($\mathcal{C}_\mathcal{A} = \bigcup_{A \in \mathcal{A}} \mathcal{C}_A$)
$\mathcal{C}_A^C$	the set of commitments in $\mathcal{C}_A$ that are relevant to C
$\mathcal{C}_\mathcal{A}^C$	the set of commitments in $\mathcal{C}_\mathcal{A}$ that are relevant to C
$\mathcal{C}_A^{Cf}$	the set of commitments in $\mathcal{C}_A$ that are a forward-shift of C
$\mathcal{C}_A^{CX}$	the set of commitments in $\mathcal{C}_A$ that are a delegation of C to $X \in \mathcal{A}$
$\mathcal{C}_A^{CfX}$	the set of commitments in $\mathcal{C}_A$ that are a forward-shift delegation of C to $X \in \mathcal{A}$
$\mathcal{C}_A^{CA}$	$\mathcal{C}_A^C \setminus (\mathcal{C}_A^{Cf} \cup \mathcal{C}_A^{CX})$ for all $X \in \mathcal{A}$

2. a *misbehavior*, when the violated commitment is confirmed by its debtor, or
3. a *misalignment*, when an active commitment is identified that is relevant to the violated commitment.

Definition 7. A *correct diagnosis* of $C \in \mathcal{C}$ by $A \subseteq \mathcal{A}$ is a diagnosis of C by A such that:

– if δ = failure, then $\exists x, y \in \mathcal{A}$, $c_1 = \mathit{violated}(c(x, y, \mathit{property}(\ e(t_1, t_2), p))) \in \mathcal{C}_\mathcal{A}^C$, such that $c_1 \in \mathcal{C}_y$, and $\mathcal{C}_x^C = \emptyset$.
– if δ = misbehavior(x), then $\exists x, y \in \mathcal{A}$, $c_1 = \mathit{violated}(c(x, y, \mathit{property}(\ e(t_1, t_2), p))) \in \mathcal{C}_\mathcal{A}^C$, such that $c_1 \in \mathcal{C}_y$, $c_1 \in \mathcal{C}_x$, and either $\mathcal{C}_x^{c_1 Y} = \emptyset$ for all $Y \in \mathcal{A}$, or $\mathcal{C}_x^{c_1 fY} \neq \emptyset$ for some $Y \in \mathcal{A}$.
– if δ = misalignment(c_2), then $\exists x, y \in \mathcal{A}$, $c_1 = \mathit{violated}(c(x, y, \mathit{property}(\ e(t_1, t_2), p)))$, $c_2 = \mathit{active}(c(x, y, \mathit{property}(\ e(t_3, t_4), p))) \in \mathcal{C}_\mathcal{A}^C$, such that $c_1 \in \mathcal{C}_y$, $c_2 \in \mathcal{C}_x$, and c_2 is a forward shift of c_1. ∎

Let us now inspect each case described in Definition 7: (1) When the diagnosis identifies a failure, then there should be a commitment that is relevant to C, such that its creditor infers the commitment but its debtor does not. (2) When the diagnosis identifies a misbehavior, then there should be a violated commitment that is relevant to C, such that its debtor infers the commitment, and the debtor either has not delegated the commitment, or it has delegated the commitment without respecting its deadline (i.e., forward-shift delegation). (3) When the diagnosis identifies a misalignment, then there should be a violated commitment that is relevant to C, such that its debtor infers that the commitment is active (i.e., forward-shift).

Note that Definition 7 describes diagnosis considering the multiagent system as a whole (e.g., all the commitments and the agents). Accordingly, we propose an algorithm to achieve correct diagnosis in a distributed fashion. Figure 2 describes the architecture used to perform such distributed diagnosis. Each agent has a separate $\mathcal{REC}$ engine running in background. The $\mathcal{REC}$ engines are used for monitoring purposes only. For example, the customer can run $\mathcal{REC}$ to monitor the status of its commitments during process execution. When an agent detects that one of its commitments is violated,

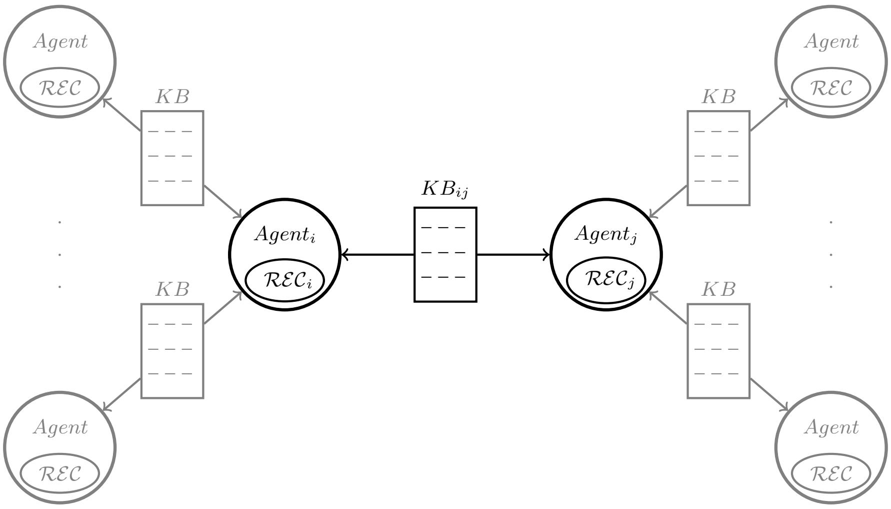


Fig. 2. Diagnosis Architecture

it will verify whether the violated commitment is aligned with other agents' commitments. Note that other agents may infer different (but relevant) commitments due to their individual observations of happened events. In such a case, the creditor of the violated commitment initiates a diagnosis process to find out what has happened.

Algorithm 1 implements this diagnosis process. The process continues as a series of diagnosis requests among a set of agents until one of the outcomes in Definition 7 is reached. Each iteration in the process is a diagnosis request from the creditor of a violated commitment to its debtor. We assume that agents are honest and collaborative during the whole process. In addition, we do not allow multiple delegations of the same commitment to several agents. Thus, an agent can delegate its commitment to a single agent only.

Let us now trace the steps of the algorithm. There are two inputs to the algorithm: (1) a violated commitment C and (2) the (initially empty) set of commitments Δ that are diagnosed on so far. When an agent A is requested to diagnose on commitment C, it proceeds according to the following five conditions:

1. *No commitment found that is relevant to* C. This means that the debtor does not infer the creditor's commitment at all (line 1)[3]. Diagnosis fails (line 2).
2. *An active commitment found that is a forward-shift of* C. This means that there is a misalignment between the creditor's and the debtor's copy of the same commitment (line 3). Diagnosis returns the debtor's copy as a candidate for alignment (line 4).
3. *No active commitment found that is a forward-shift of* C *and no commitment found that is a delegation of* C. This means that the debtor also infers the creditor's violated commitment (line 5). Diagnosis returns the debtor as a culprit (line 6).

[3] Note that each check excludes Δ from $\mathcal{C}_A^C$.

Algorithm 1. diagnose$_A(C, \varDelta)$: Diagnosis request to agent A about C

```
Input: $C \in \mathcal{C}$
Input: $\varDelta \subseteq \mathcal{C}$
Output: $\delta \in \{$failure, misbehavior($X$), misalignment($C'$)$\}$, $X \in \mathcal{A}$, $C' \in \mathcal{C}$
1  if $\mathcal{C}_A^C \setminus \varDelta = \emptyset$ then
2      $\delta$=failure;
3  else if $\exists C' \in \mathcal{C}_A^{Cf} \setminus \varDelta$ s.t. $active(C')$ then
4      $\delta$=misalignment($C'$);
5  else if $\forall X.\mathcal{C}_A^{CX} \setminus \varDelta = \emptyset \wedge \mathcal{C}_A^{CfX} \setminus \varDelta = \emptyset$ then
6      $\delta$=misbehavior($A$);
7  else if $\exists X$ s.t. $C' \in \mathcal{C}_A^{CfX} \setminus \varDelta \wedge active(C')$ then
8      $\delta$=misbehavior($A$);
9  else
10     $\delta$=diagnose$_X$($C'$,$\varDelta \cup \mathcal{C}_A^{CA}$) s.t. $C' \in \mathcal{C}_A^{CfX} \cup \mathcal{C}_A^{CX}$;
11 return $\delta$;
```

4. *An active commitment found that is a forward-shift delegation of C.* This means that the debtor has delegated its commitment without respecting its deadline (line 7). Diagnosis returns the debtor as a culprit (line 8).
5. *A violated commitment found that is either a delegation or a forward-shift delegation of C.* This means that the debtor has delegated its commitment, and that the delegated commitment is also violated (line 9). Diagnosis continues with the debtor of the delegated commitment (line 10).

The following properties hold for Algorithm 1:

Property 1: Algorithm 1 terminates. We consider two cases for termination: (1) there does not exist any circular chain of delegations, and (2) there exists a circular chain of delegations. Termination for the former case is trivial since the number of iterations is bounded with the number of agents in the system. For the latter case, consider the following circular chain of delegations among the commitments; c_1 = c(x, y, ...), c_2 = c(z, x, ...), ..., c_{n-1} = c(w, u, ...), c_n = c(y, w, ...). After each agent takes one diagnosis turn, agent y is requested to diagnose on commitment c_n. Now, y cannot request a further diagnosis from agent x on c_1 since c_1 is already contained in the set of commitments that are previously diagnosed on.

Property 2: Algorithm 1 makes a correct diagnosis. If the algorithm returns a misalignment, then a misalignment has occurred in the system. Similarly, if the algorithm returns a misbehavior, then a misbehavior has occurred in the system. However, note that the other direction is not always true. That is, if a misalignment has occurred in the system, the algorithm may return a misbehavior if it is also the case that a misbehavior has occurred prior to the misalignment. Similarly, if a misbehavior has occurred in the system, the algorithm may return a misalignment if it is also the case that a misalignment has occurred prior to the misbehavior. This is intuitive as we try to deal with the first possible reason for the exception.

Let us now discuss the outcomes of this diagnosis process in case of misbehavior or misalignment.

Property 3: Algorithm 1 finds the "culprit". This is true when a single misbehavior has occurred in the system, and no misalignments have occurred. In that case, Algorithm 1 returns an answer $\delta(X), X \in \mathcal{A}$, such that there exists an alternative possible trace of events in X's execution which will lead to no violation.

In the case of a misalignment, Algorithm 1 returns a commitment C' which is the reason of the misalignment. If that is the only reason of violation in the system (i.e., if there are no other misbehaviors nor misalignments), a simple way to achieve realignment is the following Policy $\mathcal{P}_1$. Agents following $\mathcal{P}_1$ will align their violated commitments with the one that is presented as the outcome of the diagnosis algorithm, by following these *commitment update rules*:

– *Alignment with forward-shift:* If the agent has commitment c_1, and the diagnosis process has proposed a commitment c_2 which is a forward-shift of c_1, then the agent will replace its commitment c_1 with c_2.
– *Alignment with forward-shift delegation:* If the agent has commitment $c_1 = s_1(c(x_1, y_1, property(e(t_1, t_2), p_1)))$, and the diagnosis process has proposed a commitment $c_2 = s_2(c(x_2, y_2, property(e(t_3, t_4), p_2)))$ which is a forward-shift delegation of c_1, then the agent will replace c_1 with $c_3 = s_2(c(x_1, y_1, property(e(t_3, t_4), p_2)))$.

The adoption of Policy $\mathcal{P}_1$ amounts to an implicit acceptance of a delayed commitment satisfaction.

Property 4: Algorithm 1 and Policy $\mathcal{P}_1$ provide a means of alignment. This is true when a misalignment has occurred in the system, and no misbehaviors have occurred prior to that misalignment. In that case, if all agents involved in the diagnosis process adopt $\mathcal{P}_1$, once Algorithm 1 terminates and all applicable $\mathcal{P}_1$ rules have been applied, there will be no more violated commitment in the system.

Example 7. Assume that the agent has $violated(c(store, customer, property\ (e(3.0, 8.0), delivered(book))))$, and it is presented with a forward-shift delegation of this commitment, $active(c(deliverer, store, property(e(7.0,10.0), delivered(book))))$. Then, the agent will update its commitment to $active(c(store, customer, property\ (e(7.0, 10.0), delivered(book))))$ following the rule for alignment with forward-shift delegation.

Example 7 describes a case where the store makes a delegation to the deliverer without respecting the deadline of the customer's commitment. In real life, in such cases, the customer often chooses to adopt to the new deadline discovered. This is a sort of *compensation* for the exception faced.

6 Case Study

Next, we present three separate traces of happened events from the delivery process, each leading to a different outcome of diagnosis. We assume that all agents adopt $\mathcal{P}_1$.

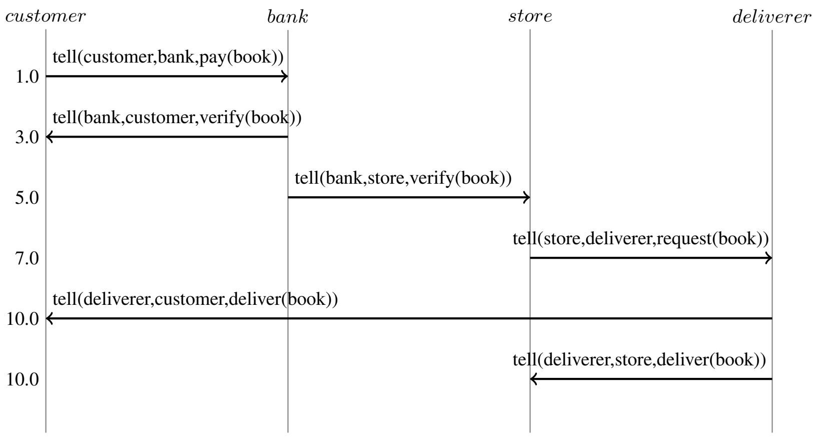

customer
bank
store
deliverer
tell(customer,bank,pay(book))
1.0
tell(bank,customer,verify(book))
3.0
tell(bank,store,verify(book))
5.0
tell(store,deliverer,request(book))
7.0
tell(deliverer,customer,deliver(book))
10.0
tell(deliverer,store,deliver(book))
10.0

Fig. 3. Trace of events for Case I

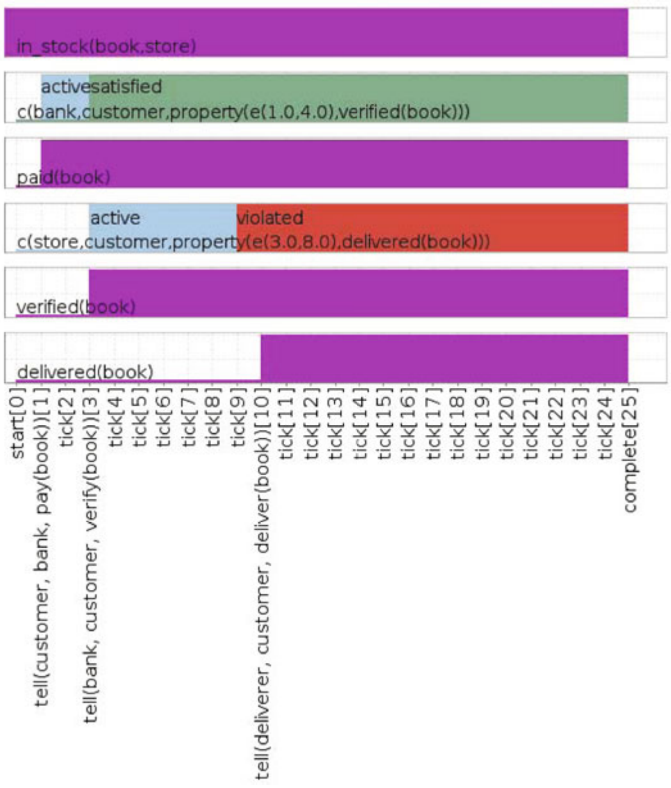


Fig. 4. $\mathcal{REC}$ output for customer's trace

6.1 Case I: Misalignment

Figure 3 shows the trace of events for the first case. The customer sends the payment to the bank at day 1, regarding its book purchase from the store. At day 3, the bank verifies the customer's payment. At day 5, the bank sends the notification to the store. Then, the store requests the delivery of the customer's book from the deliverer at day 7. Finally, the deliverer delivers the book to the customer at day 10. At the same time, the deliverer notifies the store about the delivery.

Let us track the commitments of the agents in time. At day 3, the customer infers c_1 = $active$ (c($store$, $customer$, $property$(e(3.0, 8.0), $delivered$ ($book$)))). At day 5, the store infers c_2 = $active$ (c($store$, $customer$, $property$(e(5.0, 10.0), $delivered$ ($book$)))). At day 7, both the store and the deliverer infer c_3 = $active$ (c($deliverer$, $store$, $property$(e(7.0, 10.0), $delivered$ ($book$)))). At the end of day 8, the customer detects that c_1 is violated. Figure 4 shows the output of $\mathcal{REC}$ for the customer's trace. Now, the customer initiates the diagnosis process with diagnose$_{store}$(c_1, $\emptyset$). Since the store has c_2, which is a forward-shift of c_1, he will immediately inform the customer of a misalignment (Lines 3-4 of Algorithm 1). The customer then updates its commitment, and waits for the new deadline. At day 10, the updated commitment is satisfied since the deliverer makes the delivery.

6.2 Case II: Misbehavior

Figure 5 shows the trace of events for the second case. Again, the customer sends the payment to the bank at day 1, regarding her purchase of the book from the store. At day 3, the bank verifies the customer's payment, and sends the notification to the store. The store requests the delivery of the customer's book from the deliverer at day 7. Finally, the deliverer delivers the book to the customer at day 10. At the same time, the deliverer notifies the store about the delivery.

Let us track the commitments of the agents in time. At day 3, both the customer and the store infer c_1 = $active$ (c($store$, $customer$, $property$(e(3.0, 8.0), $delivered$

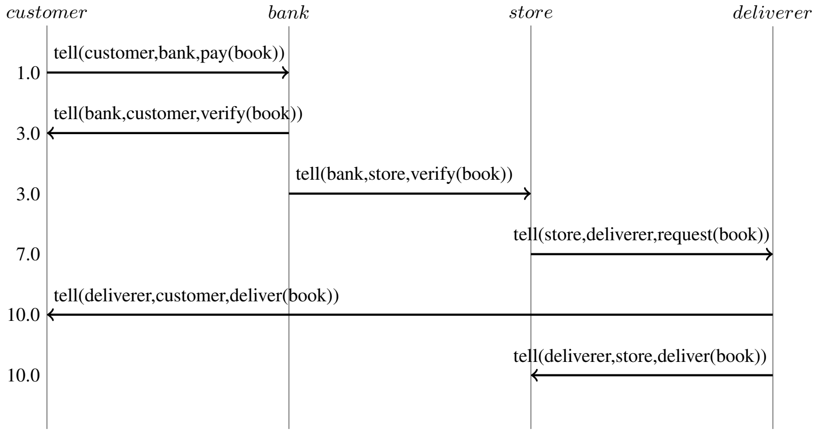

customer
bank
store
deliverer
tell(customer,bank,pay(book))
1.0
tell(bank,customer,verify(book))
3.0
tell(bank,store,verify(book))
3.0
tell(store,deliverer,request(book))
7.0
tell(deliverer,customer,deliver(book))
10.0
tell(deliverer,store,deliver(book))
10.0

Fig. 5. Trace of events for Case II

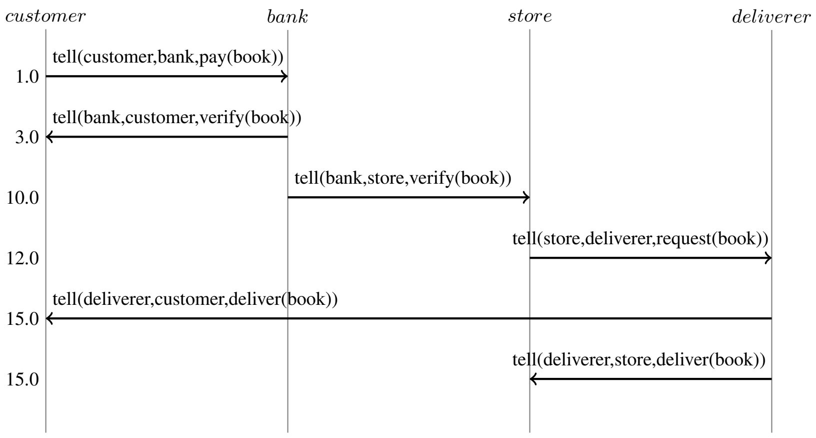

customer
bank
store
deliverer
tell(customer,bank,pay(book))
1.0
tell(bank,customer,verify(book))
3.0
tell(bank,store,verify(book))
10.0
tell(store,deliverer,request(book))
12.0
tell(deliverer,customer,deliver(book))
15.0
tell(deliverer,store,deliver(book))
15.0

Fig. 6. Trace of events for Case III

$(book)))$). At day 7, both the store and the deliverer infer c_2 = $active$ (c($deliverer$, $store$, $property$(e(7.0, 10.0), $delivered$ ($book$)))). At the end of day 8, the customer detects that c_1 is violated. So, she initiates the diagnosis process with the diagnosis request diagnose$_{store}$(c_1, $\emptyset$). Since the store now has c_2 (in addition c_1), which is a forward-shift delegation of c_1, he will inform the customer of a misbehavior of himself (Lines 7-8 of Algorithm 1).

6.3 Case III: Failure

Figure 6 shows the trace of events for the third case. The customer sends the payment to the bank at day 1, regarding its book purchase from the store. At day 3, the bank verifies the customer's payment. At day 10, the bank sends the notification to the store. Then, the store requests the delivery of the customer's book from the deliverer at day 12. Finally, the deliverer delivers the book to the customer at day 15. At the same time, the deliverer notifies the store about the delivery.

Let us track the commitments of the agents in time. At day 3, the customer infers c_1 = $active$ (c($store$, $customer$, $property$(e(3.0, 8.0), $delivered$ ($book$)))). At the end of day 8, the customer detects that c_1 is violated. So, she initiates the diagnosis process with the diagnosis request diagnose$_{store}$(c_1, $\emptyset$). Since the store has no commitments that are relevant to c_1, he will signal a failure (Lines 1-2 of Algorithm 1).

7 Discussion and Future Work

In this paper, we have mainly studied diagnosis of exceptions when the commitments of agents are misaligned with each other. Among the set of possible causes for misalignment [3,6], we are interested in the temporal aspects. That is, we aimed at fixing misalignments that are caused by conflicts in the commitments' deadlines. We have argued that a conflict of deadlines among two relevant commitments may be caused either by individual observations of agents that are in conflict with each other (i.e., misalignment), or by a delegation that does not respect a previously established deadline (i.e., misbehavior). We have proposed a commitment similarity relation that can be used to verify if two commitments are aligned in time. In the case of misalignment, the agents can update their commitments based on the alignment policy we have proposed. Providing an update of contract deadlines is an effective way of compensation that mimics real-life situations very closely. While this constitutes one step of diagnosis, we also provide the culprit agent in the case of misbehavior.

The most similar work in commitment alignment to ours is that of Chopra and Singh's [3]. The key points regarding our formalization and theirs are:

- There are no temporal constraints on commitments in Chopra and Singh's formalization. However, without an explicit notion of time, it is hard to capture the scenarios that are presented in this paper.
- Chopra and Singh propose a *strength* relation for commitments based on their properties (conditions and propositions). Currently, we consider only base-level commitments with single properties. However, we focus on the *similarity* relation for commitments since it provides a basis for verifying alignment. On one hand, the similarity relation takes into account the deadlines associated with commitments when verifying alignment in time. On the other hand, it takes into account the agents associated with commitments when tracing for delegations.
- Chopra and Singh propose a solution for misalignment by ensuring that the creditor of a commitment informs the debtor when the condition of the commitment is brought about. So, the debtor of the commitment will also infer the same base-level commitment the creditor infers. We believe that this solution may not be feasible for large-scale e-commerce applications. Most of the time, the execution will proceed as desired, and the agents will infer the same commitments. Thus, it is more reasonable to verify alignment if something goes wrong (i.e., in the case of an exception). Moreover, when deadlines are involved, a delay in such a notification message will also cause a similar misalignment between the debtor and the creditor's individual commitments.

In order to demonstrate how our approach works, we have extended a delivery process description [4] by involving temporal constraints, and formalized it in $\mathcal{REC}$ [1,8]. We have designed and presented three different exception cases according to the three possible outcomes of our diagnosis algorithm. For future work, we plan to extend our commitment similarity relation to cover the *strength* relation of Chopra and Singh. We also plan to investigate cases where multiple delegations are possible for the same commitment. Another possible direction for future work is to decide what to do next with the culprit agent identified (e.g., recovery). We are currently working on how to proceed with such diagnosis via the exchange of happened events. That is, the agents should reason both on the similarity among events and the relevance between commitments and events in order to find a suitable recovery.

Acknowledgement

The first author is supported by Boğaziçi University Research Fund (BAP09A106P), and The Scientific and Technological Research Council of Turkey (105E073).

References

1. Chesani, F., Mello, P., Montali, M., Torroni, P.: Commitment tracking via the reactive event calculus. In: IJCAI 2009: Proceedings of the 21st International Joint Conference on Artifical Intelligence, pp. 91–96 (2009)
2. Chittaro, L., Montanari, A.: Temporal representation and reasoning in artificial intelligence: Issues and approaches. Ann. Math. Artif. Intell. 28(1-4), 47–106 (2000)
3. Chopra, A.K., Singh, M.P.: Multiagent commitment alignment. In: AAMAS 2009: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, pp. 937–944 (2009)
4. Kafalı, Ö., Yolum, P.: Detecting exceptions in commitment protocols: Discovering hidden states. In: Languages, Methodologies and Development Tools for Multi-Agent Systems Workshop, MALLOW 2009 (2009)
5. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing 4(1), 67–95 (1986)
6. Schroeder, M., Schweimeier, R.: Arguments and misunderstandings: Fuzzy unification for negotiating agents. Electronic Notes in Theoretical Computer Science 70(5), 1–19 (2002)
7. Singh, M.P.: An ontology for commitments in multiagent systems: Toward a unification of normative concepts. Artificial Intelligence and Law 7, 97–113 (1999)
8. Torroni, P., Chesani, F., Mello, P., Montali, M.: Social commitments in time: Satisfied or compensated. In: Baldoni, M., Bentahar, J., van Riemsdijk, M.B., Lloyd, J. (eds.) DALT 2009. LNCS, vol. 5948, pp. 228–243. Springer, Heidelberg (2010)
9. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: applying event calculus planning using commitments. In: AAMAS 2002: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 527–534 (2002)

Verifying Business Process Compliance by Reasoning about Actions*

Davide D'Aprile[1], Laura Giordano[1], Valentina Gliozzi[2], Alberto Martelli[2], Gian Luca Pozzato[2], and Daniele Theseider Dupré[1]

[1] Dipartimento di Informatica, Università del Piemonte Orientale
{davide.daprile,laura.giordano,dtd}@mfn.unipmn.it
[2] Dipartimento di Informatica, Università di Torino
{gliozzi,mrt,pozzato}@di.unito.it

Abstract. In this paper we address the problem of verifying business process compliance with norms. To this end, we employ reasoning about actions in a temporal action theory. The action theory is defined through a combination of Answer Set Programming and Dynamic Linear Time Temporal Logic (DLTL). The temporal action theory allows us to formalize a business process as a temporal domain description, possibly including temporal constraints. Obligations in norms are captured by the notion of commitment, which is borrowed from the social approach to agent communication. Norms are represented using (possibly) non monotonic causal laws which (possibly) enforce new obligations. In this context, verifying compliance amounts to verify that no execution of the business process leaves some commitment unfulfilled. Compliance verification can be performed by Bounded Model Checking.

1 Introduction

Verifying the compliance of business processes towards normative regulations has become an important issue to be addressed. Many organizations (banks, hospitals, public administrations, etc.), whose activities are subject to regulations are required to justify their behaviors with respect to the norms and to show that the business procedures they adopt conform to such norms. In the financial domain, in particular, the Sarbanes-Oxley Act (commonly named SOX), enacted in 2002 in the USA, describes mandates and requirements for financial reporting, and was proposed in order to restore investor confidence in capital markets after major accounting scandals. MiFID (Markets in Financial Instruments Directive) is a EU law, effective from 2007, with similar goals, including transparency.

In this paper, in order to address the problem of business process compliance verification, we introduce a language for reasoning about action which extends Answer Set Programming. Temporal logic can be usefully exploited both in the

* This work has been partially supported by Regione Piemonte, Project "ICT4LAW - *ICT Converging on Law: Next Generation Services for Citizens, Enterprises, Public Administration and Policymakers*".

specification of an action domain and in the verification of its properties (see, e.g., [15]). In this paper, we provide a way to specify a business process as a temporal action domain and then we reason about it in the temporal action theory. The same formalism is used for the representation of both processes and norms towards which the process has to be compliant. In particular, causal laws of the action theory are well suited to model norms as directional rules, and defeasible negation of ASP can be exploited to model exceptions to the norms, by allowing norms to be defeasible. To represent the obligations which can be enforced by the application of norms, we make use of a notion of commitment, which is borrowed from the area of multi-agent communication [24,9,15].

For the specification and verification of business processes, we rely on a Temporal Action Theory [13], which combines Answer Set Programming with Dynamic Linear Time Temporal Logic (DLTL) [20]. DLTL extends propositional temporal logic of linear time with regular programs of propositional dynamic logic, that are used for indexing temporal modalities. The action language allows for general temporal constraints to be included in the domain description. The definition of action theories based on ASP presents several advantages over the approach in [14], which is based on a monotonic solution for the frame problem. First, the adoption of a non-monotonic solution to the frame problem, based on ASP default negation, allows to avoid the limitation of the completion solution in [14], which requires action and causal laws to be stratified to avoid unexpected extensions in case of cyclic dependencies. Second, ASP allows for a simple definition of defeasible action laws and defeasible causal laws, using default negation. Defeasibility of causal laws is needed if they are used to model norms with exceptions. Finally, bounded model checking [4] can be used for the verification of temporal properties of domain descriptions in temporal ASP. The approach developed in [19] for bounded LTL model checking with Stable Models, has been extended in [13] to deal with DLTL bounded model checking.

Given the specification of a business process as an action domain and the specification of norms as a set of (defeasible) causal rules generating commitments, the problem of compliance verification consists in verifying that there is no execution of the business process which leaves some commitment unfulfilled. This verification can be done using bounded model checking thechniques.

2 Running Example

As a running example we consider a fragment of the business process of an investment firm, where the firm offers financial instruments to an investor. The description of the business process in YAWL is given in Figure 1. We chose YAWL (Yet Another Workflow Language) [25] as specification language for our running business process example, since it provides a number of advantages with respect to several available alternatives:

– YAWL has been implemented in an open source workflow system and can be seen as a reference implementation of the workflow patterns (the outcome of an analysis activity based on business process modeling practice) [26].

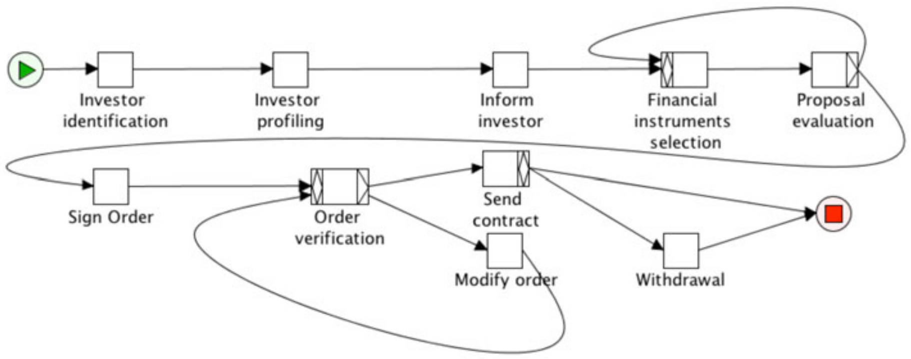


Fig. 1. Example business process in YAWL

- It is the most powerful business process modeling language, with respect to control-flow, data and resource perspectives, the three orthogonal views in a business process specification.
- It has been defined free from commercial interests.
- It provides a graphical user interface, based on a few basic elements, for business process specification needs; this implies a better learning curve.
- It is heavily XML-based, which facilitates interoperability.
- It comes with a formal foundation, which gives the possibility to perform formal analysis for achieving validation and verification goals.

Let us consider a regulation containing the following norms:

(1) the firm shall provide to the investor adequate information on its services and policies before any contract is signed;
(2) if the investor signs an order, the firm is obliged to provide him a copy of the contract.

The execution of each task in the process has some preconditions and effects. Due to the presence of norms, the execution of a task in the process above may generate obligations to be fulfilled. For instance, according to the second norm, signing an order generates for the firm the obligation to provide copy of the contract to the investor.

Verifying the compliance of a business process to a regulation requires to check that, in all the executions of the business process, the obligations triggered by the norms are fulfilled.

In the following, we provide the specification of the business process and of the related norms in an action theory. The problem of verifying compliance of the business process to the norms is then defined as a reasoning problem in the action theory. We first introduce the action language used, which is based on a temporal extension of answer set programming.

3 Action Theories in Temporal ASP

A domain description is defined as a set of laws describing the effects of actions as well as their executability preconditions. Actions may have direct effects, that are described by action laws, and indirect effects, that capture the causal dependencies among fluents and are described by causal laws. The execution of an action a in a state s leads to a new state s' in which the effect of the action holds. The properties (fluent) which hold in s and are not affected by the action a, still hold in s'. Let us first describe the notions of fluent and fluent literal.

Let $\mathcal{P}$ be a set of atomic propositions, the *fluent names*. A *simple fluent literal* l is a fluent name f or its negation $\neg f$. Given a fluent literal l, such that $l = f$ or $l = \neg f$, we define $|l| = f$. We will denote by Lit the set of all simple fluent literals. In the language we also make use of temporal literal, that is literals that are prefixed by temporal modalities, as $[a]l$ and $\bigcirc l$. Their intended meaning is the following: $[a]l$ holds in a state when l holds in the state obtained after the execution of action a; $\bigcirc l$ holds in a state if l holds in the next state.

Lit_T is the set of *(temporal) fluent literals*: if $l \in Lit$, then $l \in Lit_T$; if $l \in Lit$, then $[a]l, \bigcirc l \in Lit_T$ (for $a \in \Sigma$, the set of actions). Given a (temporal) fluent literal l, *not* l represents the default negation of l. A (temporal) fluent literal possibly preceded by a default negation, will be called an *extended fluent literal.*

A *domain description* D is defined as a tuple $(\Pi, Frame, \mathcal{C})$, where Π contains *action laws*, *causal laws*, *precondition laws* and the *initial state*, $Init$; $Frame$ provides a classification of fluents as frame fluents and non-frame fluents; $\mathcal{C}$ is a set of *temporal constraints.*

The *action laws* in Π have the form:

$$\Box([a]l_1 \ or \ldots or \ [a]l_k \leftarrow l'_1 \wedge \ldots \wedge l'_m)$$

where $l_1, \ldots, l_m$ and $l'_1, \ldots, l'_k$ are simple fluent literals. Its meaning is that executing action a in a state in which the conditions $l'_1, \ldots, l'_m$ hold causes either the effect l_1 or ... or the effect l_k to hold. Consider, for instance, the nondeterministic action of $order\_verification(T, C)$, which checks if the order of the financial product T by customer C is correct or not. In the first case, the order is accepted, otherwise it is not:

$$\Box([order\_verification(T, C)]confirmed(T, C) \ or$$
$$[order\_verification(T, C)]\neg confirmed(T, C)$$

In case of deterministic actions, there is a single disjunct in the head of the action law. For instance, the action of informing the investor has the effect that the investor has acquired information:

$$\Box([inform(C)]informed(C)$$

Causal laws are intended to express "causal" dependencies among fluents. *Static Causal laws* in Π have the form:

$$\Box(l \leftarrow l_1 \wedge \ldots \wedge l_m \wedge not\ l'_1 \wedge \ldots \wedge not\ l'_r)$$

where $l, l_1, \ldots, l_m$ are simple fluent literals. Their meaning is that: if $l_1, \ldots, l_m$ hold in a state, l is also caused to hold in that state. For instance,

$$\Box(\neg order\_confirmed(T, C) \leftarrow order\_deleted(T, C))$$

where "confirmed" means "confirmed by the firm" and is a possible effect of order verification, while "deleted" means "withdrawn by the customer", models the fact that the direct effect "deleted" of withdrawal has the indirect effect of making the order no longer effective for the firm as well.

Dynamic causal laws in Π have the form:

$$\Box(\bigcirc l \leftarrow l_1, \ldots, l_m, \bigcirc l_{m+1}, \ldots, \bigcirc l_k)$$

meaning that: if $l_1, \ldots, l_m$ hold in a state and $l_{m+1}, \ldots, l_k$ hold in the next state, then l is caused to hold in the next state.

Precondition laws have the form:

$$\Box([a]\bot \leftarrow l_1, \ldots, l_m)$$

with $a \in \Sigma$ and $l_1, \ldots, l_k$ are simple fluent literals. The meaning is that the execution of an action a is not possible if $l_1, \ldots, l_k$ hold (that is, no state results from the execution of a in a state in which $l_1, \ldots, l_k$ holds). An action for which there is no precondition law is always executable. The precondition law

$$\Box([proposal\_evaluation(T, C)]\bot \leftarrow \neg selected(T, C) \vee \neg informed(C))$$

states that an investor can be requested to evaluate a proposed investment only if the proposal has been selected and the investor has been already informed of the firm policy. Similar preconditions can either be asserted in the model, or verified to be true. The second option is suitable for the case where the process explicitly includes, as in figure 1, activities that make the precondition true; the first one is suitable for the case where such activities are abstracted away.

The *initial state*, *Init*, is a (possibly incomplete) set of simple fluent literals, the fluent which are known to hold initially. For instance, $Init = \{investor, \neg informed, \neg signed, etc.\}$.

The *temporal constraints* in $\mathcal{C}$ are arbitrary temporal formulas of DLTL. They are used to restrict the space of the possible extensions. DLTL [20] extends LTL by allowing the until operator $\mathcal{U}^\pi$ to be indexed by a program π, an expression of Propositional Dynamic Logic (PDL). The usual LTL modalities $\Box$ (always), $\Diamond$ (eventually) and $\mathcal{U}$ (until) can be defined from $\mathcal{U}^\pi$ as well as the new temporal modalities $[\pi]$ and $\langle\pi\rangle$. Informally, a formula $[\pi]\alpha$ is true in a world w of a linear temporal model if α holds in all the worlds of the model which are reachable from w through any execution of the program π. A formula $\langle\pi\rangle\alpha$ is true in a world w of a linear temporal model if there exists a world of the model reachable from w through an execution of the program π, in which α holds. A formula $\alpha\mathcal{U}^\pi\beta$ is true at τ if "α until β" is true on a finite stretch of behavior which is in the linear time behavior of the program π. The program π can be any regular expression built from atomic actions using sequence (;), non-deterministic choice (+) and finite iteration ($*$).

As an example of temporal constraint, $\neg sent\_contract \ \mathcal{U} \ signed$ states that the contract is not sent to the customer until it has been signed. A temporal constraint can also require a complex behavior to be performed, through the specification of a program. For instance (the complete version of the program for the process in figure 1 will be given in Section 4), the program

$$\begin{aligned}\pi = \ & inform(C); select\_financial\_instrument(T, C);\\ & ((sign\_order(T, C); send\_contract) + withdraw(T, C))\end{aligned}$$

describes a process in which: the investor C is informed, a financial instrument T is selected for C, C either signs the contract and a copy of the contract is set to him, or C withdraws. The formula $\langle\pi\rangle true$ requires that there is an execution of the program π starting from the initial state.

As in [23,22] we call *frame* fluents those fluents to which the law of inertia applies. We consider frame fluents as being dependent on the actions. *Frame* is a set of pairs (p, a), where $p \in \mathcal{P}$ is a fluent and $a \in \Sigma$, meaning that p is a frame fluent for action a, that is, p is a fluent to which persistency applies when action a is executed. Instead, non-frame fluents with respect to a do non persist and may change value in a non-deterministically, when a is executed.

Unlike [14], we adopt a non-monotonic solution to the frame problem, as usual in the context of ASP. The persistency of frame fluents from a state to the next one can be enforced by introducing *persistency laws* of the form:

$$\Box([a]l \leftarrow l, not\ [a]\neg\, l),$$

for each simple fluent literal l and action $a \in \Sigma$, such that $(|l|, a) \in Frame$. Its meaning is that, if l holds in a state, then l holds in the state obtained by executing action a, if it can be assumed that $\neg l$ does not hold in the resulting state.

For capturing the fact that a fluent literal l which is non-frame with respect to $a \in \Sigma$ may change its value non-deterministically when a is executed, we introduce the axiom:

$$\Box([a]p\ or\ [a]\neg p \leftarrow true)$$

for all p and a such that $(p, a) \notin Frame$. When a is executed, either the non-frame fluent literal p holds in the resulting state, or $\neg p$ holds. We will call $Frame_D$ the set of laws introduced above for dealing with frame and non-frame fluents. They have the same structure as action laws, but frame axioms contain default negation in their bodies. Indeed, both action laws and causal laws can be extended for free by allowing default negation in their body. This extension has been considered for instance in [8].

As concerns the initial state, we assume that its specification is, in general, incomplete. However, we reason on complete initial states obtained by completing the initial state in all the possible ways. and we assume that, for each fluent literal p, the domain description contains the law:

$$p\ or\ \neg p \leftarrow true$$

meaning that either p is assumed to hold in the initial state, or $\neg p$ is assumed to hold. This approach is in accordance with our treatment of non-deterministic actions and, as we will see, gives rise to extensions in which all states are complete, where each extension represents a run, i.e. a possible evolution of the world from the initial state. We will call $Init_D$ the set of laws introduced above for completing the initial state in all the possibile ways.

4 Specifying a Business Process as an Action Domain

In the following, we provide the specification of a business process as an action domain description. In the following, *profiling* stands for *investor profiling*, *inform* stands for *inform investor*, *fi_selection* stands for *financial instrument selection*, *p_eval* stands for *proposal evaluation*, *order_verif* stands for *order verification*. The following action and causal laws describe the effect of the actions in the process:

$$\Box([investor\_identification(C)]investor(C))$$
$$\Box([profiling(C)]investor\_classified(C,D))$$
$$\Box([profiling(C)](risk\_averse(C)\ or\ risk\_neutral(C)\ or\ risk\_seeking(C)))$$
$$\Box([inform(C)]informed(C))$$
$$\Box([fi\_selection(t_1,C)]selected(t_1,C)\ or \ldots$$
$$\ldots or\ [fi\_selection(t_n,C)]selected(t_n,C) \leftarrow$$
$$financial\_instr(t_1) \wedge \ldots \wedge financial\_instr(t_n) \wedge risk\_averse(C))$$
$$\ldots$$
$$\Box([p\_eval(T,C)]accepted(T,C)\ or\ [p\_eval(T,C)]\neg accepted(T,C))$$
$$\Box([sign\_order(T,C)]order\_signed(T,C))$$
$$\Box([order\_verif](T,C)]order\_confirmed(T,C)\ or$$
$$[order\_verif](T,C)]\neg order\_confirmed(T,C))$$
$$\Box([send\_contract(T,C)]sent\_contract(T,C))$$
$$\Box([withdraw(T,C)]order\_deleted(T,C))$$
$$\Box(\neg order\_confirmed(T,C) \leftarrow order\_deleted(T,C))$$
$$\Box([end\_procedure]end)$$

The following precondition laws can either be asserted or verified (see sect. 3):

$$\Box([p\_eval(T,C)]\bot \leftarrow \neg selected(T,C) \vee \neg informed(C))$$
$$\Box([send\_contract(T,C)]\bot \leftarrow \neg confirmed(T,C))$$

The meaning of the second one is that it is possible to send a contract to the investor only if the contract has been confirmed.

In order to specify business processes as programs, we introduce *test actions*. DLTL does not include test actions. We define test actions as atomic actions with no effects. For example, $accepted(T,C)?$ is an action testing the result of proposal evaluation in figure 1. Preconditions of $accepted(T,C)?$ are given by the following precondition law:

$$\Box([accepted(T,C)?]\bot \leftarrow \neg accepted(T,C)$$

stating that $accepted(T, C)?$ is not executable in a state in which the investor has not accepted the proposal.

A loop **repeat** *activity* **until** *test* can then be written as follows:

$activity;$
$(\neg test?; activity;)^*$
$test?;$

Note that a regular expression e^* represents the *infinite* set of strings where each string is formed by a *finite* number of occurrences of e. Therefore, only the (infinite) set of finite executions of the loop is represented. This interpretation is consistent with the combination of "classical soundness" and "strong fairness" in Workflow Nets, the class of Petri Nets which form the basis for the semantics of YAWL; see, e.g., the comments in [27] after Definition 8 ("without this assumption, all nets allowing loops in their execution sequences would be called unsound, which is clearly not desirable.").

The control flow of the process in figure 1, which is defined quite rigidly, can be modeled by the following program π:

$investor\_identification(C);$
$profiling(C);$
$inform(C);$
$fi\_selection(T, C);$
$p\_eval(T, C);$
$(\neg accepted(T, C)?; fi\_selection(T, C); p\_eval(T, C))^*;$
$accepted(T, C)?;$
$sign\_order(T, C);$
$order\_verif(T, C);$
$(\neg order\_confirmed(T, C)?; modify\_order(T, C); order\_verif(T, C))^*;$
$order\_confirmed(T, C)?;$
$send\_contract(T, C);$
$(withdraw(T, C) + skip);$
$end\_procedure$

where the action *skip* is defined as the empty action, with no effect.
Given the specification of the program π given above, we introduce the following constraints in $\mathcal{C}$:

$\langle \pi \rangle True$

meaning that in each extension of the domain description, there exists a state reachable from the initial one through the execution of the program π (in which of course $True$ holds). Namely, the sequence of the actions executed must start with an execution of the program π.

The approach we adopt in this paper for reasoning about actions is well suited for reasoning about infinite action sequences. To deal with finite computations we introduce a dummy action, which can be repeated infinitely many times after the termination of the process (thus giving rise to an infinite computation). We introduce the following constraints:

$$\Diamond\langle dummy\rangle True$$
$$\Box[dummy]\langle dummy\rangle True$$

stating that: the dummy action must eventually be executed and, from that point on, the dummy action is executed repeatedly. The precondition law:

$$\Box[dummy]\bot \leftarrow \neg end$$

states that the dummy action cannot be executed if the process has not reached the end point.

Although the above specification of the process is very rigid, as it is given through the program expression π, in general a more flexible specification can be provided by encoding the control flow of the process through the specification of action effects and preconditions (as done, for instance, for flexibly encoding agent protocols in [15]). This is, in general, the approach for translating YAWL processes into a domain description with action effects and preconditions, even though in this paper we do not propose an approach to cover all the expressiveness of the YAWL language.

5 Normative Specification

As we have seen in the previous section, the action theory provides a specification of the business process which allows the effect of atomic tasks in the process to be made explicit. According to the normative specification, the execution of each task in the business process can, in addition, trigger some normative position (obligation, permission, prohibition). For instance, as said above, the *identification* task in the business process in Figure 1, which introduces a new investor C, also generates the obligation to inform the investor. This obligation must be fulfilled during the course of execution of the business process, if the process is compliant with the norm stating that the firm has the obligation to inform customers.

In the following we make use of causal laws to represent norms in the action theory, and we introduce a notion of commitment to model obligations. The use of commitments has long been recognized as a "key notion" to allow coordination and communication in multi-agent systems [21]. Their use in social approaches to agent communication is essentially motivated by requirements of verifiability. Among the most significant proposals to use commitments in the specification of protocols (or more generally, in agent communication) are [24,18,9]. A notion of commitment for reasoning about agent protocols in a temporal action logic have been proposed in [15]. In [1] an alternative notion to commitment called *expectation* is proposed. We refer to section 7 for a discussion of this approach.

Following [15], we introduce two kinds of commitments (which are regarded as special fluent propositions): *Base-level commitments* having the form $C(i, j, A)$ and meaning that agent i is committed to agent j to bring about A (where A is an arbitrary propositional formula not containing commitment fluents); *Conditional commitments* having the form $CC(i, j, B, A)$ and meaning that agent i is committed to agent j to bring about A, if condition B is brought about.

A base level commitment $C(i, j, A)$ can be naturally regarded as an obligation (namely, $O\ A$, "A is obligatory"), in which the debtor and the creditor are made explicit. The two kinds of base-level and conditional commitments we use here are essentially those introduced in [29]. Our present choice is different from the one in [18], where agents are committed to execute an action rather than to achieve a condition.

The idea is that commitments (or obligations) are created as effects of the execution of some basic task in the business process and they are "discharged" when they have been fulfilled. A commitment $C(i, j, A)$, created at a given state of a run of the process, is regarded to be fulfilled in the run if there is a later state of the run in which A holds. As soon as committment is fullfilled in a run, it is considered to be satisfied and no longer active: it can be discharged.

Given the notion of commitment introduced above, norms can be modeled as precondition and causal laws which trigger new commitments/obligations. For instance, we can encode the norms in Section 2 by the following precondition and causal laws:

$$\Box([sign\_order(T, C)]\bot \leftarrow \neg informed(C))$$
$$\Box(C(firm, C, sent\_contract(T, C)) \leftarrow order\_signed(T, C))$$

The first one is a precondition for $sign\_order(T, C)$, and it is quite obviously true in the example process model, because $informed(C)$ is the effect of the action $inform(C)$ which is always executed before $sign\_order(T, C)$ is reached (and there is no action making $informed(C)$ false). Verifying preconditions may be more interesting in more complex processes where the action may be reached via several paths.

The second one, a causal law, states that when an order is signed by C, the firm is committed to C to send her the information required. The commitment remains active until some action is executed, which makes $sent\_contract(T, C)$ true. In the business process, the commitment is fulfilled by the execution of the action $send\_contract(T, C)$.

Causal laws are needed for modeling the interplay of commitments. In particular, for each commitment $C(i, j, \alpha)$, we introduce the following causal laws in the domain description:

(i) $\Box(\bigcirc\neg C(i, j, \alpha) \leftarrow C(i, j, \alpha) \wedge \bigcirc\alpha)$
(ii) $\Box(\bigcirc C(i, j, \alpha) \leftarrow CC(i, j, \beta, \alpha) \wedge \bigcirc\beta)$
(iii) $\Box(\bigcirc\neg CC(i, j, \beta, \alpha) \leftarrow CC(i, j, \beta, \alpha) \wedge \bigcirc\beta)$

A commitment to bring about α is considered fulfilled and is discharged (i) as soon as α holds. A conditional commitment $CC(i, j, \beta, \alpha)$ becomes a base-level commitment $C(i, j, \alpha)$ when β has been brought about (ii) and, in that case, the conditional commitment is discharged (iii).

One of the central issues in the representation of norms comes from the defeasible nature of norms. Norms may have exceptions: recent norms may cancel older ones; more specific norms override more general norms; and in other cases, explicit priority information (not necessarily related to recency or specificity) is needed for disambiguation. Consider the following example from [17]:

$$r_1\colon C(S,M,O,\mathit{discount}) \leftarrow \mathit{sells}(S,M,O) \wedge \mathit{premium\_customer}(M)$$
$$r_2\colon \neg C(S,M,O,\mathit{discount}) \leftarrow \mathit{sells}(S,M,O) \wedge \mathit{special\_order}(S,M,O)$$

Rule r_1 says that a seller has the obligation to apply a discount to premium customers. Rule r_2 says that premium customer are not entitled for a discount in case the order (O) is a special order. Suppose that rule r_2 is explicitly given priority over r_1 ($r_2 > r_1$). The priority between the conflicting norms r_1 and r_2, with $r_2 > r_1$, can be modeled using default negation. For instance, we can transform the rules r_1 and r_2 as follows:

$$\Box(C(s,m,o,\mathit{discount}) \leftarrow \mathit{sells}(s,m,o) \wedge \mathit{premium}(m) \wedge \mathit{not}\ \mathit{bl}(r_1))$$
$$\Box(\neg C(s,m,o,\mathit{discount}) \leftarrow \mathit{sells}(s,m,o) \wedge \mathit{special\_order}(c) \wedge \mathit{not}\ \mathit{bl}(r_2))$$
$$\Box(\mathit{bl}(r_1) \leftarrow \mathit{sells}(s,m,o) \wedge \mathit{special\_order}(c) \wedge \mathit{not}\ \mathit{bl}(r_2))$$
$$\Box(\neg \mathit{bl}(r_1) \leftarrow \mathit{not}\ \mathit{bl}(r_1))$$
$$\Box(\neg \mathit{bl}(r_2) \leftarrow \mathit{not}\ \mathit{bl}(r_2))$$

(where $\mathit{bl}(r_i)$ means that r_i is blocked) so that rule r_2, when applicable, blocks the application of r_1, but not vice-versa.

In the context of ASP, more general and complex encodings of prioritized rules into standard rules with default negation have been studied in [7] and in [5]. We expect that a similar approach can be exploited in this setting to model defeasible norms as prioritized defeasible causal laws.

Another issue to be addressed when modeling norms is that of formalizing violations and reparation obligations. When an obligation is violated, another obligation can be generated as a reparation of that violation. In [17] this problem is addressed through the definition of reparation chains $OA \otimes OB \otimes OC$, where OB is the reparation of the violation of OA, and OC is the reparation of the violation of OB, so that the rules representing norms can have a reparation chain in the conclusion. For instance, the norm $OPay\_in\_time \otimes OPay\_with\_Interest \leftarrow Invoice$ says that, after the invoice has been received, the obligation to pay in time is generated but, if this obligation is not fulfilled, the obligation to pay with interest is generated. We can represent this norm with the following laws:

$$r_1\colon C(i,j,Pay\_in\_time) \leftarrow Invoice(j,i)$$
$$r_2\colon C(i,j,Pay\_with\_Interest) \leftarrow$$
$$Invoice(j,i) \wedge C(i,j,Pay\_in\_time) \wedge \neg Pay\_in\_time$$
$$r_3\colon \neg C(i,j,Pay\_in\_time) \leftarrow$$
$$Invoice(j,i) \wedge C(i,j,Pay\_in\_time) \wedge \neg Pay\_in\_time$$

The first law states that after i receives the invoice from j, i is committed to j to pay in time; r_2 and r_3 state that, if there is a commitment to pay in time and it is violated, then this commitment is discharged and the commitment to pay with interest is generated. Rule r_3 has priority over r_1.

6 Temporal Answer Sets and Compliance Verification

Once the specification of the business process has been given as a domain description in an action theory, the problem of verifying its compliance with some regulation can be modeled as the problem of verifying that all the executions of the business process fulfill the obligations that are generated during the execution of the process. In order to characterize the executions of the business process and to check if they violate some obligation, we introduce the notion of extension of a domain description.

A domain description $D = (\Pi, Frame, \mathcal{C})$, is a general logic program extended with a restricted use of temporal modalities. The action modalities $[a]$ and $\bigcirc$ may occur in front of simple literals within rules and the $\Box$ modality occurs in front of all rules in Π. Following [13], we introduce a notion of *temporal answer set*, extending the notion of *answer set* [10]. The extensions of a domain description are then defined as the temporal answer sets of $\Pi \cup Frame_D \cup Init_D$ satisfying the integrity constraints $\mathcal{C}$.

We define a partial temporal interpretation S as a set of literals of the form $[a_1; \dots; a_k]l$ where $a_1, \dots, a_k \in \Sigma$, meaning that literal l holds in S in the state obtained by executing the actions $a_1, \dots, a_k$ in the order.

Definition 1. *Let $\sigma \in \Sigma^\omega$. A* partial temporal interpretation S over σ *is a set of temporal literals of the form $[a_1; \dots; a_k]l$, where $a_1 \dots a_k$ is a prefix of σ, and it is not the case that both $[a_1; \dots; a_k]l$ and $[a_1; \dots; a_k]\neg l$ belong to S (namely, S is a* consistent *set of temporal literals).*

We define a notion of *satisfiability of a literal l in a temporal interpretation S in the state $a_1 \dots a_k$* as follows. A literal l is *true* in a partial temporal interpretation S in the state $a_1 \dots a_k$ (and we write $S, a_1 \dots a_k \models_t l$), if $[a_1; \dots; a_k]l \in S$; a literal l is *false* in a partial temporal interpretation S in the state $a_1 \dots a_k$ (and we write $S, a_1 \dots a_k \models_f l$), if $[a_1; \dots; a_k]\overline{l} \in S$; and, finally, a literal l is *unknown* in a partial temporal interpretation S in the state $a_1 \dots a_k$ (and we write $S, a_1 \dots a_k \models_u l$), otherwise.

The notion of satisfiability of a literal in a partial temporal interpretation in a given state, can be extended to temporal literals and to rules in a natural way.

For temporal literals $[a]l$, we have: $S, a_1 \dots a_k \models_t [a]l$ if $[a_1; \dots; a_k; a]l \in S$ or $a_1 \dots a_k, a$ is not a prefix of σ; $S, a_1 \dots a_k \models_f [a]l$ if $[a_1; \dots; a_k; a]\overline{l} \in S$ or $a_1 \dots a_k, a$ is not a prefix of σ; and $S, a_1 \dots a_k \models_u [a]l$, otherwise.

For temporal literals of the form $\bigcirc l$: $S, a_1 \dots a_k \models_t \bigcirc l$ if $[a_1; \dots; a_k; b]l \in S$, for some b with $a_1 \dots a_k b$ prefix of σ; $S, a_1 \dots a_k \models_f \bigcirc l$ if $[a_1; \dots; a_k; b]\overline{l} \in S$, for some b with $a_1 \dots a_k b$ prefix of σ; $S, a_1 \dots a_k \models_u \bigcirc l$, otherwise.

For default negation, we have: $S, a_1 \dots a_k \models_t$ *not* l if $S, a_1 \dots a_k \models_f l$ or $S, a_1 \dots a_k \models_u l$; and $S, a_1 \dots a_k \models_f$ *not* l, otherwise.

The three valued evaluation of conjunctions and disjunctions of literals is defined as usual in ASP (see, for instance, [10]). Finally, we say that a rule $\Box(H \leftarrow Body)$ is satisfied in a partial temporal interpretation S if, for all action sequences $a_1 \dots a_k$ (including the empty one), $S, a_1 \dots a_k \models_t Body$ implies $S, a_1 \dots a_k \models_t H$. We say that a rule $[a_1; \dots; a_h](H \leftarrow Body)$, is satisfied in a partial temporal interpretation S if $S, a_1 \dots a_h \models_t Body$ implies $S, a_1 \dots a_h \models_t H$.

We are now ready to define the notion of answer set for a set of P of rules that do not contain default negation. Let P be a set of rules over an action alphabet Σ, not containing default negation, and let $\sigma \in \Sigma^\omega$.

Definition 2. *A partial temporal interpretation S over σ is a* temporal answer set of P *if S is minimal (in the sense of set inclusion) among the partial interpretations satisfying the rules in P.*

We want to define answer sets of a program P possibly containing negation. Given a partial temporal interpretation S over $\sigma \in \Sigma^\omega$, we define the *reduct, P^S, of P relative to S* extending the transformation in [10] to compute a different reduct of P for each prefix $a_1, \ldots, a_h$ of σ.

Definition 3. *The* reduct, $P^S_{a_1,\ldots,a_h}$, of P relative to S and to the prefix $a_1, \ldots, a_h$ of σ *, is the set of all the rules $[a_1; \ldots; a_h](H \leftarrow l_1 \wedge \ldots \wedge l_m)$, such that $\Box(H \leftarrow l_1 \wedge \ldots \wedge l_m \wedge not\ l_{m+1} \wedge \ldots \wedge not\ l_k)$ is in P and, for all $i = m+1, \ldots, k$, either $S, a_1, \ldots, a_h \models_f l_i$ or $S, a_1, \ldots, a_h \models_u l_i$, (where $l, l_1, \ldots, l_k$ are simple or temporal literals). The* reduct P^S of P relative to S over σ *is the union of all reducts $P^S_{a_1,\ldots,a_h}$ for all prefixes $a_1, \ldots, a_h$ of σ.*

Definition 4. *A partial temporal interpretation S over σ is an* answer set of P *if S is an answer set of the reduct P^S.*

The definition above is a natural generalization of the usual notion of answer set to programs with temporal rules. Observe that, σ has infinitely many prefixes, so that the reduct P^S is infinite and answer sets are infinite. This is in accordance with the fact that temporal models are infinite. Given to the laws for completing the initial state in $Init_D$, we can prove the following:

Proposition 1. *Given a domain description D over Σ and an infinite sequence σ, any answer set of $\Pi \cup Frame_D \cup Init_D$ over σ is a total answer set over σ.*

It can be shown that, by persistency laws, the execution of an action in a complete state produces a new complete state, which is only determined by the action laws, causal laws and persistency laws executed in that state.

In the following, we define the notion of *extension* of a domain description $D = (\Pi, Frame, \mathcal{C})$ over Σ in two steps: first, we find the answer sets of $\Pi \cup Frame_D \cup Init_D$; second, we filter out all the answer sets which do not satisfy the temporal constraints in $\mathcal{C}$. For the second step, we need to define when a temporal formula α is satisfied in a total temporal interpretation S. Observe that a total answer set S over σ can be regarded as a linear temporal (DLTL) model [20]. Given a total answer set S over σ we define the corresponding temporal model as $M_S = (\sigma, V_S)$, where $p \in V_S(a_1, \ldots, a_h)$ if and only if $[a_1; \ldots; a_h]p \in S$, for all atomic propositions p. We say that a total answer set S over σ satisfies a DLTL formula α if $M_S, \varepsilon \models \alpha$.

Definition 5. *An* extension of a domain description $D = (\Pi, Frame, \mathcal{C})$ over *Σ, is any (total) answer set S of $\Pi \cup Frame_D \cup Init_D$ satisfying the constraints in $\mathcal{C}$.*

Notice that, in general, a domain description may have more than one extension even for the same action sequence σ: the different extensions of D with the same σ account for the different possible initial states (when the initial state is incompletely specified) as well as for the different possible effects of nondeterministic actions.

The extensions of the domain description define all the possible executions of the business process. To check if there is an execution which violates some obligation we model the need to fulfil a commitment $C(i, j, \alpha)$ as the temporal formula:

$$\Box(C(i, j, \alpha) \rightarrow \Diamond\alpha)$$

Such formulae, together with the precondition formulae corresponding to norms, is the set of formulae to be verified in order to check compliance. We can then introduce the following definition.

Definition 6. *Let D_B the domain description providing the specification of a business process B and let P_N and C_N be, respectively, the set of precondition laws and causal laws in a set N of norms. The business process B is compliant with N if for each extension S of the domain description $D_B \cup C_N$:*

– *for each precondition law P in P_N, S satisfies P;*
– *for each commitment $C(i, j, \alpha)$ occurring in C_N, S satisfies the formula $\Box(C(i, j, \alpha) \rightarrow \Diamond\alpha)$.*

Consider the domain description $D_B \cup C_N$, including the specification D_B of the business problem example and the causal law

$$\Box(C(firm, C, sent\_contract(T, C)) \leftarrow order\_signed(T, C))$$

Each extension S of the domain description satisfies the temporal formulas

$$\Box(C(firm, C, sent\_contract(T, C)) \rightarrow \Diamond sent\_contract(T, C))$$
$$\Box([sign\_order(T, C)]\bot \leftarrow \neg informed(C))$$

Hence, the business process is compliant with the norms.

Observe that a weaker notion of compliance can be defined by weakening the fulfilment condition to $\Box(C(i, j, \alpha) \rightarrow \Diamond(\alpha \vee \neg C(i, j, \alpha))$, meaning that a commitment occurring on a run is weakly fulfilled if there ia a later state in which either α holds or the commitment has been discharged. This notion of fulfillment can be used to deal with reparation chains [17].

In [13] we describe bounded model checking techniques for computing the extensions of a temporal domain description and for verifying temporal properties of a domain description. More precisely, we describe a translation of a temporal domain description into standard ASP, so that the temporal answer sets of the domain description can then be computed as the standard answer sets of its translation. Temporal constraints, which are part of the domain description, are evaluated over temporal answer sets using *bounded model checking* techniques [4]. The approach proposed for the verification of DLTL formulas extends the one developed in [19] for bounded LTL model checking with Stable Models.

7 Conclusions and Related Work

The paper deals with the problem of verifying the compliance of business processes with norms. Our approach is based on a temporal extension of ASP for reasoning about actions. Both the business process and the norms are given a specification in a domain description. In particular, defeasible causal laws are used for modeling norms and commitments are introduced for representing obligations. The verification of compliance can be performed by using bounded model checking techniques, which generalize LTL bounded model checking [19].

Temporal rule patterns for regulatory policies are introduced in [12], where regulatory requirements are formalized as sets of compliance rules in a real-time temporal object logic. The proposed REALM approach is based on three central pillars: (1) the domain of discourse of a regulation is captured by a concept model expressed as UML class diagrams (concept model); (2) the regulatory requirements are formalized as a set of logical formulas expressed in a real-time temporal object logic, namely a combination of Alur and Henzinger's Timed Propositional Temporal Logic and many-sorted first-order logic (compliance rule set). (3) Information about the structure of the legal source as well as life-cycle data are captured as separate metadata, which allow to annotate both the concept model and the compliance rule set to ensure traceability from regulatory requirements to model elements and vice versa. The approach is used essentially for event monitoring.

[11] proposes an approach based on annotations of business process models. In detail, the tasks of a business process model in BPMN are annotated with their effects. A suitable mechanism to propagate and cumulate the effects of a task to next ones is presented. Process compliance verification establishes that a business process model is consistent with a set of compliance rules. This approach does not introduce normative concepts.

[17] proposes an approach to the problem of business process compliance based on the idea of annotating the business process. Process annotations and normative specifications are provided in the same logical language, namely, the Formal Contract Language (FCL). FCL combines defeasible logic [3] and deontic logic of violations [16] which allows to represent exceptions as well as to express reparation chains, to reason with violations, and the obligations resulting from violations. The notions of ideal, sub-ideal and non-ideal situations are defined to describe two degrees of compliance between execution paths and FCL constraints. Compliance is verified by traversing the graph describing the process and identifying the effects of tasks and the obligations triggered by the task execution. Algorithms for propagating obligations through the process graph are defined. Obligations are discharged when they are fulfilled. In this paper we provide a characterization the compliance problem as a problem of reasoning about actions, which provides business process specification, normative specification, and compliance verification in an integrated representation and reasoning framework.

In [28] process models are considered in which individual activities are annotated with logical preconditions and effects. A formal execution semantics for annotated business processes is introduced and several verification tasks are defined to check whether the business process control flow interacts correctly with the behaviour of the individual activities.

An approach to compliance based on a commitment semantics in the context of multi-agent systems is proposed in [6]. In this work, the authors formalize notions of conformance, coverage, and interoperability, proving that they are orthogonal to each other. The basic idea of the authors is that for an agent to be compliant with a protocol, it must be conformant with it, and conformance can be checked at design time. Another approach to the verification of agents compliance with protocols, based on a temporal action theory, has been proposed in [15]. These papers do not address the problem of compliance with norms.

The notion of *expectation*, which is alternative to the notion of commitment, has been proposed in [1] for the specification of agent protocols. Expectations are used for modelling obligations and prohibitions in the abductive computational framework SOCS [2]. Intuitively, obligations and prohibitions are mapped into abducible predicates, expressing positive and negative expectations, respectively; norms are formalized by abductive integrity constraints. The paper points out that the abductive proof procedure $\mathcal{S}$CIFF can be used for on-the-fly verification of agents conformance to norms.

References

1. Alberti, M., Daolio, D., Torroni, P., Gavanelli, M., Lamma, E., Mello, P.: Specification and Verification of Agent Interaction Protocols in a Logic-based System. In: SAC 2004, pp. 72–78 (2004)
2. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P., Sartor, G.: Mapping of Deontic Operators to Abductive Expectations. In: NORMAS, pp. 126–136 (2005)
3. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation results for defeasible logic. ACM Trans. on Computational Logic 2, 255–287 (2001)
4. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers 58, 118–149 (2003)
5. Brewka, G., Eiter, T.: Preferred answer sets for extended logic programs. Artificial Intelligence 109(1-2), 297–356 (1999)
6. Chopra, A.K., Sing, M.P.: Producing compliant interactions: Conformance, coverage and interoperability. In: Baldoni, M., Endriss, U. (eds.) DALT 2006. LNCS (LNAI), vol. 4327, pp. 1–15. Springer, Heidelberg (2006)
7. Delgrande, J.P., Schaub, T., Tompits, H.: A framework for compiling preferences in logic programs. Theory and Practice of Logic Programming 3(2), 129–187 (2003)
8. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: Planning under Incomplete Knowledge. In: Computational Logic, pp. 807–821 (2000)
9. Fornara, N., Colombetti, M.: Defining Interaction Protocols using a Commitment-based Agent Communication Language. In: AAMAS 2003, pp. 520–527 (2003)
10. Gelfond, M.: Answer Sets. In: Handbook of Knowledge Representation, ch. 7. Elsevier, Amsterdam (2007)
11. Ghose, A., Koliadis, G.: Auditing business process compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 169–180. Springer, Heidelberg (2007)
12. Giblin, C., Müller, S., Pfitzmann, B.: From Regulatory Policies to Event Monitoring Rules: Towards Model-Driven Compliance Automation. IBM Reasearch Report (2007)

13. Giordano, L., Martelli, A., Theseider Dupré, D.: Reasoning about Actions with Temporal Answer Sets. In: Proc. CILC 2010, 25th Italian Conference on Computational Logic (2010)
14. Giordano, L., Martelli, A., Schwind, C.: Reasoning About Actions in Dynamic Linear Time Temporal Logic. The Logic Journal of the IGPL 9(2), 289–303 (2001)
15. Giordano, L., Martelli, A., Schwind, C.: Specifying and Verifying Interaction Protocols in a Temporal Action Logic. Journal of Applied Logic (Special issue on Logic Based Agent Verification) 5, 214–234 (2007)
16. Governatori, G., Rotolo, A.: Logic of Violations: A Gentzen System for Reasoning with Contrary-To-Duty Obligations. Australasian Journal of Logic 4, 193–215 (2006)
17. Governatori, G., Sadiq, S.: The journey to business process compliance. In: Handbook of Research on BPM, pp. 426–454. IGI Global (2009)
18. Guerin, F., Pitt, J.: Verification and Compliance Testing. In: Huget, M.-P. (ed.) Communication in Multiagent Systems. LNCS (LNAI), vol. 2650, pp. 98–112. Springer, Heidelberg (2003)
19. Heljanko, K., Niemelä, I.: Bounded LTL model checking with stable models. Theory and Practice of Logic Programming 3(4-5), 519–550 (2003)
20. Henriksen, J.G., Thiagarajan, P.S.: Dynamic Linear Time Temporal Logic. Annals of Pure and Applied logic 96(1-3), 187–207 (1999)
21. Jennings, N.R.: Commitments and Conventions: the foundation of coordination in multi-agent systems. The knowledge engineering review 8(3), 233–250 (1993)
22. Kartha, G.N., Lifschitz, V.: Actions with Indirect Effects (Preliminary Report). In: KR 1994, pp. 341–350 (1994)
23. Lifschitz, V.: Frames in the Space of Situations. Artif. Intellig. 46, 365–376 (1990)
24. Singh, M.P.: A social semantics for Agent Communication Languages. In: Dignum, F.P.M., Greaves, M. (eds.) Issues in Agent Communication. LNCS (LNAI), vol. 1916, pp. 31–45. Springer, Heidelberg (2000)
25. van der Aalst, W., ter Hofstede, A.: YAWL: Yet Another Workflow Language. Information Systems 30(4), 245–275 (2005)
26. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases 14, 5–51 (2003)
27. van der Aalst, W.M.P., van Hee, K.M., ter Hofstede, A.H.M., Sidorova, N., Verbeek, H.M.W., Voorhoeve, M., Wynn, M.T.: Soundness of Workflow Nets: Classification, Decidability, and Analysiss. BPM Center Report BPM-08-02, BPMcenter.org (2008)
28. Weber, I., Hoffmann, J., Mendling, J.: Beyond soundness: On the verification of semantic business process models. Distributed and Parallel Databases 27(3), 271–343 (2010)
29. Yolum, P., Singh, M.P.: Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In: AAMAS 2002, pp. 527–534 (2002)

Appendix

A Dynamic Linear Time Temporal Logic

The appendix describes shortly the syntax and semantics of linear time logic DLTL [20].

Let Σ be a finite non-empty alphabet. The members of Σ are actions. Let Σ^* and Σ^ω be the set of finite and infinite words on Σ, where $\omega = \{0, 1, 2, \ldots\}$. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We denote by σ, σ' the words over Σ^ω and by τ, τ' the words over Σ^*. Moreover, we denote by $\leq$ the usual prefix ordering over Σ^* and, for $u \in \Sigma^\infty$, we denote by *prf(u)* the set of finite prefixes of u.

We define the set of programs (regular expressions) $Prg(\Sigma)$ generated by Σ as follows:

$$Prg(\Sigma) ::= a \mid \pi_1 + \pi_2 \mid \pi_1; \pi_2 \mid \pi^*$$

where $a \in \Sigma$ and π_1, π_2, π range over $Prg(\Sigma)$. A set of finite words is associated with each program by the mapping $[[]] : Prg(\Sigma) \rightarrow 2^{\Sigma^*}$, which is defined in the standard way.

Let $\mathcal{P} = \{p_1, p_2, \ldots\}$ be a countable set of atomic propositions containing $\top$ and $\bot$.

$$\mathrm{DLTL}(\Sigma) ::= p \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha\mathcal{U}^\pi\beta$$

where $p \in \mathcal{P}$ and α, β range over DLTL(Σ).

A model of DLTL(Σ) is a pair $M = (\sigma, V)$ where $\sigma \in \Sigma^\omega$ and $V : prf(\sigma) \rightarrow 2^{\mathcal{P}}$ is a valuation function. Given a model $M = (\sigma, V)$, a finite word $\tau \in prf(\sigma)$ and a formula α, the satisfiability of a formula α at τ in M, written $M, \tau \models \alpha$, is defined as follows:

- $M, \tau \models p$ iff $p \in V(\tau)$;
- $M, \tau \models \neg\alpha$ iff $M, \tau \not\models \alpha$;
- $M, \tau \models \alpha \vee \beta$ iff $M, \tau \models \alpha$ or $M, \tau \models \beta$;
- $M, \tau \models \alpha\mathcal{U}^\pi\beta$ iff there exists $\tau' \in [[\pi]]$ such that $\tau\tau' \in prf(\sigma)$ and $M, \tau\tau' \models \beta$. Moreover, for every τ'' such that $\varepsilon \leq \tau'' < \tau'$[1], $M, \tau\tau'' \models \alpha$.

A formula α is satisfiable iff there is a model $M = (\sigma, V)$ and a finite word $\tau \in prf(\sigma)$ such that $M, \tau \models \alpha$.

The formula $\alpha\mathcal{U}^\pi\beta$ is true at τ if "α until β" is true on a finite stretch of behavior which is in the linear time behavior of the program π.

The derived modalities $\langle\pi\rangle$ and $[\pi]$ can be defined as follows: $\langle\pi\rangle\alpha \equiv \top\mathcal{U}^\pi\alpha$ and $[\pi]\alpha \equiv \neg\langle\pi\rangle\neg\alpha$.

Furthermore, if we let $\Sigma = \{a_1, \ldots, a_n\}$, the $\mathcal{U}$, O (next), $\Diamond$ and $\Box$ operators of LTL can be defined as follows: $\bigcirc\alpha \equiv \bigvee_{a\in\Sigma}\langle a\rangle\alpha$, $\alpha\mathcal{U}\beta \equiv \alpha\mathcal{U}^{\Sigma^*}\beta$, $\Diamond\alpha \equiv \top\mathcal{U}\alpha$, $\Box\alpha \equiv \neg\Diamond\neg\alpha$, where, in $\mathcal{U}^{\Sigma^*}$, Σ is taken to be a shorthand for the program $a_1 + \ldots + a_n$. Hence both LTL(Σ) and PDL are fragments of DLTL(Σ). As shown in [20], DLTL(Σ) is strictly more expressive than LTL(Σ). In fact, DLTL has the full expressive power of the monadic second order theory of ω-sequences.

[1] We define $\tau \leq \tau'$ iff $\exists\tau''$ such that $\tau\tau'' = \tau'$. Moreover, $\tau < \tau'$ iff $\tau \leq \tau'$ and $\tau \neq \tau'$.

From Organisation Specification to Normative Programming in Multi-Agent Organisations

Jomi F. Hübner[1], Olivier Boissier[2], and Rafael H. Bordini[3]

[1] Dept Automation and Systems Engineering
Federal University of Santa Catarina
jomi@das.ufsc.br
[2] Ecole Nationale Supérieure des Mines
Saint Etienne, France
boissier@emse.fr
[3] Institute of Informatics
Federal University of Rio Grande do Sul
R.Bordini@inf.ufrgs.br

Abstract. In this paper, we show how we can automatically translate high-level organisation modelling languages into simpler languages based on the idea of normative programming. With this approach, while designers and agents still use a highly abstract organisational modelling language to specify and reason about the multi-agent organisation, the development of the organisation management infrastructure is facilitated in the following manner. The high-level organisation specification is automatically translated into a simple normative programming language that we have recently introduced and for which we have given formal semantics. The organisation management infrastructure can then be based on an interpreter for the simpler normative language. We illustrate the approach showing how $\mathcal{M}$OISE's organisation modelling language (with primitives such as roles, groups, and goals) can be translated into our normative programming language (with primitives such as norms and obligations). We briefly describe how this all has been implemented on top of ORA4MAS, the distributed artifact-based organisation management infrastructure for $\mathcal{M}$OISE.

1 Introduction

The use of organisational and normative concepts is widely accepted as an appropriate approach for the design and implementation of Multi-Agent Systems (MAS) [3]. They are thus present in several languages and frameworks for intelligent multi-agent systems. They are also used at runtime to make the agents aware of the organisations in which they take part and to support and monitor their activities. While the support aspect is important for any large-scale system, the monitoring one is particularly relevant for open MAS where the behaviour of the entering agents is unknown. A clear trend in the development of such systems is to provide organisation-oriented modelling languages that the MAS designer (human or agent, in the case of self-organisation) uses to write a program that prescribes the *organisational* functioning of the system [5,4,3,15,17,7], complementing agent programming languages that define the *individual* functioning within such system. These languages are interpreted by an Organisation Management Infrastructures (OMI) to realise the monitoring aspect of agent organisations.

In our work, we are particularly interested in flexible and adaptable development of OMIs. The exploratory stage of current OMIs often requires changes in their implementations so that one can experiment with new features. The refactoring of the OMI for such experiments, when the interpreter for the high-level modelling language has ad hoc implementations, is usually an expensive task that we wish to simplify. Our approach aims at expressing the various different constructs of the high-level modelling language into a unified framework by means of *norms*. The OMI is then realised by a mechanism for interpreting and managing the status of such norms instead of specific mechanisms for each of the constructs of the richer modelling language.

The solution proposed allows us to keep the language available to the designer and agents with high-level concepts such as groups, roles, and global plans. That language can be translated into (or compiled to) a simpler normative programming language that is then interpreted by the OMI. The problem of implementing the OMI is thereby reduced to: (1) the development of an interpreter for the normative language and (2) a translation problem (from the organisation modelling language to the normative programming language). More precisely, our starting language is the $\mathcal{M}$OISE Organisation Modelling Language (OML — see Sec. 3) and our target language is the Normative Organisation Programming Language (NOPL — see Sec. 4). NOPL is a particular class of a normative programming language that we introduced and formalised in [9], and we summarise it in Sec. 2. The translation process from OML into NOPL is fully automatic thanks to the contributions in this paper. All of this has been implemented on top of ORA4MAS, a distributed artifact-based approach for OMI (Sec. 5). This paper also gives an interesting contribution in elucidating the power of the *norm* abstraction in normative programming languages, which is enough to cover organisation specifications. The longer organisation specification/program translated into a normative programming language, although less readable for humans, is efficiently interpreted within OMI implementations.

The main components of our approach are, therefore: (i) a normative organisation programming language; (ii) the translation from an organisational modelling language into the normative organisation programming language; and (iii) an implemented artifact-based OMI that interprets the target normative language. The contributions of our approach are better discussed and placed in the context of the relevant literature in Sec. 6.

2 Normative Programming Language

A normative language is usually based on three primitives (obligation, permission, and prohibition) and two enforcement strategies (sanction and regimentation) [17,20,7]. While sanction is a reactive strategy applied after the event of a violation, regimentation is a preventive strategy whereby agents are not capable of violation [12]. Regimentation is important for an OMI since it allows the designer to define norms that must be followed because their violation present serious risks for the organisation.

The language we created is based on the following assumptions. (i) Permissions are defined by omission, as in [8]. (ii) Prohibitions are represented either by regimentation or as an obligation for someone else to decide how to handle the situation. For example, consider the norm “it is prohibited to submit a paper with more than 16 pages”. In case of regimentation of this norm, attempts to submit a paper with more than 16 pages will fail (i.e. they will be prevented from taking place). In case this norm is not to be regimented, the designer could define a norm such as “when a paper with more than 16 pages is submitted, the chair must decide whether to accept the submission or not”. (iii) Sanctions are represented as obligations (i.e. someone else is *obliged* to apply the sanction). (iv) Finally, norms are assumed to be consistent (either the programmer or program generator are supposed to handle this issue). Thus, the language can be relatively simple, reduced to two main constructs: *obligation* and *regimentation*.

Given the above requirements and simplifications, we can now introduce our Normative Programming Language (NPL). A normative program *np* is composed of: (i) a set of facts and inference rules (following the syntax used in *Jason* [1]); and (ii) a set of norms. A NPL norm has the general form

$$\texttt{norm}\ id\ \texttt{:}\ \varphi\ \texttt{->}\ \psi$$

where id is a unique *identifier* of the norm; φ is a formula that determines the *activation condition* for the norm; and ψ is the *consequence* of the activation of the norm. Two types of norm consequences ψ are available:

– *fail* – `fail`(r): represents the case where the norm is regimented; argument r represents the reason for the failure;
– *obl* – `obligation`(a, r, g, d): represents the case where an obligation for some agent a is created. Argument r is the reason for the obligation (which has to include the id of the norm that originated the obligation); g is the formula that represents the obligation itself (a state of the world that the agent must try to bring about, i.e. a goal it has to achieve); and d is the deadline to fulfil the obligation.

A simple example to illustrate the language is given below; we use source code comments to explain the program.

```
np example {
  a(1). a(2).                            // facts
  ok(X) :- a(A) & b(B) & A>B & X = A*B.  // rule
    // note that b/1 is not defined in the program;
    // it is a dynamic fact provided at run-time

  // alice has 4 hours to achieve a value of X < 5
  norm n1: ok(X) & X > 5
   -> obligation(alice,n1,ok(X) & X<5,`now`+`4 hours`).

  // bob is obliged to sanction alice in case X > 10
  norm n2: ok(X) & X > 10
   -> obligation(bob,n2,sanction(alice),`now`+`1 day`).

  // example of regimented norm; X cannot be > 15
  norm n3: ok(X) & X > 15 -> fail(n3(X)).
}
```

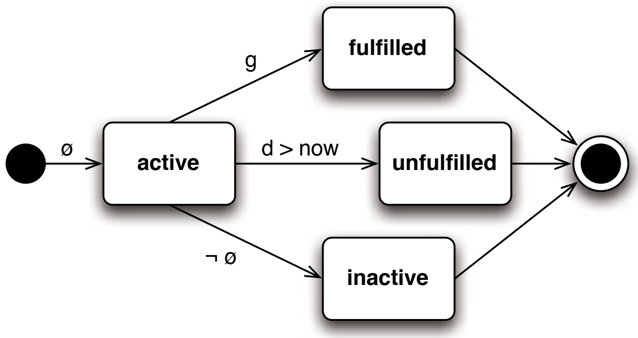


Fig. 1. State Transitions for Obligations

As in other approaches (e.g. [6,19]), a normative program expresses both static and declarative aspects of norms. The dynamic aspects result from the interpretation of such programs and the consequent creation of obligations for participating agents. An obligation has therefore a run-time life-cycle. It is created when the activation condition φ of some norm n holds. The activation condition formula is used to instantiate the values of variables a, r, g, and d of the obligation to be created. Once created, the initial state of an obligation is *active* (Fig. 1). The state changes to *fulfilled* when agent a fulfils the norm's obligation g before the deadline d. The obligation state changes to *unfulfilled* when agent a does not fulfil the norm's obligation g before the deadline d. As soon as the activation condition of the norm that created the obligation (φ) ceases to hold, the state changes to *inactive*. Note that a reference to the norm that led to the creation of the obligation is kept as part of the obligation itself (in the r argument), and the activation condition of this norm must remain true for the obligation to stay active; only an active obligation will become either fulfilled or unfulfilled, when the deadline is eventually reached. Fig. 1 shows the obligation life-cycle.

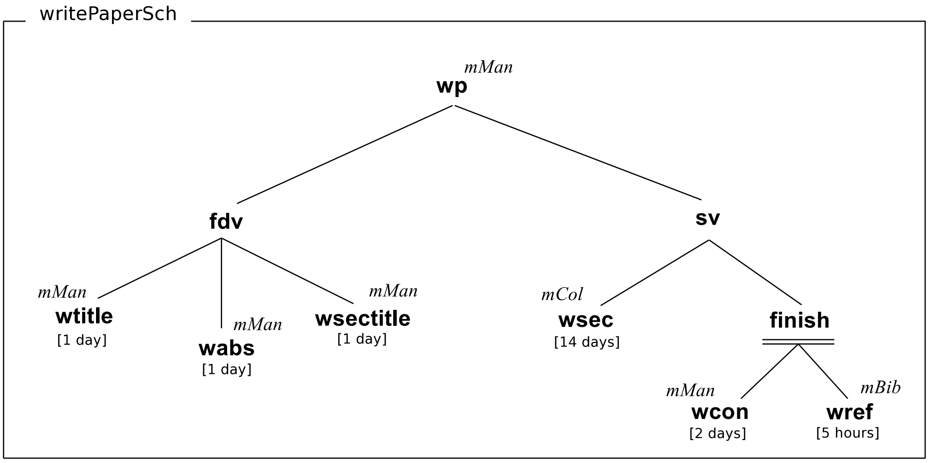
The syntax and semantics of NPL was introduced in [9]. the semantics was given using the well-known structural operational semantics approach.

3 $\mathcal{M}$OISE

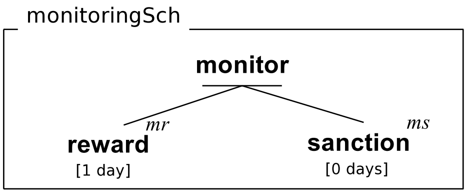
The $\mathcal{M}$OISE framework includes an organisational modelling language (OML) that explicitly decomposes the specification of organisations into structural, functional, and normative dimensions [11]. The structural dimension includes the *roles*, *groups*, and *links* (e.g. communication) within the organisation. The definition of roles is such that when an agent chooses to play some role in a group, it is accepting some behavioural constraints and rights related to this role. The functional dimension determines how the *global collective goals* should be achieved, i.e. how these goals are decomposed (through global *plans*) and grouped into coherent sets of subgoals (through *missions*) to be distributed among the agents. The decomposition of global goals results in a goal tree, called *scheme*, where the leaf-goals can be achieved individually by the agents. The normative dimension binds the structural dimension with the functional one by means of the specification of *permissions* and *obligations* towards missions given to particular roles. When an agent chooses to play some role in a group, it accepts these permissions and obligations.

As an illustrative and simple example of an organisation specified using $\mathcal{M}$OISE$^+$, we consider a scenario where agents aiming to write a paper together use an organisational specification to help them collaborate. We will focus on the functional and normative dimensions in the remainder of this paper. As for the structure of the organisation, it suffices to know that there is only one group (`wpgroup`) where two roles (*editor* and *writer*) can be played.

To coordinate the achievement of the goal of writing a paper, a scheme is defined in the functional specification of the organisation (Fig. 2(a)). In this scheme, a draft version of the paper has to be written first (identified by the goal *fdv* in Fig. 2(a)). This



(a) Paper Writing Scheme



(b) Monitoring Scheme

mission	cardinality
$mMan$	1..1
$mCol$	1..5
$mBib$	1..1
mr	1..1
ms	1..1

(c) Mission Cardinalities

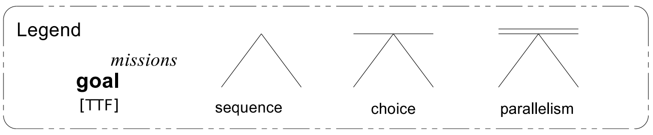


Fig. 2. Functional Specification for the Paper Writing Example

goal is decomposed into three subgoals: writing a title, an abstract, and the section titles; the subgoals have to be achieved in this very sequence. Other goals, such as *finish*, have subgoals that can be achieved in parallel. The specification also includes a "time-to-fulfil" (TTF) attribute for goals indicating a deadline for the agent to achieve the goal. The goals of this scheme are distributed into three missions which have specific cardinalities (see Fig. 2(c)): the mission $mMan$ is for the general management of the process (one and only one agent must commit to it), mission $mCol$ is for the collaboration in writing the paper content (from one up to five agents can commit to it), and mission $mBib$ is for gathering the references for the paper (one and only one agent must commit to it). A mission defines all the goals an agent commits to when participating in the execution of a scheme; for example, a commitment to mission $mMan$ is effectively a commitment to achieve four goals of the scheme. Goals without an assigned mission (e.g. *fdv*) are satisfied through the achievement of their subgoals.

Table 1. Normative Specification for the Paper Writing Example

id	condition	role	type	mission	TTF
n1		editor	per	$mMan$	–
n2		writer	obl	$mCol$	1 day
n3		writer	obl	$mBib$	1 day
n4	violation(n2)	editor	obl	ms	3 hours
n5	conformance(n3)	editor	obl	mr	3 hours
n6	#mc	editor	obl	ms	1 hour

#mc stands for the condition "more agents committed to a mission than permitted by the mission cardinality"

The normative specification relates roles and missions through norms (Table 1). For example, the oml-norm[1] `n2` states that any agent playing the role *writer* has one day to commit to mission $mCol$. Designers can also express application-dependent conditions (as in oml-norms `n4`–`n6`). Oml-norms `n4` and `n5` define sanction and reward strategies for violation and conformance of oml-norms `n2` and `n3` respectively. Oml-norm `n5` can be read as "the agent playing role 'editor' has 3 hours to commit to mission mr when norm `n3` is fulfilled". Once committed to mission mr, the editor has to achieve the goal *reward*. Note that an oml-norm in $\mathcal{M}$OISE is always an obligation or permission to commit to a mission. Goals are therefore indirectly linked to roles since a mission is a set of goals. Prohibitions are assumed 'by default' with respect to the specified missions: if the normative specification does not include a permission or obligation for a role-mission pair, it is assumed that the role does not grant the right to commit to the mission.

The OML is accompanied by a graphical language (see Fig. 2) and XML is used to store the organisational specifications (OS). In Sec. 4, instead of considering all the details of the graphical or the XML representation of an OS, we will consider the data

[1] To make clear the distinction between norms at the OML level with the ones at the NPL level, we will use the expression *oml-norm* when necessary.

structure produced by the OML parser. The data structure for an OS contains a set $\mathcal{FS}$ of scheme specifications and a set $\mathcal{NS}$ of oml-norms (again, only the functional and normative dimensions are being considered here).

When a scheme specification S is parsed, a tuple of the following type is produced:

$$\langle id, \mathcal{M}, maxmp, minmp, \mathcal{G}, gm, gpc, ttf, g_r \rangle$$

where

- id is a unique identification for S;
- $\mathcal{M}$ is a set of mission identifiers that agents can commit to within the scheme;
- $maxmp : \mathcal{M} \rightarrow \mathbb{Z}$: is a function that maps each mission to the maximum number of commitments of that mission in the scheme (upper bound of mission cardinality);
- $minmp : \mathcal{M} \rightarrow \mathbb{Z}$: maps each mission to the minimum number of commitments of that mission necessary for the scheme to be considered well-formed (lower bound of mission cardinality);
- $\mathcal{G}$ is the set of goals within the scheme;
- $gm : \mathcal{G} \rightarrow \mathcal{M}$ maps each goal to its mission;
- $gpc : \mathcal{G} \rightarrow 2^{\mathcal{G}}$ maps goals to their precondition goals;[2]
- $ttf : \mathcal{G} \rightarrow \mathbb{Z}$ maps goals to their TTF; and
- $g_r \in \mathcal{G}$ is the root goal of the scheme.

For each oml-norm in the normative specification, the parser produces a tuple

$$\langle id, c, \rho, t, m, ttf \rangle$$

where id is a unique identification for the oml-norm; c is the activation condition for the oml-norm; ρ is the role; t is the type (obliged or permitted); m is the mission; and ttf is the deadline. We can read that oml-norm as 'when c holds, agents playing ρ are t to commit to mission m by ttf'.

4 From OML to NOPL

After the presentation of NPL (the generic target language of the translation) and OML (the source for the translation), this section defines NOPL, a particular class of NPL programs applied to the $\mathcal{M}$OISE OML. The NOPL syntax and semantics are the same as presented in Sec. 2. However, the set of facts, rules, and norms are specific to the $\mathcal{M}$OISE model and to the $\mathcal{M}$OISE artifact-based OMI presented in Sec. 5. Benefiting from the distributed nature of this OMI, our proposal consists of translating the OS defined in $\mathcal{M}$OISE OML into *different* NOPL programs. For each group type defined in the OML, a separate NOPL program is produced by the translation. The same criteria is used to translate schemes. Since the OMI has one artifact to manage each instance of a group or scheme, the corresponding translated NOPL programs are used in the deployment of the artifacts.

[2] The precondition goals are deduced from the goal decomposition tree of the scheme (as presented in Fig. 2(a)). For example, the goal of "writing the paper conclusions" (*wcon*) can only be achieved after the goal of "writing sections" (*wsec*) has been achieved.

The path from NPL to NOPL consists firstly in the definition of facts and rules to express the different concepts and properties expressed in the OS. Dynamic facts are also introduced to represent the current state of the organisation. Below, we describe these rules and facts, step by step.

We use *translation rules* (briefly "t-rules") to formalise how the OS is translated into NOPL. Such rules have the following format:

$$\frac{\mathit{condition}}{\texttt{<code>}}\ (\mathit{ID})$$

where *ID* is the name of the t-rule, $\mathit{condition}$ is a Boolean expression, and `<code>` is an excerpt of code in NOPL that is produced in case the condition holds. Details of the application of these rules are provided in the examples given later.

In this paper, we consider only the translation rules for producing scheme normative programs, i.e. NOPL programs used to manage the corresponding scheme artifacts in the OMI. The t-rule that generates the NOPL code for a scheme specification is:

$$\frac{\langle id, \mathcal{M}, maxmp, minmp, \mathcal{G}, gm, gpc, ttf, g_r\rangle \in \mathcal{FS}}{\begin{array}{l}\texttt{np scheme(}id\texttt{) \{}\\ \quad \mathit{SM}(S) \quad \mathit{SMR}(S) \quad \mathit{SG}(\mathcal{G}) \quad \mathit{SR}(S) \quad \mathit{SSP} \quad \mathit{NS}\\ \texttt{\}}\end{array}}\ (S)$$

The condition for this t-rule is simply that the scheme specification belongs to the functional specification. The produced code (typeset in typewriter font) is a normative program with an identification id and facts, rules, and norms that are produced by specific t-rules (*SM*, *SMR*, *SG*, *SR*, *SSP*, and *NS*) defined below. Variables, typeset in italics (as in id), are replaced by their values obtained from the condition of the t-rule.

Facts. For scheme normative programs, the following facts are produced by the translation:

- `mission_cardinality(`m`,`min`,`max`)`: is a fact that defines the cardinality of a mission (e.g. `mission_cardinality(mCol,1,5)`).
- `mission_role(`m`,`ρ`)`: role ρ is permitted or obliged to commit to mission m (e.g. `mission_role(mMan,editor)`).
- `goal(`m`,`g`,`$pre\text{-}cond$`,`'ttf'`)`: is a fact that defines the arguments for a goal g: its mission, identification, preconditions, and TTF (e.g. `goal(mMan,wsec, [wcon], '2 days')`).

The t-rules *SM*, *SMR*, and *SG* generate these facts from the specification (all sets and functions, such as $\mathcal{M}_S$ and $minmp_S$, used in the t-rule refer to the scheme S being translated):

$$\frac{m \in \mathcal{M}_S \qquad maxmp_S(m) > 0}{\texttt{mission\_cardinality(}m\texttt{,}minmp_S(m)\texttt{,}maxmp_S(m)\texttt{)}\ .}\ (\mathit{SM(S)})$$

$$\frac{\langle id, c, \rho, t, m, ttf\rangle \in \mathcal{NS} \qquad maxmp_S(m) > 0}{\texttt{mission\_role(}m\texttt{,}\rho\texttt{)}\ .}\ (\mathit{SMR(S)})$$

$$\frac{g \in \mathcal{G}}{\texttt{goal(}gm(g)\texttt{,}g\texttt{,}gpc(g)\texttt{,}ttf(g)\texttt{).}}\quad (SG(\mathcal{G}))$$

The following dynamic facts will be provided at runtime by the artifact (cf. Sec. 5) that manages the scheme instance:

- `plays`(a,ρ,gr): agent a plays the role ρ in the group instance identified by gr.
- `responsible`(gr,s): the group instance gr is responsible for the missions of scheme instance s.
- `committed`(a,m,s): agent a is committed to mission m in scheme s.
- `achieved`(s,g,a): goal g in scheme s has been achieved by agent a.

Rules. Besides facts, we define some rules that are useful to infer the state of the scheme (e.g. whether it is well-formed) and goals (e.g. whether it is ready to be adopted or not). The rules produced by *SR* are general for all schemes and those produced by *SRW* are specific to the scheme being translated.

$$\frac{\qquad}{\qquad}\quad (SR(S))$$

```
is_finished(S) :- satisfied(S,g_r).

mission_accomplished(S,M) :-
   .findall(Goal, goal(M,Goal,_,_,), MissionGoals) &
   all_satisfied(S,MissionGoals).

all_satisfied(_,[]).
all_satisfied(S,[G|T]) :- satisfied(S,G) & all_satisfied(S,T).

// goal G of scheme S is ready to be adopted:
// all its preconditions have been achieved
ready(S,G) :-
   goal(_,G,PCG,_) & all_satisfied(S,PCG).

// number of players of a mission M in scheme S
mplayers(M,S,V) :- .count(committed(_,M,S),V).
   // .count(X) counts how many instances of X are known to the agent

well_formed(S) :- SRW(S).
```

$$\frac{m \in \mathcal{M} \qquad maxmps_S(m) > 0}{\begin{array}{l}\texttt{mission\_accomplished(S,}m\texttt{)}\\ \texttt{|}\\ \texttt{mplayers(}m\texttt{,S,V}m\texttt{) \& V}m\texttt{ >= }minmps_S(m)\texttt{ \& V}m\texttt{ <= }maxmps_S(m)\end{array}}\quad (SRW(S))$$

Note that these rules implement the semantics of *mission accomplishment*, *well-formed* and *ready goal* as intended in the $\mathcal{M}$OISE model.

As an example, the output of the translation produced by *SR* for the paper writing scheme is listed below.

```
is_finished(S) :- satisfied(S,wp). // wp is the root goal

mission_accomplished(S,M) :-
  .findall(Goal, goal(M,Goal,_,_), MissionGoals) &
  all_satisfied(S,MissionGoals).

all_satisfied(_,[]).
all_satisfied(S,[G|T]) :- satisfied(S,G) & all_satisfied(S,T).

ready(S,G) :- goal(_, G, PCG, _) & all_satisfied(S,PCG).

mplayers(M,S,V) :- .count(committed(_,M,S),V).
well_formed(S) :-
  (mission_accomplished(S,mMan) |
   mplayers(mMan,S,VmMan) & VmMan >= 1 & VmMan <= 1)
  &
  (mission_accomplished(S,mCol) |
   mplayers(mCol,S,VmCol) & VmCol >= 1 & VmCol <= 5)
  &
  (mission_accomplished(S,mBib) |
   mplayers(mBib,S,VmBib) & VmBib >= 1 & VmBib <= 1).
```

Norms. We have three classes of norms in NOPL for schemes: norms for goals, norms for properties, and domain norms (which are explicitly stated in the normative specification as oml-norms). For the former class, we define the following generic norm to express the $\mathcal{M}$OISE semantics for commitment:

```
norm ngoal:  committed(A,M,S) & goal(M,G,_,D) &
             well_formed(S) & ready(S,G)
-> obligation(A,ngoal,achieved(S,G,A),`now` + D).
```

This norm can be read as "when an agent A: (1) is committed to a mission M that (2) includes a goal G, and (3) the mission's scheme is well-formed, and (4) the goal is ready, then agent A is obliged to achieve the goal G before its deadline D". It also illustrates the advantage of using a translation to implement the OMI instead of an object-oriented programming language. For example, if some application or experiment requires a semantics of commitment where the agent is obliged to achieve the goal even if the scheme is not well-formed, it is simply a matter of changing the translation to a norm that does not include the `well_formed(S)` predicate in the activation condition of the norm. One could even conceive an application using schemes being managed by different NOPL programs (i.e. schemes translated differently).

For the second class of norms, only the mission cardinality property is introduced in this paper since other properties are handled in a similar way. In the case of mission cardinality, the norm has to define the consequences of situations where there are more agents committed to a mission than permitted in the scheme specification. As presented in Sec. 2, two kinds of consequences are possible, obligation and regimentation, and the designer chooses one or the other when writing the OS. Regimentation is the default consequence and it is used when there is no norm with condition #mc in the normative specification. Otherwise, if there is a norm such as n6 in Table 1, the consequence will be an obligation. The two t-rules below detail the produced norms for the regimentation and obligation cases of mission cardinality.

$$\neg\exists\,\langle id, c, \rho, t, m, ttf\rangle \in \mathcal{NS}\ \ .\ \ c = \#mc \qquad (SSP_1)$$

```
norm mc:
      mission_cardinality(M,_,MMax) &
      mplayers(M,S,MP) & MP > MMax
   -> fail(mission_cardinality).
```

$$\langle id, c, \rho, t, m, ttf\rangle \in \mathcal{NS} \qquad c = \#mc \qquad (SSP_2)$$

```
norm mc:
      mission_cardinality(M,_,MMax) &
      mplayers(M,S,MP) & MP > MMax &
      responsible(Gr,S) & plays(A,ρ,Gr)
-> obligation(A,mc,committed(A,m,_),`now`+`ttf`).
```

In our running example, the norm produced by SSP_1 to regiment mission cardinality is:

```
norm mc:
   mission_cardinality(M,_,MMax) &
   mplayers(M,S,MP) &  MP > MMax
-> fail(mission_cardinality).
```

and the norm produced if SSP_2 were used instead (for sanction rather than regimentation) would be:

```
norm mc:
   mission_cardinality(M,_,MMax) &
   mplayers(M,S,MP) &  MP > MMax &
   responsible(Gr,S) & plays(A,editor,Gr)
-> obligation(A,mc,committed(A,ms,_), `now`+`1 hour`).
```

where the agent playing editor is obliged to commit to the mission ms within one hour (corresponding to the oml-norm n6 in Table 1).

For the third class of norms, each oml-norm of type obligation in the normative specification of the OS has a corresponding norm in the NOPL program. Whereas an OML obligation refers to a role and a mission, NPL requires that obligations are for agents and towards a goal. The NOPL norm thus identifies each agent playing the role in groups responsible for the scheme and, if the number of current players still does not reach the maximum cardinality, and the mission was not accomplished yet, the agent is obliged to achieve a state where it is committed to the mission. The following t-rule expresses just that:

$$\frac{\langle id, c, \rho, t, m, ttf\rangle \in \mathcal{NS} \qquad m \in \mathcal{M} \qquad t = obl}{\texttt{norm } id\texttt{:}} \; (NS)$$

```
norm id:
      c &
      plays(A,ρ,Gr) & responsible(Gr,S) &
      mplayers(m,S,V) & V < maxmp(m) &
      not mission_accomplished(S,m)
-> obligation(A,id,committed(A,m,S),`now`+`ttf`).
```

For instance, the NOPL norm resulting from the translation of oml-norm `n2` in Table 1 with the t-rule above is:

```
norm n2:  plays(A,writer,Gr) & responsible(Gr,S) &
          mplayers(mCol,S,V) & V < 5 &
          not mission_accomplished(S,mCol)
-> obligation(A,n2,committed(A,mCol,S),`now`+`1 day`).
```

Note that if some mission is already accomplished (as defined by the t-rule *SR*), there is no obligation to commit to it (this interpretation of "obligation to commit" was originally proposed by [18]). An agent can thus commit to a mission, fulfil its goals, and leave the scheme before it is finished. Without this last condition, the agent has to participate in the scheme until it is removed from the multi-agent system. Note also that if the scheme already has 5 engaged agents, there is no obligation for other players of role *writer* to commit to $mCol$. In fact, if a sixth agent wanted to commit to $mCol$, norm `mc` would produce a failure.

Besides the obligations defined in the OML, we also have permissions and (by default) prohibitions. Since everything is permitted by default in NPL, OML permissions do not need to be translated. The OML prohibitions are handled in NOPL by a generic norm that fails when an agent is committed to a mission not permitted by its roles (according to the `mission_role` relation):

```
norm mission_permission:
   committed(Agt,M,S) &
   not (plays(Agt,R,_) & mission_role(M,R))
-> fail(mission_permission(Agt,M,S)).
```

The norm above uses regimentation to prohibit an agent to commit to a mission if it is not allowed to do so. Obligations could be used instead of regimentation, as illustrated by the following norm:

```
norm mission_permission:
   committed(Agt,M,S) &
   not (plays(Agt,R,_) & mission_role(M,R)) &
   plays(E,editor,_) // agent playing editor is obliged to
                     // to commit to a sanction mission
-> obligation(E,mp,committed(E,ms,_), `now`+`1 hour`).
```

The type of the consequence for the mission permission norm, whether a `fail` or an `obligation`, is defined by parameters passed on to the translator program.

Also regarding prohibitions, one could ask: Is an agent prohibited to leave its missions without fulfilling the mission's goals? There are two answers depending whether the following norm is included or not.

```
norm mission_leaved:
   leaved_mission(Agt,M,S) &
   not mission_accomplished(S,M)
-> fail(mission_leaved(Agt,M,S)).
```

If the above norm is included, the leave-mission action (which adds the fact `leaved_mission`) will fail. The action is regimented in this case. Otherwise, no error will be produced by the action. However, the norms generated from t-rule *NS* will be activated again and the agent becomes again obliged to commit to the mission.

5 Artifact-Based Architecture

The ideas presented in this paper have been implemented as part of an OMI that follows the Agent & Artifact model [13,10][3]. In this approach, a set of organisational artifacts is available in the MAS environment providing operations and observable properties for the agents so that they can interact with the OMI. For example, each scheme instance is managed by a "scheme artifact". A scheme artifact, shown in Fig. 3, provides operations such as "commit to mission" and "goal x has been achieved" (whereby agents can act upon the scheme) and observable properties (whereby agents can perceive the current state of the scheme). We can effortlessly distribute the OMI by deploying as many artifacts as necessary for the application.

Following the ideas introduced in this paper, each organisational artifact has within it an NPL interpreter that is given as input: (i) the NOPL program automatically generated from the OS for the type of the artifact (e.g. the artifact that will manage the writing paper scheme will receive as input the NOPL program translated from that scheme specification), and (ii) dynamic facts representing the current state of (part of) the organisation (e.g. the scheme artifact itself will produce dynamic facts related to the current state of the scheme instance). The interpreter is then used to compute: (i) whether some operation will bring the organisation into an inconsistent state (where inconsistency is defined by means of the specified regimentations), and (ii) the current state of the obligations.

Algorithm 1, implemented on top of CArtAgO [16], shows the general pattern we used to implement every operation (e.g. commitment to mission) in the organisational artifacts. Whenever an operation is triggered by an agent, the algorithm first stores a "backup" copy of the current state of the artifact (line 5). This backup is restored (line 10) if the operation leads to failure (e.g. committing to a mission not permitted). The overall functioning is that invalid operations do not change the artifact state.[4] A valid

[3] An implementation of the translator and the OMI is available at `http://moise.sourceforge.net`.

[4] This functioning requires that operations are not executed concurrently, which can be easily configured in CArtAgO.

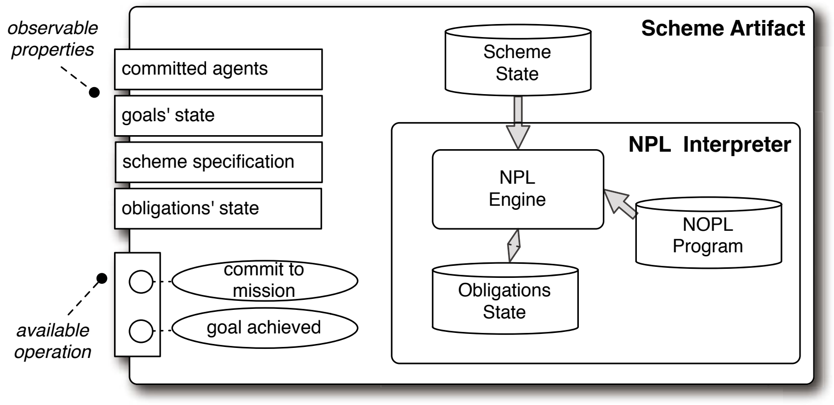


Fig. 3. General View of the Scheme Artifact

Algorithm 1. Artifact Integration with NOPL

```
1: $oe$ is the state of the organisation managed by the artifact
2: $p$ is the current NOPL program
3: $npi$ is the NPL interpreter
4: **when** an operation $o$ is triggered by agent $a$ **do**
5:     $oe' \leftarrow oe$ // creates a “backup” of the current $oe$
6:     executes operation $o$ to change $oe$
7:     $f \leftarrow$ a list of predicates representing $oe$
8:     $r \leftarrow npi(p, f)$ // runs the interpreter for the new state
9:     **if** $r$ = fail **then**
10:        $oe \leftarrow oe'$ // restore the backup state
11:        **return** fail operation $o$
12:    **else**
13:        update obligations in the observable properties
14:        **return** succeed operation $o$
```

operation is thus an operation that changes the state of the artifact to one where no `fail` (i.e. regimentation) is produced by the NPL interpreter. In case the operation is valid, the algorithm simply updates the current state of the obligations (line 13). Although the NPL handles *states* in the norm conditions, this pattern of integration has allowed us to use NPL to manage agent *actions*, i.e. the regimentation of operations on artifacts.

Notice that the NOPL program is not seen by the agents. They continue to perceive and reason on the scheme specification as written in the OML. The NOPL is used only within the artifact to simplify its development.

6 Related Work

This work is based on several approaches to organisation, institutions, and norms (cited throughout the paper). In this section, we briefly relate and compare our main contributions to such work.

The first contribution of the proposal, the NPL, should be considered specially for two properties of the language: its simplicity and its formal basis (that led to an available implementation). Similar work has been done by Tinnemeier et al. [17], where the operational semantics for a normative language was also proposed. Their approach and ours are similar on certain points. For instance, both consider norms as "declarative" norms (i.e. "ought-to-be" norms) in the sense that obligations and regimentation bear on goals. However, our work differs in several aspects. The NOPL class of NPL programs is for the OMI and not for programmers to use. The designers/programmers continue to use OML to define both an organisation and the norms that have to be managed within such a structure. Organisation primitives of the OML are higher-level and tailored for organisation modelling, therefore an OML specification is significantly more *concise* than its translation into a normative language.

Another clear distinction is that we rely on a dedicated programming model (the Agent & Artifact model) providing a clear connection of the organisation to the environment and allowing us to implement regimentation on physical actions [14]. The artifacts model also simplified the distribution of the management of the state of the organisation with several instances and kinds of artifacts, avoiding over-centralisation in the management of organisational and normative aspects of multi-agent systems.

Going back to the issue of conciseness and expressiveness, we do not claim that OML is more expressive than NPL. In fact, since the OML can be translated into NPL and some NPL programs cannot be translated into OML (for instance the program in the end of Sec. 2), NPL is strictly more expressive in theoretical terms. NOPL, on the other hand, has the same expressiveness of OML, since we can translate NOPL back into OML (this translation is not the focus of this paper but is feasible). However, we are not looking for general purpose or more expressive programming languages, but languages that help automating part of the OMI development. In this regard, the OML was designed to be more concise and more abstract than NOPL. The OML allows the designer to specify complex properties of the organisation with natural abstractions and fewer lines of code, which are translated into several lines of NOPL code that is interpreted by the NPL engine.

Regarding the second contribution, namely the automatic translation, we were inspired by work on ISLANDER [2,7]. The main difference here is the initial and target languages. While they translate a normative specification into a rule-based language, we start from a high-level organisation modelling language and the target is a simple normative programming language. NOPL is more specific than rule-based languages, being specifically tailored from our NPL for the $\mathcal{M}$OISE OML.

Regarding the third contribution, the OMI, we started from ORA4MAS [10]. The advantages of the approach presented here are twofold: (i) it is easier to change the translation than the Java implementation of the OMI; and (ii) from the operational semantics of NPL and the formal translation we are taking significant steps towards a formal semantics for $\mathcal{M}$OISE, which is a well-known organisational model that has not yet been fully formalised.

$\mathcal{M}$OISE shares some concepts with a variety of organisational models available in the multi-agent systems literature, so we expect to be able to use our approach to give concrete semantics and efficient implementations for a variety of other modelling languages too.

7 Conclusions

In this paper, we introduced a translation from an organisation specification written in $\mathcal{M}$OISE OML into a normative program that can be interpreted by an artifact-based OMI. Focusing on the translation, we can bring flexibility to the development of OMIs. Our work also emphasises the point that a normative programming language can be based on only two basic concepts: regimentation and obligation. Prohibitions are considered either as regimentation or as an obligation for someone else to sanction in case of violation of the prohibitions. The resulting NPL is thus simpler to formalise and implement.

Another result of this work is to show that an organisational language (OML) can be translated (and reduced to) a normative programming language. Roughly, all management within an OMI can be based on the management of norms and obligations. This result emphasises the importance of norms as a fundamental concept for the development of OMIs. Future work will explore that capacity of NPL for other organisational and institutional languages.

We also plan to investigate further possible relationships among norms, for instance when the activation of a norm triggers another. This form of chain triggering of norms is already possible in the language. The current state of an obligation is one of the dynamic facts updated by the artifact and accessible to the NPL interpreter, and it can be used in the condition of norms. For example, we can write: `norm x: active(obligation(....)) -> fail(....).`. However, this feature of the language requires further experimentation.

It also remains future work to evaluate how the reorganisation process available in $\mathcal{M}$OISE will impact on the normative-based artifacts. Changes in the organisation specification imply changes in the corresponding NOPL programs. We can simply change these programs in the artifacts, but there are problems that require investigation. For example, the problem of what to do with the active obligations, created from an organisation that changed, and which might need to be dropped in some cases and prevented from being dropped just because of a change in the NOPL program in other cases. The revision of such obligations is one of the main issue we will consider in this area of future work.

Acknowledgements

The authors are grateful for the supported given by CNPq, grants 307924/2009-2, 307350/2009-6, and 478780/2009-5.

References

1. Bordini, R.H., Hübner, J.F., Wooldrige, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley Series in Agent Technology. John Wiley & Sons, Chichester (2007)

2. da Silva, V.T.: From the specification to the implementation of norms: an automatic approach to generate rules from norm to govern the behaviour of agents. Journal of Autonomous Agents and Multi-Agent Systems 17(1), 113–155 (2008)
3. Dignum, V. (ed.): Handbook of Research on Multi-agent Systems: Semantics and Dynamics of Organizational Models. Information Science Reference (2009)
4. Esteva, M., de la Cruz, D., Sierra, C.: ISLANDER: an electronic institutions editor. In: Castelfranchi, C., Lewis Johnson, W. (eds.) Proceedings of the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002). LNCS (LNAI), vol. 1191, pp. 1045–1052. Springer, Heidelberg (2002)
5. Ferber, J., Gutknecht, O.: A meta-model for the analysis and design of organizations in multi-agents systems. In: Demazeau, Y. (ed.) Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS 1998), pp. 128–135. IEEE Press, Los Alamitos (1998)
6. Fornara, N., Colombetti, M.: Specifying and enforcing norms in artificial institutions. In: Omicini, A., Dunin-Keplicz, B., Padget, J. (eds.) Proceedings of the 4th European Workshop on Multi-Agent Systems (EUMAS 2006) (2006)
7. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: Constraining rule-based programming norms for electronic institutions. Journal of Autonomous Agents and Multi-Agent Systems 18(1), 186–217 (2009)
8. Grossi, D., Aldewered, 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)
9. Hübner, J.F., Boissier, O., Bordini, R.H.: A normative organisation programming language for organisation management infrastructures. In: Padget, J., et al. (eds.) COIN 2009. LNCS (LNAI), vol. 6069, pp. 114–129. Springer, Heidelberg (2010)
10. Hübner, J.F., Boissier, O., Kitio, R., Ricci, A.: Instrumenting multi-agent organisations with organisational artifacts and agents: "giving the organisational power back to the agents". Journal of Autonomous Agents and Multi-Agent Systems 20(3), 369–400 (2010)
11. Hübner, J.F., Sichman, J.S., Boissier, O.: Developing organised multi-agent systems using the MOISE+ model: Programming issues at the system and agent levels. International Journal of Agent-Oriented Software Engineering 1(3/4), 370–395 (2007)
12. Jones, A.J.I., Sergot, M.: On the characterization of law and computer systems: the normative systems perspective. In: Deontic logic in computer science: normative system specification, pp. 275–307. John Wiley and Sons Ltd., Chichester (1993)
13. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the A&A meta-model for multi-agent systems. Journal of Autonomous Agents and Multi-Agent Systems 17(3), 432–456 (2008)
14. Piunti, M., Ricci, A., Boissier, O., Hübner, J.F.: Embodying organisations in multi-agent work environments. In: Proceedings of International Joint Conferences on Web Intelligence and Intelligent Agent Technologies (WI-IAT 2009), pp. 511–518. IEEE/WIC/ACM (2009)
15. Pynadath, D.V., Tambe, M.: An automated teamwork infrastructure for heterogeneous software agents and humans. Autonomous Agents and Multi-Agent Systems 7(1-2), 71–100 (2003)
16. Ricci, A., Piunti, M., Viroli, M., Omicini, A.: Environment programming in CArtAgO. In: Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.) Multi-Agent Programming: Languages, Tools and Applications, ch. 8, pp. 259–288. Springer, Heidelberg (2009)
17. Tinnemeier, N., Dastani, M., Meyer, J.-J.: Roles and norms for programming agent organizations. In: Sichman, J., Decker, K., Sierra, C., Castelfranchi, C. (eds.) Proc. of AAMAS 2009, pp. 121–128 (2009)

18. van Riemsdijk, B., Hindriks, K., Jonker, C.M., Sierhuis, M.: Formal organizational constraints: A semantic approach. In: Hoek, Kaminka, Lesperance, Luck, Sen (eds.) Proc. of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), pp. 823–830 (2010)
19. Vázquez-Salceda, J., Aldewereld, H., Dignum, F.: Norms in multiagent systems: some implementation guidelines. In: Proceedings of the Second European Workshop on Multi-Agent Systems, EUMAS 2004 (2004), http://people.cs.uu.nl/dignum/papers/eumas04.PDF
20. López y López, F., Luck, M.: Constraining autonomy through norms. In: Luck, M., d'Inverno, M. (eds.) Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 674–681. ACM Press, New York (2002)

Finding Uniform Strategies for Multi-agent Systems

Jan Calta[1], Dmitry Shkatov[2], and Holger Schlingloff[3]

[1] Humboldt University, Berlin, Germany
calta@informatik.hu-berlin.de
[2] University of the Witwatersrand, Johannesburg, South Africa
dmitry@cs.wits.ac.za
[3] Humboldt University, Berlin, Germany
hs@informatik.hu-berlin.de

Abstract. We present an algorithm for finding uniform strategies in multi-agent systems with incomplete information. The algorithm finds all maximal uniform strategies for agents with incomplete information for enforcing a property expressible in the language of Alternating-time Temporal Logic ATL. The main application of the algorithm is automated program synthesis for systems that can be modeled as multi-agent systems with incomplete information (e.g., decentralized distributed systems).

1 Introduction

Over the last few years, the multi-agent systems paradigm has been deployed in safety-critical applications, such as sensor networks for detection of earthquakes (e.g., SOSEWIN network developed in the scope of the SAFER project [1]). For such applications, it is crucially important to verify or design software controlling the system using a formal procedure, which guarantees with certainty that the desired goal has been achieved. In this paper, we present an algorithm for designing software for such systems conforming to required specifications.

As a formal model for multi-agent systems, we use Concurrent Epistemic Game Structures (CEGS) introduced in [12]; as a modeling language, we use the language of Alternating-time Temporal Logic ATL [2]. One of the most important aspects of CEGS's are *strategies* agents use to enforce properties expressible in a given modeling language. CEGS's allow us to model systems in which agents have incomplete information. As noted in [8], in such systems, not all strategies are of interest, but only those in which every agent performs the same action in the states indistinguishable to them; only such strategies, usually referred to as *uniform*, can be viewed as formal models of algorithms. Another distinction is usually made between strategies in CEGS's: memory-based (agents remember some, probably the whole, history of the computation) vs. memoryless (the agents base their decisions on the current state of the computation); in the present paper we only consider memoryless strategies. Given the choice of CEGS's as models for multi-agent systems, the problem of designing software conforming to a given specification turns into the problem of finding uniform strategies enforcing a property. This property has to be expressible in the chosen modeling language. For reasons that will become clearer later on, even though we use the syntax of ATL, we define the semantics in terms of uniform strategies; moreover, we evaluate formulas at sets of states rather than states. The result is the logic which we call ATL_u, which is thus our formalism of choice for verification and program synthesis in the context of multi-agent systems with incomplete information.

The main goal of the paper is to present an algorithm for finding uniform strategies in CEGS's with incomplete information. The paper is structured as follows. In section 2, we define the logic ATL_u as well as the concepts used in the rest of the paper. In section 3, we present the algorithm for finding uniform strategies and prove its correctness. In section 4, we present an example of running the algorithm on a simple multi-agent system. In section 5, we estimate the complexity of the algorithm. Finally, in conclusion, we summarize our results and point to directions for further research.

2 Preliminaries

We use concurrent epistemic game structures (CEGS), as defined in [5], as models for reasoning about agents with incomplete information.[1] A CEGS is a tuple

$$\mathfrak{M} = \langle Agt, St, \Pi, \pi, Act, d, o, \sim_1, ..., \sim_k \rangle,$$

where:

- $Agt = \{1, \ldots, k\}$ is a finite set of agents; a (possibly, empty) subset of Agt is called a *coalition*; arbitrary agents will be denoted with $a, b, \ldots$; arbitrary coalitions with $A, B, \ldots$;
- St is a nonempty, finite set of states;
- Π is a set of atomic propositions;
- $\pi : \Pi \to \mathcal{P}(St)$ is a valuation function;
- Act is a nonempty, finite set of actions;
- $d : Agt \times St \to \mathcal{P}(Act)$ assigns to an agent and a state a nonempty subset of Act, which we think of as actions available to the agent at that state. For every $q \in St$, an *action vector* at q is a k-tuple $\langle \alpha_1, \ldots, \alpha_k \rangle$ such that $\alpha_a \in d(a, q)$, for every $1 \leq a \leq k$. The set of all action vectors at q is denoted by $D(q)$;
- o assigns to every $q \in St$ and every $v \in D(q)$ the outcome $o(q, v) \in St$;
- $\sim_1, \ldots, \sim_k \subseteq St \times St$ are indistinguishability relations for agents $1, \ldots, k$. We assume that $\sim_a$, for each $a \in Agt$, is an equivalence relation and, moreover, that $q \sim_a q'$ implies $d(a, q) = d(a, q')$ (i.e., an agent has the same choice of actions at indistinguishable states).

[1] The notion of agent, as used in the literature, is quite an abstract one. For the purposes of this paper, however, the reader can think of agents as components of a distributed system.

One can use CEGS's to synthesize software for distributed systems, or to verify that such systems satisfy certain properties. To do this in a formal way, we introduce logic ATL$_u$, whose syntax is defined as follows[2]:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\!\langle A \rangle\!\rangle \mathsf{X}\, \varphi \mid \langle\!\langle A \rangle\!\rangle \mathsf{G}\, \varphi \mid \langle\!\langle A \rangle\!\rangle \varphi\, \mathsf{U}\, \varphi,$$

where $\mathsf{p} \in \Pi$ and $A \subseteq Agt$. The operator $\langle\!\langle\ \rangle\!\rangle$ is called *coalitional operator*, while the operators X, G and U are temporal operators *next, always* and *until*, respectively. We now introduce some notions necessary to define the semantics of ATL$_u$.

In what follows, given a tuple t of length at least i, we denote by $t[i]$ the ith element of t. The symbol $\sharp$ is a placeholder for a particular action of the respective agent. Thus, if we have three agents, then the tuple $< a_1, a_2, \sharp >$ represents the situation where agents 1 and 2 have chosen actions a_1 and a_2, respectively, but the action of agent 3 is unspecified. We will use this "action underspecification" in a number of ways further on in the paper. Note that the meaning of $\sharp$ is the same as in [4].

Definition 1. *Let $q \in St$ and let $A \subseteq \{1, .., k\}$ be a coalition of agents. An* A-move *at state q is a pair $\langle q, m_A \rangle$ where m_A is a k-tuple such that $m_A[a] \in d(a, q)$, for every $a \in A$, and $m_A[a] = \sharp$ otherwise. For the reasons that will become clear later on, we also count as an A-move for an arbitrary coalition A at q the pair $\langle q, \sharp^k \rangle$. The set of all A-moves at state q is denoted by $D_A(q)$.*

Definition 2. *Let $q \in St$ and let $m_A \in D_A(q)$ such that $m_A \neq \sharp^k$. The* outcome *of $\langle q, m_A \rangle$, denoted by $out(q, m_A)$, is the set of all states q' such that there is an action vector $v \in D(q)$ with $v[a] = m_A[a]$, for all $a \in A$, such that $o(q, v) = q'$. The outcome of $\langle q, \sharp^k \rangle$ is the set of states q' such that $o(q, v) = q'$ for some $v \in D(q)$.*

Definition 3. *A (memoryless)* strategy *S of a coalition A, denoted by S_A, is a partial function assigning $|A|$-tuples of actions to states, such that if S_A is defined on $q \in St$ then $\langle q, S_A(q) \rangle$ is an A-move from $D_A(q)$. The domain of S_A is denoted by $dom(S_A)$. A strategy S_A is* uniform *if $q \sim_a q'$ implies $S_A(q)[a] = S_A(q')[a]$ for every $q, q' \in dom(S_A)$ and every $a \in A$.*

We define strategies as a partial functions for technical reasons that will be explained later on. Note that a strategy S_A can be seen as a set of A-moves $\langle q, m_A \rangle$ such that $S_A(q) = m_A$.

Given a sequence of states Λ, we denote by $|\Lambda|$ the number of states in Λ; if Λ is infinite, $|\Lambda| = \omega$. The ith state of Λ is denoted by $\Lambda[i]$.

Definition 4. *A* path *Λ is a (possibly, infinite) sequence of states $q_1, q_2, q_3 \ldots$ that can be effected by subsequent transitions; that is, for every $1 \leq i < |\Lambda|$, if $q_i \in \Lambda$, then there exists an action vector $v \in D(q_i)$ such that $q_{i+1} = o(q_i, v)$.*

[2] The syntax of ATL$_u$ is identical to Alternating-Time Temporal Logic (ATL) [2].

We now define outcomes of strategies. We use the notation S_A to refer to a strategy of coalition A.

Definition 5. *Let $q \in St$ and let S_A be a strategy such that $\langle q, m_A \rangle \in S_A$. The* outcome *of S_A at q, denoted by $out(q, S_A)$, is the set of paths $\{\Lambda \mid \Lambda[1] = q$ and for each $1 \leq i < |\Lambda|$ there is an A-move $\langle \Lambda[i], m_A \rangle \in S_A$ such that $\Lambda[i+1] \in out(\Lambda[i], m_A)\}$. If $\Lambda \in out(q, S_A)$ is finite, then we require that either $\langle \Lambda[|\Lambda|], \sharp^k \rangle \in S_A$ or $\Lambda[|\Lambda|] \notin dom(S_A)$.*

Intuitively, $\langle \Lambda[|\Lambda|], \sharp^k \rangle \in S_A$ means that it does not matter what the agents in A do at the last state of a path that is an outcome of S_A. This possibility of giving the agents in A a "free rein" at the end of carrying out a strategy is what motivated us to count $\sharp^k$ as an A-move, for every $A \subseteq Agt$.

Outcome $out(q, S_A)$ contains every path starting at q that may result from coalition A performing A-moves assigned by S_A to the states on the path. We use notation $out(Q, S_A)$ as a shorthand for $\bigcup_{q \in Q} out(q, S_A)$.

Example 1. Consider the CEGS, depicted in Fig. 1, with $Agt = \{1, 2\}$ and $Act = \{\alpha_1, \beta_1, \alpha_2, \beta_2\}$. Coalition A consists of agent 1 ($A = \{1\}$) and thus every A-move is a tuple with an action for agent 1 and the placeholder $\sharp$ for agent 2. Only one A-move is possible at state q, namely $\langle \alpha_1, \sharp \rangle$, so that $D_A(q) = \{\langle \alpha_1, \sharp \rangle\}$; analogously, $D_A(q') = \{\langle \alpha_1, \sharp \rangle, \langle \beta_1, \sharp \rangle\}$ and $D_A(q'') = \{\langle \alpha_1, \sharp \}$. The outcome of $\langle \alpha_1, \sharp \rangle$ at q is $\{q', q''\}$. $S_A = \{\langle q, \langle \alpha_1, \sharp \rangle \rangle, \langle q', \langle \alpha_1, \sharp \rangle \rangle\}$ is a uniform strategy of A. The outcome of the S_A at q is $\{q, q''; q, q', q''\}$.

We now turn to defining the meaning of ATL$_u$-formulas over CEGS. Taking our cue from [5], we evaluate formulas at sets of states rather than states. Intuitively, given a CEGS $\mathfrak{M}$ and a set $Q \subseteq St$, we have $\mathfrak{M}, Q \models \langle\langle A \rangle\rangle \psi$ if there is a uniform strategy S_A such that ψ is satisfied by all paths in $out(Q, S_A)$. Notice that we evaluate formulas at *arbitrary states* rather than at those states that happen to be equivalence classes, as is done in [10] and [5]. Our reason for evaluating at sets of states is pragmatic: given a formula $\langle\langle A \rangle\rangle \psi$ our algorithm return the set of states at which the coalition A can enforce ψ. This, together with our treatment of strategies as partial functions, immediately gives us all the states from with A can enforce ψ, alongside with the actual strategies for doing so. We now formally define the semantics of ATL$_u$.

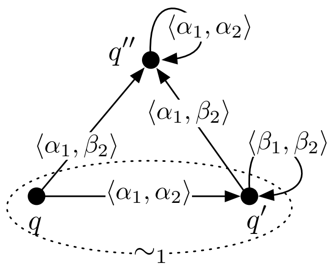

q''
$\langle \alpha_1, \alpha_2 \rangle$
$\langle \alpha_1, \beta_2 \rangle$
$\langle \alpha_1, \beta_2 \rangle$
$\langle \beta_1, \beta_2 \rangle$
$\langle \alpha_1, \alpha_2 \rangle$
q
q'
$\sim_1$

Fig. 1. An example of a CEGS

$\mathfrak{M}, Q \models \mathsf{p}$ iff $Q \subseteq \pi(\mathsf{p})$;
$\mathfrak{M}, Q \models \neg\varphi$ iff $\mathfrak{M}, Q \not\models \varphi$;
$\mathfrak{M}, Q \models \varphi_1 \wedge \varphi_2$ iff $\mathfrak{M}, Q \models \varphi_1$ and $\mathfrak{M}, Q \models \varphi_2$;
$\mathfrak{M}, Q \models \langle\langle A\rangle\rangle\psi$ iff there exists a uniform S_A such that $\mathfrak{M}, \Lambda \Vdash \psi$, for every $\Lambda \in \mathit{out}(Q, S_A)$;
$\mathfrak{M}, \Lambda \Vdash \mathsf{X}\ \varphi$ iff $\mathfrak{M}, \{\Lambda[2]\} \models \varphi$;
$\mathfrak{M}, \Lambda \Vdash \mathsf{G}\ \varphi$ iff Λ is infinite, and $\mathfrak{M}, \{\Lambda[i]\} \models \varphi$, for every $i \geq 1$;
$\mathfrak{M}, \Lambda \Vdash \varphi_1 \mathsf{U} \varphi_2$ iff $\mathfrak{M}, \{\Lambda[j]\} \models \varphi_2$, for some $j \geq 1$, and $\mathfrak{M}, \{\Lambda[i]\} \models \varphi_1$, for every $1 \leq i < j$.

Notice that the expressions $\mathsf{X}\,\varphi, \mathsf{G}\,\varphi$, and $\varphi_1 \,\mathsf{U}\, \varphi_2$ referred to in the last three clauses of the above definition are not ATL$_u$-formulas; they hold at paths rather that being satisfied by sets of states, as in the case of ATL$_u$-formulas.

For technical reasons, which will become clear later on, we want the meaning of every ATL$_u$-formula—rather than only formulas beginning with a coalitional operator, also referred to as "strategic formulas"—to be defined in terms of uniform strategies. To that end, we give an alternative semantics of ATL$_u$-formulas, where the meaning of "non-strategic" formulas is defined in terms of the empty coalition $\emptyset$ (which is equivalent to the universal quantifier of CTL). As the only $\emptyset$-move is $m_\emptyset = \sharp^k$, every strategy of the empty coalition is uniform. Intuitively, a strategy $S_\emptyset$ can be thought of as the domain of $S_\emptyset$, because it is not important for us what particular actions the agents perform at the states in the domain of $S_\emptyset$. We now redefine the meaning of non-strategic ATL$_u$-formulas in terms of uniform strategies. Thus,

$\mathfrak{M}, Q \models \mathsf{p}$ iff there exists $S_\emptyset$ such that $Q = \mathit{dom}(S_\emptyset) \subseteq \pi(\mathsf{p})$;
$\mathfrak{M}, Q \models \neg\varphi$ iff there exists $S_\emptyset$ such that $Q = \mathit{dom}(S_\emptyset)$ and $\mathfrak{M}, \mathit{dom}(S_\emptyset) \not\models \varphi$;
$\mathfrak{M}, Q \models \varphi \wedge \psi$ iff there exists $S_\emptyset$ such that $Q = \mathit{dom}(S_\emptyset)$ and $\mathfrak{M}, \mathit{dom}(S_\emptyset) \models \varphi$ and $\mathfrak{M}, \mathit{dom}(S_\emptyset) \models \psi$;

In what follows, if CEGS $\mathfrak{M}$ is clear from the context, we write $Q \models \varphi$ instead of $\mathfrak{M}, Q \models \varphi$.

We now turn to the problem of finding uniform strategies enforcing a given ATL$_u$-formula in a given CEGS.

3 Finding Uniform Strategies in CEGSs

The main purpose of the present paper is to describe an algorithm for finding uniform strategies in CEGSs. The problem of finding such strategies can be viewed as a "constructive" model-checking problem for ATL$_u$; that is, given an ATL$_u$ formula $\langle\langle A\rangle\rangle\psi$ and a CEGS $\mathfrak{M}$, we want to find all uniform strategies of coalition A in $\mathfrak{M}$ that enforce ψ. For each such strategy S_A, we also get the set of states from which S_A can be effected and which, thus, satisfies the formula $\langle\langle A\rangle\rangle\psi$. Therefore, the problem that is solved by our algorithm is an extension of a model-checking problem for ATL$_u$.

Since uniform strategies can be viewed as programs, the extended algorithm presented in this section allows us to synthesize distributed programs achieving the outcomes that can be expressed using ATL_u formulas.

Since the formula φ given as an input to our algorithm may contain a number of strategic subformulas, each requiring for its satisfaction the existence of (possibly, more than one) uniform strategy, we have to find all uniform strategies associated with each such subformula of φ, including φ itself. As with each strategic subformula of φ there might be associated several uniform strategies, some of which may contain others, we are only interested in finding maximal uniform strategies for each such subformula, i.e., the ones that can not be extended to a bigger set of states.

Definition 6. *Let S be a (not necessarily uniform) strategy, and let φ be an ATL_u-formula.*

- *If $\varphi = \langle\!\langle A \rangle\!\rangle \psi$ (i.e., φ is a strategic formula) then S is a* strategy for φ *if S is a strategy of coalition A, and $\Lambda \Vdash \psi$ for every $\Lambda \in out(dom(S), S)$;*
- *otherwise, S is a* strategy for φ *if S is a strategy of the coalition $\emptyset$, and $dom(S) \models \varphi$.*

Definition 7

- *A uniform strategy S_A for an ATL_u-formula $\varphi = \langle\!\langle A \rangle\!\rangle \psi$ is* maximal *if there is no uniform strategy S'_A for φ such that $S_A \subset S'_A$.*
- *Let M be a set of A-moves and let $S_A \subseteq M$ be a uniform strategy. S_A is* maximal in M *if there is no uniform strategy S'_A such that $S_A \subset S'_A \subseteq M$.*

We can now more precisely restate the problem our algorithm solves as follows: given a CEGS $\mathfrak{M}$ and an ATL_u-formula φ, find *all maximal uniform strategies S_A for every subformula of φ, including φ itself.*

The control structure of our algorithm (Alg. 1) is based on the control structure of the model-checking algorithm for ATL from [2]. The major difference between the two algorithms is that our algorithm returns, for each subformula ψ of the input formula φ, the set of all maximal uniform strategies for φ, rather then just a set of states at which φ holds.

The input of the algorithm is a CEGS $\mathfrak{M}$ and an ATL_u-formula φ. The given CEGS $\mathfrak{M}$ with $|Agt| = k$ is used as a global variable accessible from any point of the algorithm. We denote by $[\psi]$ the set of all maximal uniform strategies for an ATL_u formula ψ. For every subformula ψ of φ, including φ itself, $[\psi]$ is available as a global variable from any point of the algorithm and is returned by the algorithm.

Note that, if ψ is a subformula of φ and there exist more than one uniform strategy for ψ, then the union of the domains of all those strategies is used to compute the maximal uniform strategies for φ. We usually enumerate maximal uniform strategies for a formula using upper indices, as in S^1. The algorithm uses the following functions:

- *Subformulas*(φ) returns the list of subformulas of φ in the following order: if ψ is a subformula of τ then ψ precedes τ.

- $Dom(M)$ returns, for a given set M of A-moves, the set $\{q \in St \mid \langle q, m_A\rangle \in M\}$. (If M is a strategy then $Dom(M)$ returns its domain.)
- $Pre(A, Q)$ returns, for a coalition A and a set of states Q, the pre-image of Q with respect to A, defined as the set $\{\langle p, m_A\rangle \mid m_A \in D_A(p)$ and $out(p, m_A) \subseteq Q\}$.
- $Uniform(\varphi, S)$, where φ is a strategic ATL$_u$ formula and S is the union of all (not necessarily uniform) strategies for φ, returns all maximal uniform strategies for φ[3].

We now explain the issues involved in computing the set of all maximal uniform strategies for a formula $\varphi = \langle\langle A\rangle\rangle\psi$.

Algorithm 1. Constructive model checking for ATL$_u$

```
strategies := ∅;
foreach $\varphi'$ in $Subformulas(\varphi)$ do
    $[\varphi'] := \emptyset$;
    case $\varphi' = \mathsf{p}$ :
        $S^1 := \{\langle q, \sharp^k\rangle \mid q \in \pi(\mathsf{p})\}$; $[\varphi'] := \{S^1\}$;
    case $\varphi' = \neg\psi$ :
        $S^1 := \{\langle q, \sharp^k\rangle \mid \nexists S \in [\psi] : q \in Dom(S)\}$; $[\varphi'] := \{S^1\}$;
    case $\varphi' = \psi_1 \wedge \psi_2$ :
        $T^1 := \bigcup_{S^i \in [\psi_1]} S^i$; $T^2 := \bigcup_{S^i \in [\psi_2]} S^i$;
        $S^1 := \{\langle q, \sharp^k\rangle \mid q \in Dom(T^1) \cap Dom(T^2)\}$; $[\varphi'] := \{S^1\}$;
    case $\varphi' = \langle\langle A\rangle\rangle \mathsf{X}\, \psi$ :
        $S := \bigcup_{S^i \in [\psi]} S^i$;
        $P := Pre(A, Dom(S))$;
        $[\varphi'] := Uniform(\varphi', P)$;
    case $\varphi' = \langle\langle A\rangle\rangle \mathsf{G}\, \psi$ :
        $S := \bigcup_{S^i \in [\psi]} S^i$;
        $T_1 := \{\langle q, \sharp^k\rangle \mid q \in Dom(S)\}$; $T_2 := \{\langle q, \sharp^k\rangle \mid q \in St\}$;
        while $Dom(T_2) \not\subseteq Dom(T_1)$ do
            $T_2 := T_1$; $T_1 := Pre(A, Dom(T_1))$;
            $T_1 := T_1 \setminus \{\langle q, m\rangle \in T_1 \mid q \notin Dom(S)\}$;
        $[\varphi'] := Uniform(\varphi', T_1)$;
    case $\varphi' = \langle\langle A\rangle\rangle\, \psi_1 \mathsf{U}\, \psi_2$ :
        $S^1 := \bigcup_{S^i \in [\psi_1]} S^i$; $S^2 := \bigcup_{S^i \in [\psi_2]} S^i$;
        $T_1 := \emptyset$; $T_2 := \{\langle q, \sharp^k\rangle \mid q \in Dom(S^2)\}$;
        while $Dom(T_2) \not\subseteq Dom(T_1)$ do
            $T_1 := T_1 \cup T_2$; $T_2 := Pre(A, Dom(T_1))$;
            $T_2 := T_2 \setminus \{\langle q, m\rangle \in T_2 \mid q \notin Dom(S^1)\}$;
        $[\varphi'] := Uniform(\varphi', T_1)$;
    strategies := strategies $\cup \{[\varphi']\}$ ;
return strategies;
```

[3] We will describe this function in detail later on.

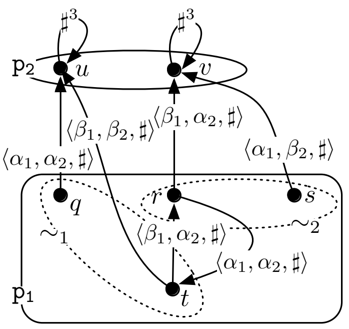

$\sharp^3$
$\sharp^3$
p_2
u
v
$\langle\beta_1,\beta_2,\sharp\rangle$
$\langle\beta_1,\alpha_2,\sharp\rangle$
$\langle\alpha_1,\alpha_2,\sharp\rangle$
$\langle\alpha_1,\beta_2,\sharp\rangle$
q
r
s
$\sim_1$
$\langle\beta_1,\alpha_2,\sharp\rangle$
$\sim_2$
$\langle\alpha_1,\alpha_2,\sharp\rangle$
p_1
t

Fig. 2. The union of all strategies of coalition $A = \{1, 2\}$ for $\varphi = \langle\langle A\rangle\rangle \mathsf{p}_1 \mathsf{U}\, \mathsf{p}_2$. Proposition p_1 holds at states q, r, s and t and p_2 holds at states u and v. Agent 3, excluded from A, has only one action available at each state in $\{q, r, s, t, u, v\}$; thus, the arrows represent A-moves.

Consider the union S of all strategies for a formula $\varphi = \langle\langle A\rangle\rangle\psi$ that is given as an input to *Uniform*. (An example for $\varphi = \langle\langle A\rangle\rangle \mathsf{p}_1 \mathsf{U}\, \mathsf{p}_2$ is depicted in Fig. 2). Note, that the union of two strategies is not necessarily a strategy itself but merely a set of A-moves, as two strategies may assign two different A-moves to the same state. Every maximal uniform strategy for φ must be a subset S' of S. To find every such subset, three issues must be resolved.

– *First*, two A-moves may assign different actions to an agent $a \in A$ in two states that are indistinguishable to a.

Definition 8. *Let $A \subseteq Agt$, and let $\langle q, m_A\rangle$ and $\langle q', m'_A\rangle$ be A-moves. $\langle q, m_A\rangle$ and $\langle q', m'_A\rangle$ are* blocking *each other, symbolically $\langle q, m_A\rangle \leftrightsquigarrow \langle q', m'_A\rangle$, if $m_A[a] \neq m'_A[a]$ and $q \sim_a q'$, for some $a \in A$.*

In Fig. 2, $\langle q, \langle\alpha_1, \alpha_2, \sharp\rangle\rangle$ and $\langle t, \langle\beta_1, \alpha_2, \sharp\rangle\rangle$ are blocking each other; the same holds true for $\langle r, \langle\beta_1, \alpha_2, \sharp\rangle\rangle$ and $\langle s, \langle\alpha_1, \beta_2, \sharp\rangle\rangle$. In general, if S' is a uniform strategy, only one A-move from a set of mutually blocking A-moves may be included in S'.

– *Second*, consider a subset of the union of all strategies for φ that is a uniform strategy (i.e., does not contain any pair of blocking A-moves). Not every such subset is necessarily a uniform strategy for φ.

Assume that S is the union of all strategies for $\varphi = \langle\langle A\rangle\rangle\psi$ and $\langle q, m_A\rangle \in M$. Consider strategy $S' = S \setminus \{\langle q^*, m^*_A\rangle \mid \langle q, m_A\rangle \leftrightsquigarrow \langle q^*, m^*_A\rangle\}$. Now, some $\langle q', m'_A\rangle \in S'$ may have become "disconnected" in S', i.e., for some state $q'' \in out(q', m'_A)$, all A-moves assigned to q'' have been removed from S'. Then, there may be an outcome of S' that is effected by $\langle q', m'_A\rangle$ but does not satisfy ψ. We now define the notion of disconnectedness.

Definition 9. *Let M be a set of A-moves and let $\langle q, m_A\rangle \in M$. We say that $\langle q, m_A\rangle$ is* disconnected *in M if there is $q' \in out(q, m_A)$ such that there is no A-move assigned to q' by M.*

As an example, in Fig. 2, assume that all A-moves at q and r as well as A-move $\langle t, \langle \beta_1, \beta_2, \sharp \rangle\rangle$ are removed from S' because they block either $\langle s, \langle \alpha_1, \beta_2, \sharp \rangle\rangle$ or $\langle t, \langle \beta_1, \alpha_2, \sharp \rangle\rangle$. Thus, $S' = \{\langle u, \sharp^3\rangle, \langle v, \sharp^3\rangle, \langle s, \langle \alpha_1, \beta_2, \sharp\rangle\rangle, \langle t, \langle \beta_1, \alpha_2, \sharp\rangle\rangle\}$. The outcome of the A-move assigned to t is r, but the only A-move assigned to r is in $S \setminus S'$. The A-move $\langle t, \langle \beta_1, \alpha_2, \sharp\rangle\rangle$ is then disconnected in S'. Thus, there is a path $\Lambda = t, r$ in $out(dom(S'), S')$ that does not satisfy $\mathsf{p_1} \mathsf{U} \mathsf{p_2}$.

In any uniform strategy S' for $\varphi = \langle\langle A\rangle\rangle\psi$ returned by *Uniform*(φ, S), every A-move that is disconnected in S' must immediately enforce φ (in such a case the A-move is a singleton strategy for φ). Otherwise, there may be a path in $out(dom(S'), S')$ that does not satisfy φ.

Moreover, a uniform strategy that is a subset of the union of all strategies for $\varphi = \langle\langle A\rangle\rangle\psi_1 \mathsf{U} \psi_2$ may contain A-moves that effect an infinite path that never reaches a state where ψ_2 holds and thus does not satisfy $\psi_1 \mathsf{U} \psi_2$. In Fig. 2, the set $S' = \{\langle u, \sharp^3\rangle, \langle v, \sharp^3\rangle, \langle r, \langle \alpha_1, \alpha_2, \sharp\rangle\rangle, \langle t, \langle \beta_1, \alpha_2, \sharp\rangle\rangle\}$ is a uniform strategy with one possible outcome $\Lambda = r, t, r, t, \ldots$ that does not satisfy $\mathsf{p_1} \mathsf{U} \mathsf{p_2}$.

–*Third*, assume that A-moves that may effect a path that does not satisfy ψ are removed from S'. In our example with $S' = \{\langle u, \sharp^3\rangle, \langle v, \sharp^3\rangle, \langle r, \langle \alpha_1, \alpha_2, \sharp\rangle\rangle, \langle t, \langle \beta_1, \alpha_2, \sharp\rangle\rangle\}$ the A-moves assigned to r and t are removed from S'. S' is now a uniform strategy for $\langle\langle A\rangle\rangle \mathsf{p_1} \mathsf{U} \mathsf{p_2}$, but it is not maximal. In Fig. 2, the set $T = S' \cup \{\langle q, \langle \alpha_1, \alpha_2, \sharp\rangle\rangle, \langle s, \langle \alpha_1, \beta_2, \sharp\rangle\rangle\}$ is a superset of S' that is a uniform strategy for $\langle\langle A\rangle\rangle \mathsf{p_1} \mathsf{U} \mathsf{p_2}$ (even a maximal one).

In our example, there are five maximal uniform strategies for $\varphi = \langle\langle A\rangle\rangle \mathsf{p_1} \mathsf{U} \mathsf{p_2}$: $S^1 = \{\langle q, \langle \alpha_1, \alpha_2, \sharp\rangle\rangle, \langle r, \langle \beta_1, \alpha_2, \sharp\rangle\rangle, \langle u, \sharp^3\rangle, \langle v, \sharp^3\rangle\}$, $S^2 = \{\langle q, \langle \alpha_1, \alpha_2, \sharp\rangle\rangle, \langle s, \langle \alpha_1, \beta_2, \sharp\rangle\rangle, \langle u, \sharp^3\rangle, \langle v, \sharp^3\rangle\}$, $S^3 = \{\langle t, \langle \beta_1, \alpha_2, \sharp\rangle\rangle, \langle r, \langle \beta_1, \alpha_2, \sharp\rangle\rangle, \langle u, \sharp^3\rangle, \langle v, \sharp^3\rangle\}$, $S^4 = \{\langle t, \langle \beta_1, \beta_2, \sharp\rangle\rangle, \langle r, \langle \alpha_1, \alpha_2, \sharp\rangle\rangle, \langle u, \sharp^3\rangle, \langle v, \sharp^3\rangle\}$ and $S^5 = \{\langle t, \langle \beta_1, \beta_2, \sharp\rangle\rangle, \langle s, \langle \alpha_1, \beta_2, \sharp\rangle\rangle, \langle u, \sharp^3\rangle, \langle v, \sharp^3\rangle\}$.

Remark 1. A set of A-moves S is a uniform strategy iff there is no pair of blocking A-moves in S.

Remark 2. Let $\varphi = \langle\langle A\rangle\rangle\psi$ be an ATL$_u$-formula, let S be the union of all strategies for φ and let $S' \subseteq S$ be a uniform strategy for φ. S' is a maximal uniform strategy for φ iff it is a uniform strategy for φ and there is no other uniform strategy S'' for φ such that $S' \subset S'' \subseteq S$.

Proposition 1. *Let $\varphi = \langle\langle A\rangle\rangle\psi$ be an ATL$_u$-formula and let S be the union of all strategies for φ. A strategy $S' \subseteq S$ is a strategy for φ iff*

1. *every A-move disconnected in S' is a singleton strategy for φ and*
2. *in case that $\varphi = \langle\langle A\rangle\rangle\psi_1 \mathsf{U} \psi_2$, there is no infinite $\Lambda \in out(dom(S'), S')$ such that $\{\Lambda[i]\} \not\models \psi_2$ for every $i \geq 1$.*

Proof. We have three cases to consider:

– $\varphi = \langle\langle A\rangle\rangle \mathsf{X}\, \psi$ – A strategy S' of coalition A is a strategy for $\langle\langle A\rangle\rangle \mathsf{X}\, \psi$ iff $\{\Lambda[2]\} \models \psi$ for every $\Lambda \in out(dom(S'), S')$ iff $out(q, m_A) \models \psi$, for every $\langle q, m_A\rangle \in S'$. Thus, every $\{\langle q, m_A\rangle\} \subseteq S'$, disconnected or not, is a singleton strategy for $\langle\langle A\rangle\rangle \mathsf{X}\, \psi$. Since S is the union of all strategies for $\langle\langle A\rangle\rangle \mathsf{X}\, \psi$, it consists solely of A-moves that are singleton strategies for $\langle\langle A\rangle\rangle \mathsf{X}\,\psi$. The outcome of every strategy that is a subset of S thus contains only such paths Λ that $\{\Lambda[2]\} \models \psi$. Thus, every strategy $S' \subseteq S$ is a strategy for $\langle\langle A\rangle\rangle \mathsf{X}\,\psi$.

- $\varphi = \langle\langle A\rangle\rangle \mathsf{G}\,\psi$ – ($\Rightarrow$) S' is a strategy for $\langle\langle A\rangle\rangle \mathsf{G}\,\psi$ iff (a) every $\Lambda \in out(dom(S'), S')$ is infinite and (b) $\{\Lambda[i]\} \models \psi$ for every $i \geq 1$. It follows from (a) that no A-move is disconnected in S', as the latter would imply the existence of a finite path in S'.

  ($\Leftarrow$) We argue by contraposition. Since S' is a subset of the union of all strategies for $\langle\langle A\rangle\rangle \mathsf{G}\,\psi$, for every $\Lambda \in out(dom(S'), S')$ and every $1 \leq i \leq |\Lambda|$, we have $\{\Lambda[i]\} \models \psi$. Thus, if strategy $S' \subseteq S$ is not a strategy for $\langle\langle A\rangle\rangle \mathsf{G}\,\psi$, this can only happen if some $\Lambda \in out(dom(S'), S')$ is finite. The last state of every such Λ must be in the outcome of some A-move disconnected in S'. Since a single A-move can only be a (singleton) strategy for $\langle\langle A\rangle\rangle \mathsf{G}\,\psi$ if it "loops back", which is incompatible with being disconnected, none of these A-moves can be a strategy for $\langle\langle A\rangle\rangle \mathsf{G}\,\psi$.
- $\varphi = \langle\langle A\rangle\rangle \psi_1 \mathsf{U}\,\psi_2$ – ($\Rightarrow$) S' is a strategy for $\langle\langle A\rangle\rangle \psi_1 \mathsf{U}\,\psi_2$ iff, for every $\Lambda \in out(dom(S'), S')$, there is $j \geq 1$ such that $\{\Lambda[j]\} \models \psi_2$ and $\{\Lambda[i]\} \models \psi_1$, for every $1 \leq i \leq j$. Thus, there is no infinite $\Lambda \in out(dom(S'), S')$ such that $\{\Lambda[i]\} \not\models \psi_2$ for every $i \geq 1$. If $\langle q, m_A\rangle \in S'$ is an A-move disconnected in S', then either $\{q\} \models \psi_2$ must hold or $\{q'\} \models \psi_2$ must hold for every $q' \in out(q, m_A)$ (otherwise there would be a path $\Lambda \in out(q, S')$ such that $\{\Lambda[i]\} \models \psi_2$ does not hold for any $i \geq 1$). Thus, every A-move disconnected in S' is a singleton strategy for $\langle\langle A\rangle\rangle \psi_1 \mathsf{U}\,\psi_2$.

  ($\Leftarrow$) We argue by contraposition. Since S is the union of all strategies for $\langle\langle A\rangle\rangle \psi_1 \mathsf{U}\,\psi_2$, for every A-move $\langle q, m_A\rangle \in S$ we have that either $\{q\} \models \psi_2$ or $\{q\} \models \psi_1$ and $\{q'\} \models \psi_1$ or $\{q'\} \models \psi_2$ for every $q' \in out(q, m_A)$. Thus, if strategy $S' \subseteq S$ is not a strategy for $\langle\langle A\rangle\rangle \psi_1 \mathsf{U}\,\psi_2$, this can only happen if, for some $\Lambda \in out(dom(S'), S')$, we have $\{\Lambda[i]\} \not\models \psi_2$, for every $i \geq 1$. If such Λ is infinite then condition 2 is not satisfied. If such Λ is finite, then the last state q of Λ must be in the outcome of some A-move $\langle q', m_A\rangle$ disconnected in S'. For this $q \in out(q', m_A)$, we have $\{q\} \not\models \psi_2$; thus, $\{\langle q', m_A\rangle\}$ cannot be a singleton strategy for $\langle\langle A\rangle\rangle \psi_1 \mathsf{U}\,\psi_2$ and condition 1 is not satisfied. □

The function *Uniform*(φ, S), which returns all maximal uniform strategies for φ, is described in Alg. 2. A strategic formula φ and the union S of all strategies for φ is passed as the input parameter.

Uniform(φ, S) works as follows. First, the problem of avoiding blocking pairs of A-moves in a resulting maximal uniform strategy for φ is solved via reduction to the problem of listing all maximal cliques in the components of a graph derived from the blocking relation between the A-moves. (The Bron-Kerbosh algorithm [3] can be used for listing of all maximal cliques.) Afterwards, in case that $\varphi = \langle\langle A\rangle\rangle \mathsf{G}\psi$, we remove from every uniform strategy S' maximal in S all disconnected assigned A-moves. In case that $\varphi = \langle\langle A\rangle\rangle \psi_1 \mathsf{U}\,\psi_2$, we remove all A-moves that can effect a path that never reaches a state from the domain of any uniform strategy for ψ_2. Thus, only uniform strategies for φ remain. Finally,

Function. `Uniform`(φ, S)

```
Input: a strategic formula $\varphi = \langle\langle A\rangle\rangle\psi$, the union $S$ of all strategies for $\varphi$
Output: all maximal uniform strategies for $\varphi$
// find set $\mathbb{S}$ of all uniform strategies that are maximal in $S$
build a graph $G = \langle S, B\rangle$ where $(\langle q, m_A\rangle, \langle q', m'_A\rangle) \in B$ iff $\langle q, m_A\rangle \leftrightsquigarrow \langle q', m'_A\rangle$ ;
find all components $C_1, .., C_m$ of $G$;
foreach component $C_i$ of $G$ do
    /* find all maximal independent sets $I_i^1, .., I_i^n$ in $C_i$ */
    build the complement graph $\overline{C}_i$ of $C_i$;
    $\mathbb{I}_i := \{$all maximal cliques $I_i^1, .., I_i^l$ of $\overline{C}_i\}$ ;    // Bron-Kerbosh algorithm
// generate all combinations of cliques, one clique per component
$\mathbb{S} := \{\emptyset\}$;
foreach component $C_i$ of $G$ do
    $\mathbb{S}' := \{\emptyset\}$;
    foreach $S_j \in \mathbb{S}$ do
        foreach clique $I_i^k \in \mathbb{I}_i$ do
            $\mathbb{S}' := \mathbb{S}' \cup \{I_i^k \cup S_j\}$;
    $\mathbb{S} := \mathbb{S}'$;
if $\varphi = \langle\langle A\rangle\rangle \mathsf{X}\psi$ then return $\mathbb{S}$;
else if $\varphi = \langle\langle A\rangle\rangle \mathsf{G}\psi$ then
    // keep in each $S_i$ only $A$-moves that are not disconnected in $S_i$
    foreach $S_i \in \mathbb{S}$ do
        $T := \emptyset$;
        while $S_i \neq T$ do $T := S_i; S_i := \{\langle q, m_A\rangle \in S_i \mid out(q, m_A) \subseteq Dom(S_i)\}$;
else if $\varphi = \langle\langle A\rangle\rangle \psi_1 \mathsf{U} \psi_2$ then
    // remove from each $S_i$ all $A$-moves that may effect a path never
       reaching $\bigcup_{S_j \in [\psi_2]} dom(S_j)$
    $D = \bigcup_{S_j \in [\psi_2]} Dom(S_j)$;
    foreach $S_i \in \mathbb{S}$ do
        $T_1 := \emptyset$; $T_2 := \{\langle q, m_A\rangle \in S_i \mid q \subseteq D\}$;
        while $T_2 \not\subseteq T_1$ do
            $T_1 := T_1 \cup T_2; T_2 := \{\langle q, m_A\rangle \in S_i \mid out(q, m_A) \subseteq Dom(T_1)\}$;
        $S_i := T_1$;
// remove all non-maximal strategies
foreach $S_i, S_j \in \mathbb{S}$ do
    if $S_i \subseteq S_j$ then $\mathbb{S} := \mathbb{S} \setminus S_i$;
return $\mathbb{S}$;
```

every strategy that is a subset of another strategy is removed. The remaining strategies are maximal uniform strategies for φ.

Proposition 2. *Let S be the union of all strategies for a strategic ATL_u-formula $\varphi = \langle\langle A\rangle\rangle\psi$ and let $\mathbb{S}$ be the set returned by Uniform(φ, S). S' is a maximal uniform strategy for φ iff $S' \in \mathbb{S}$.*

Proof. First, we show that, as an intermediate step, (1) *Uniform*(φ, S) finds all uniform strategies that are maximal in S. According to Remark 1, a set of A-moves S' is a uniform strategy iff for every $\langle q, m_A\rangle \in S'$ there is no $\langle q', m'_A\rangle \in S'$ such that $\langle q, m_A\rangle \leftrightsquigarrow \langle q', m'_A\rangle$. Consider graph $G = \langle S, B\rangle$ where $(\langle q, m_A\rangle, \langle q', m'_A\rangle) \in B$ iff $\langle q, m_A\rangle \leftrightsquigarrow \langle q', m'_A\rangle$. There is no $\langle q, m_A\rangle, \langle q', m'_A\rangle \in S' \subseteq S$ such that $\langle q, m_A\rangle \leftrightsquigarrow \langle q', m'_A\rangle$ iff S' is an independent set in G, i.e., a clique (a complete subgraph) in the complement graph $\overline{G}$. S' is a uniform strategy maximal in S iff it is a maximal independent set in G. Let $C_1, \ldots, C_n$ be all disconnected components of G. For every $1 \leq i \leq n, 1 \leq j \leq n, i \neq j$ there is no $\langle q, m_A\rangle \in C_i$ and $\langle q', m'_A\rangle \in C_j$ such that $\langle q, m_A\rangle \leftrightsquigarrow \langle q', m'_A\rangle$. Thus, the set of all maximal independent sets in G is $\{\bigcup_{i=1,\ldots,n} I_i \mid I_i \textit{ is a maximal independent set of } C_i\}$. Function *Uniform* finds all combinations of the maximal independent sets from every disconnected component of G, one set per component, thus producing (as an intermediate step) the set of all uniform strategies that are maximal in S.

Next, we show that (2) if a set of A-moves S' is a uniform strategy for φ then $S' \subseteq S''$ for some $S'' \in \mathbb{S}$. Since S' is a strategy for φ, $S' \subseteq S$. Since $S' \subseteq S$ is a uniform strategy, there exists some uniform strategy M maximal in S such that $S' \subseteq M$. According to (1), every such M is found in *Uniform*(φ, S). We have three cases to consider:

– $\varphi = \langle\langle A\rangle\rangle \mathsf{X}\rho$ – Every uniform strategy maximal in S is directly returned in $\mathbb{S}$. Thus, $M \in \mathbb{S}$ and $S' \subseteq M$.
– $\varphi = \langle\langle A\rangle\rangle \mathsf{G}\rho$ – For every uniform strategy M maximal in S, *Uniform*(φ, S) finds the greatest fixed point P_g of function $F(X) = \{\langle q, m_A\rangle \in M \mid out(q, m_A) \subseteq Dom(X)\}$ where the domain of F is the power set of M. Assume that there exists $\langle q, m_A\rangle \in S' \setminus P_g$. Then, $out(q, m_A) \not\subseteq S'$ and thus there exists a finite $\Lambda \in out(q, S')$. Hence, S' is not a strategy for $\langle\langle A\rangle\rangle \mathsf{G}\rho$, which is a contradiction. Thus, every $\langle q, m_A\rangle \in S'$ must be in $P_g \subseteq M$. $P_g \notin \mathbb{S}$ iff $P_g \subset S''$ for some $S'' \in \mathbb{S}$. Hence, either P_g or some S'' such that $P_g \subset S''$ is in $\mathbb{S}$ and $S' \subseteq P_g$.
– $\varphi = \langle\langle A\rangle\rangle \rho_1 \mathsf{U} \rho_2$ – For every uniform strategy M maximal in S, *Uniform*(φ, S) finds the least fixed point P_l of function $G(X) = \{\langle q, m_A\rangle \in M \mid \{q\} \models \rho_2\} \cup \{\langle q, m_A\rangle \in M \mid out(q, m_A) \subseteq Dom(X)\}$ where the domain of G is the power set of M. Assume that there exists $\langle q, m_A\rangle \in S' \setminus P_l$. Then, there exists $\Lambda \in out(q, S')$ such that $\{\Lambda[i]\} \not\models \rho_2$ for every $1 \leq i \leq |\Lambda|$. Hence, S' is not a strategy for $\langle\langle A\rangle\rangle \rho_1 \mathsf{U} \rho_2$, which is a contradiction. Thus, every $\langle q, m_A\rangle \in S'$ must be in $P_l \subseteq M$. $P_l \notin \mathbb{S}$ iff $P_l \subset S''$ for some $S'' \in \mathbb{S}$. Hence, either P_l or some S'' such that $P_l \subset S''$ is in $\mathbb{S}$ and $S' \subseteq P_l$.

Next, we show that (3) every $S' \in \mathbb{S}$ is a uniform strategy for φ. According to (1), *Uniform*(φ, S) finds all uniform strategies that are maximal in S. Only such strategies or their nonempty subsets are included in $\mathbb{S}$. Since every nonempty subset of a uniform strategy is a uniform strategy, every $S' \in \mathbb{S}$ is uniform strategy. We have three cases to consider:

- $\varphi = \langle\langle A\rangle\rangle \mathsf{X}\rho$ – Every $S' \in \mathbb{S}$ is a subset of S and consists solely of A-moves that are singleton strategies for $\langle\langle A\rangle\rangle \mathsf{X}\rho$. It follows from Proposition 1 that every such S' is a strategy for $\langle\langle A\rangle\rangle \mathsf{X}\rho$.
- $\varphi = \langle\langle A\rangle\rangle \mathsf{G}\rho$ – Every $S' \in \mathbb{S}$ is a subset of S and the greatest fixed point of function $F(X) = \{\langle q, m_A\rangle \in M \mid out(q, m_A) \subseteq Dom(X)\}$ where the domain of F is the power set of M and M is some uniform strategy maximal in S. Since $out(q, m_A) \subseteq S'$ for every $\langle q, m_A\rangle$, there is no disconnected A-move in S' and according to Proposition 1, S' is a strategy for $\langle\langle A\rangle\rangle \mathsf{G}\rho$.
- $\varphi = \langle\langle A\rangle\rangle \rho_1 \mathsf{U}\, \rho_2$ – Every $S' \in \mathbb{S}$ is a subset of S and is the least fixed point of function $G(X) = \{\langle q, m_A\rangle \in M \mid \{q\} \models \rho_2\} \cup \{\langle q, m_A\rangle \in M \mid out(q, m_A) \subseteq Dom(X)\}$ where the domain of G is the power set of M and M is some uniform strategy maximal in S. For every $\langle q, m_A\rangle \in S'$ either $\{q\} \models \rho_2$ and thus $\langle q, m_A\rangle \in S'$ is a singleton strategy for $\langle\langle A\rangle\rangle \rho_1 \mathsf{U}\, \rho_2$ or for every $\Lambda \in out(q, S')$ we have $\{\Lambda[i]\} \models \rho_2$ for some i. Thus, according to Proposition 1, S' is a strategy for $\langle\langle A\rangle\rangle \rho_1 \mathsf{U}\, \rho_2$.

Next, we show that (4) if a set of A-moves S' is a maximal uniform strategy for φ then $S' \in \mathbb{S}$. It follows from (2) that for every maximal uniform strategy S' for φ there exists $S'' \in \mathbb{S}$ such that $S' \subseteq S''$. Since (3), S'' is a uniform strategy for φ. Since S' is a maximal uniform strategy for φ, it is not a proper subset of any other uniform strategy for φ. Thus, $S' = S'' \in \mathbb{S}$.

Lastly, we show that (5) every $S' \in \mathbb{S}$ is a maximal uniform strategy for φ. According to (3), every $S' \in \mathbb{S}$ is a uniform strategy for φ. If some uniform strategy $S' \in \mathbb{S}$ for φ is not maximal, then there must exist a maximal uniform strategy S'' for φ such that $S' \subset S''$. Since for every $S' \in \mathbb{S}$ there is no $S'' \in \mathbb{S}$ such that $S' \subset S''$, we have that $S'' \notin \mathbb{S}$. According to (4), $S'' \in \mathbb{S}$, which is a contradiction and thus S' must be a maximal uniform strategy for φ.

Since (4) and (5), S' is a maximal uniform strategy iff $S' \in \mathbb{S}$. □

We now prove the correctness of the algorithm for finding uniform strategies:

Claim. Given an ATL$_u$-formula φ, the algorithm 1 returns all maximal uniform strategies for every subformula ψ of φ (including φ itself).

Proof. For the case that ψ is a non-strategic formula (i.e., either ψ is an atomic proposition, or $\psi = \neg\rho$, or $\psi = \rho_1 \wedge \rho_2$), the strategy $S_\emptyset$ returned by the algorithm assigns the $\emptyset$-move to every state that satisfies ψ. Thus, the domain of $S_\emptyset$ is the set of states where ψ holds. As every strategy consisting of $\emptyset$-moves is uniform, $S_\emptyset$ is uniform. Since there is no strategy S for ψ such that $dom(S_\emptyset) \subset dom(S)$, $S_\emptyset$ is the maximal uniform strategy for ψ.

For the case that $\psi = \langle\langle A\rangle\rangle \tau$, we show that the set of A-moves passed to function *Uniform* is the union of all strategies for ψ:

- $\psi = \langle\langle A\rangle\rangle \mathsf{X}\rho$ – The arguments for function *Pre* are coalition A and the set $Q = \bigcup_{S^i \in [\rho]} dom(S^i)$. Function *Pre* returns a set P of all A-moves $\langle q, m_A\rangle$ such that $out(q, m_A) \subseteq Q$. A set of A-moves S is a strategy for $\langle\langle A\rangle\rangle \mathsf{X}\rho$ iff $\{\Lambda[2]\} \models \rho$ for every $\Lambda \in out(dom(S), S)$ iff $out(q, m_A) \subseteq Q$ for every

$\langle q, m_A \rangle \in S$. Thus, every strategy for $\langle\!\langle A \rangle\!\rangle \mathsf{X} \rho$ is a subset of P. Every A-move in P is a singleton strategy for $\langle\!\langle A \rangle\!\rangle \mathsf{X} \rho$. Thus, P is the union of all strategies for $\langle\!\langle A \rangle\!\rangle \mathsf{X} \rho$.

– $\psi = \langle\!\langle A \rangle\!\rangle \mathsf{G} \rho$ – The set T_1 passed to *Uniform* is the greatest fixed point of $F(X) = \{\langle q, m_A \rangle \in Pre(A, Dom(X)) \mid \{q\} \models \rho\}$. For every strategy $S \not\subseteq T_1$ there is $\langle q, m_A \rangle \in S \setminus T_1$ such that there is $\Lambda \in out(q, S)$ that is either finite or $\{\Lambda[i]\} \not\models \rho$ for some i and hence, $\Lambda \not\Vdash \mathsf{G} \rho$. Thus, every strategy for $\langle\!\langle A \rangle\!\rangle \mathsf{G} \rho$ must be a subset of T_1. For every $\langle q, m_A \rangle \in T_1$ there exists a strategy $S \subseteq T_1$ for $\langle\!\langle A \rangle\!\rangle \mathsf{G} \rho$ such that $\langle q, m_A \rangle \in S$. Thus, T_1 is the union of all strategies for $\langle\!\langle A \rangle\!\rangle \mathsf{G} \rho$.

– $\psi = \langle\!\langle A \rangle\!\rangle \rho_1 \mathsf{U} \rho_2$ – The set T_1 passed to *Uniform* is the least fixed point of $G(X) = \{\langle q, \sharp^k \rangle \mid \{q\} \models \rho_2\} \cup \{\langle q, m_A \rangle \in Pre(A, Dom(X)) \mid \{q\} \models \rho_1\}$. For every strategy $S \not\subseteq T_1$ there is $\langle q, m_A \rangle \in S \setminus T_1$ such that $\{q\} \not\models \rho_2$ and $\{q'\} \not\models \rho_2$ and $\{q'\} \not\models \rho_1$ for some $q' \in out(q, m_A)$. Hence, $\Lambda \not\Vdash \rho_1 \mathsf{U} \rho_2$ for some $\Lambda \in out(q, S)$. Thus, every strategy for $\langle\!\langle A \rangle\!\rangle \rho_1 \mathsf{U} \rho_2$ must be a subset of T_1. For every $\langle q, m_A \rangle \in T_1$ there exists a strategy $S \subseteq T_1$ for $\langle\!\langle A \rangle\!\rangle \rho_1 \mathsf{U} \rho_2$ such that $\langle q, m_A \rangle \in S$. Thus, T_1 is the union of all strategies for $\langle\!\langle A \rangle\!\rangle \rho_1 \mathsf{U} \rho_2$.

Thus, for every $\psi = \langle\!\langle A \rangle\!\rangle \tau$ that is a subformula of φ, the union S of all strategies for ψ is passed to function *Uniform*(ψ, S) and according to Proposition 2, the set of all maximal uniform strategies for ψ is returned. □

The approach used in our algorithm is similar to model checking for Constructive Strategic Logic [5] — first guess a uniform strategy and then verify with the CTL model checking algorithm that it is a strategy for a given formula. However, our algorithm provides a method for construction of the uniform strategies, performs only necessary checks to avoid full-scale CTL model checking and returns all maximal uniform strategies of a given formula, rather then just a set of all states which satisfy the formula.

4 Example

To demonstrate the algorithm, we use the example of the SOSEWIN sensor network for detection of earthquakes developed in the scope of the SAFER project [1]. This network is partitioned into clusters of a fixed size. Every node is connected to two other nodes in the cluster. A protocol for election of a leader is necessary to fulfill the task of the network. The aim of the protocol is to establish exactly one leader in the cluster. For the sake of simplicity, we fix the size of the cluster to three nodes and further limit the information available to each node by reducing the size of their neighborhood.

We represent the cluster of three nodes as a one-dimensional cellular automaton – a row of three cells. Each cell may have one of two colors—black when the cell is designated as the leader and white otherwise—and is connected to the cell on its left and right side. The neighborhood of each cell consists of the cell on its left, that is, each cell knows its own color and the color of the cell on its left. The neighbor of the leftmost cell is the rightmost cell. In each step of the system each cell synchronously observes the colors in its neighborhood and decides to either keep its current color or swap it, according to an applicable rule. Given a cell and the colors of the cell's neighborhood the rule defines the next cell's color. Thus, a set of rules for all cells can be seen as a uniform strategy of the cellular automaton.

We want to find all maximal uniform strategies for the following property: In one step a state of the cellular automaton is reached, where exactly one cell is black and the other two are white. Moreover, if the cellular automaton is at such state, the cells may not change their colors.

First, we specify a CEGS representing our system:

- Each cell is one agent. We denote the leftmost cell by c_1, the middle cell by c_2 and the rightmost cell by c_3. Thus, the set of agents is $Agt = \{c_1, c_2, c_3\}$. Since all cells are involved in enforcing the property, they are all included in the coalition A.
- A state of the system is given by the colors of all three cells. We denote the state where the cell c_1 is black and the cell c_2 and c_3 is white by ■□□. The set of all states is $St = \{$□□□, □□■, □■□, □■■, ■□□, ■□■, ■■□, ■■■$\}$.
- There is a single atomic proposition $\mathtt{p}$ that holds in those states where exactly one cell is black. That is, $\pi(\mathtt{p}) = \{$□□■, □■□, ■□□$\}$.
- Each cell can either keep its color (denoted by $-$) or swap it (denoted by s). Thus, the set of actions is $Act = \{-, s\}$.
- Each cell has both actions available at all states except those where proposition $\mathtt{p}$ holds. In states where $\mathtt{p}$ holds only action – is available to every cell (this constraint is a part of the property that the system should enforce). Thus, for every cell $c \in Agt$ we have $d(c, q) = \{-\}$ for every state $q \in \pi(\mathtt{p})$ and $d(c, q') = \{-, s\}$ for every state $q' \notin \pi(\mathtt{p})$.
- The transition function o can be derived from the description of the action. For example, $o($■□■$, \langle -, -, s\rangle) =$ ■□□.
- Since each cell knows only its own color and the color of the cell on its left, the two states that differ in the color of the cell on its right are indistinguishable for the cell.

We can now express the desired property formally: $\varphi = \langle\langle A\rangle\rangle \mathsf{X}\mathtt{p}$. Algorithm 1 calls function $Pre(A, \pi(\mathtt{p}))$ to find the union S of all strategies for φ. S consists of A-moves represented by the nodes in Fig. 3. The domain of S consists of all system states.

Function $Uniform(\varphi, S)$ constructs the graph G for the blocking relation on S (Fig. 3). There is only one component in G and one maximal independent set in G is depicted in grey in the figure. In case of *next* operator every A-move in S is a singleton strategy for φ. Thus, every maximal independent set in G represents one maximal uniform strategy φ.

There are four maximal uniform strategies for φ. From these four strategies only the one depicted on the figure consists of such A-moves that to every cell the same action is prescribed for the same combination of its color and the color of its left neighbor. From this strategy one set of rules can be derived that can be performed by every cell, namely: swap the color if both colors are black, keep the

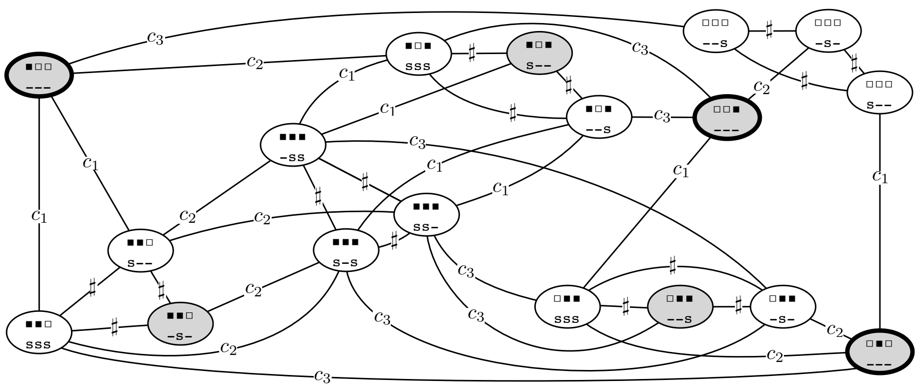


Fig. 3. The union of all strategies S for $\varphi = \langle\langle A\rangle\rangle \mathsf{X}\mathtt{p}$: Each vertex is an A-move. An edge connects A-moves that block each other and the label of the edge denotes the agent to that the connected states are indistinguishable. $\sharp$ denotes all agents. Proposition $\mathtt{p}$ is true at the states from the bold vertices. One maximal uniform strategy for φ (a maximal independent set) consists of the grey vertices.

color otherwise. The system following this strategy reaches a state from $\pi(\mathtt{p})$ in one step from all states except ■■■ and □□□ (while respecting the constraint on the behavior at states from $\pi(\mathtt{p})$).

5 The Complexity of Finding Uniform Strategies

The input of the algorithm 1 is a CEGS $\mathfrak{M}$ with m transitions and an ATL$_u$ formula φ of length l.

For each subformula ψ of φ, the algorithm first finds the union S of all strategies for ψ. In the worst case, this involves computing the least or the greatest fixed point of *Pre*, which finds a pre-image for a given set of states and a coalition of agents. Thus, every transition in $\mathfrak{M}$ must be checked and finding the union S of all strategies for φ takes $O(m)$ steps.

Next, for each subformula ψ, *Uniform*(ψ, S) constructs the graph $G = \langle S, B\rangle$ for the blocking relation on the A-moves from S. The size of S is at most m. To decide whether a pair of vertices is connected by an edge, the actions from the A-moves are compared for each agent from A that cannot distinguish between the states to that the A-moves are assigned. This involves at most k comparisons for a pair of vertices, where $k = |Agt|$. As the presence of the edge is decided for every pair of vertices, the construction of G takes $O(m^2)$ steps. The disconnected components of G can be identified during the construction for no extra price.

Next, each disconnected component C of graph G is turned into its complement graph $\overline{C}$. In the worst case, G consists of only one component and the construction of $\overline{G}$ requires $O(|S|^2) \leq O(m^2)$ steps.

Next, all maximal cliques are found in every complement graph $\overline{C}$. The Bron-Kerbosh algorithm solving this task has the worst-time complexity $O(3^{v/3})$ for a graph with v vertices [11] and is optimal since there are at most $3^{v/3}$ maximal cliques in such graph [9]. If G consists of j disconnected components $C_1, \ldots, C_j$ with $c_1, \ldots, c_j$ vertices, respectively, then the task requires $O(\sum_{i=1}^{j} 3^{c_i/3}) \leq O(3^{m/3})$ steps. Then, all combinations of maximal cliques – one for each disconnected complement component – are generated to provide all uniform strategies maximal in S. Since there is at most $3^{c_i/3}$ maximal cliques in $\overline{C_i}$, we need up to $O(\prod_{i=1}^{j} 3^{c_i/3}) \leq O(3^{m/3})$ steps to produce all uniform strategies maximal in S.

Next, in some cases the least or greatest fixed point in the uniform strategy S' maximal in S is computed. At worst case, every transition from $out(dom(S'), S')$ must be checked for the presence in the fixed point and thus, $O(m)$ steps are necessary. Since there is up to $O(3^{m/3})$ uniform strategies maximal in S, the computation takes up to $O(m \cdot 3^{m/3})$ steps.

Lastly, every strategy that is a subset of another strategy is removed so that only maximal uniform strategies for ψ remain. At worst case, every pair of strategies must be compared (at most $3^{2 \cdot m/3}$ comparisons) and each comparison may involve checks for each pair of A-moves (at most m^2 checks). Thus, removing the non-maximal strategies may take at most $O(m^2 \cdot 3^{2 \cdot m/3})$ steps.

Since the same procedure is done for every strategic subformula of φ, the worst-case complexity of finding all maximal uniform strategies for every subformula of φ including φ itself is $O(l \cdot m^2 \cdot 3^{2 \cdot m/3})$.

If filtering out non-maximal uniform strategies for φ is not required, the worst-case complexity is $O(l \cdot m \cdot 3^{m/3})$. This result is in line with the fact that model checking complexity of ATL$_{ir}$ is Δ_2^P-complete [7], i.e., the problem can be solved in polynomial time with an oracle for some NP-complete problem. In our case, finding all uniform strategies maximal in S is reduced to the problem if listing all maximal cliques, which is NP-complete. With the routine for finding such strategies replaced by the oracle for this problem, our algorithm would work in polynomial time. Note, however, that our algorithm not only finds the states where a formula holds, but also returns all maximal uniform strategies for the formula.

6 Conclusion

We presented an algorithm for finding uniform strategies for multi-agent systems with agents with incomplete information. Given a strategic formula $\varphi = \langle\langle A \rangle\rangle \psi$, we find all maximal uniform strategies that agents in A can use to enforce ψ, rather then simply the set of states satisfying φ. The formulas we consider have the syntax of ATL, but we define their semantics in terms of uniform strategies. The algorithm has exponential worst-case complexity, since it uses, as a subroutine, a procedure for finding all maximal cliques in a graph—a problem that is known to be NP-complete. Further research will be focused on reducing the complexity of the algorithm by using a model with an implicit representation of the incomplete information, e.g., modular interpreted systems [6] and with symbolic representation of the system state space.

Acknowledgements. We thank anonymous reviewers for their helpful comments. This work is supported by the grant SUA 08/034 from the National Research Foundation of South Africa and the German Academic Exchange Serivce DAAD, as well as by the Deutsche Forschungsgemeinschaft GRK 1324/1 (METRIK).

References

1. SAFER - Seismic eArly warning For EuRope (2010), `http://www.saferproject.net`
2. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM 49(5), 672–713 (2002)
3. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Commun. ACM 16(9), 575–577 (1973)
4. Goranko, V., Shkatov, D.: Tableau-based decision procedures for logics of strategic ability in multi-agent systems. ACM Transactions on Computational Logic (2009)
5. Jamroga, W., Ågotnes, T.: Constructive knowledge: what agents can achieve under imperfect information. Journal of Applied Non-classical Logics 17(4), 423–475 (2007)
6. Jamroga, W., Ågotnes, T.: Modular interpreted systems. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2007 (2007)
7. Jamroga, W., Dix, J.: Model checking abilities under incomplete information is indeed delta2-complete. In: Proceedings of the 4th European Workshop on Multi-Agent Systems, EUMAS 2006 (2006)
8. Jamroga, W.: Some remarks on alternating temporal epistemic logic. In: Dunin-Keplicz, B., Verbrugge, R. (eds.) Formal Approaches to Multiagent Systems 2003, pp. 133–140 (2003)
9. Moon, J., Moser, L.: On cliques in graphs. Israel Journal of Mathematics 3(1), 23–28 (1965)
10. Schobbens, P.-Y.: Alternating-time logic with imperfect recall. Electronic Notes in Theoretical Computer Science 85(2), 82–93 (2004)
11. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical Computer Science 363(1), 28–42 (2006)
12. van der Hoek, W., Wooldridge, M.: Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. Studia Logica (2003)

Playing Extensive Form Games in Parallel

Sujata Ghosh[1,⋆], R. Ramanujam[2] and Sunil Simon[3,⋆⋆]

[1] Department of Artificial Intelligence
University of Groningen
sujata@ai.rug.nl

[2] The Institute of Mathematical Sciences
C.I.T. Campus, Chennai 600 113, India
jam@imsc.res.in

[3] Institute for Logic, Language and Computation
University of Amsterdam
s.e.simon@uva.nl

Abstract. Consider a player playing against different opponents in two extensive form games simultaneously. Can she then have a strategy in one game using information from the other? The famous example of playing chess against two grandmasters simultaneously illustrates such reasoning. We consider a simple dynamic logic of extensive form games with sequential and parallel composition in which such situations can be expressed. We present a complete axiomatization and show that the satisfiability problem for the logic is decidable.

1 Motivation

How can any one of us[1] expect to win a game of chess against a Grandmaster (GM)? The strategy is simple: play simultaneously against two Grandmasters! If we play black against GM 1 playing white, and in the parallel game play white against GM 2 playing black, we can do this simply. Watch what GM 1 plays, play that move in the second game, get GM 2's response, play that *same* move as our response in game 1, and repeat this process. If one of the two GMs wins, we are assured of a win in the other game. In the worst case, both games will end in a draw.

Note that the strategy construction in this example critically depends on several features:

– Both games need to be played in *lock-step synchrony*; if they are slightly out of step with each other, or are sequentialized in some way, the strategy is not applicable. So concurrency is critically exploited.

⋆ Supported by the Netherlands Organisation of Scientific Research grant 600.065.120.08N201.

⋆⋆ Supported by the European Commission grant EMECW15.

[1] By "us" we mean poor mortals who know how to play the game but lack expertise.

- The strategy cannot be constructed *a priori*, as we do not know what moves would be played by either of the GMs. Such reasoning is intrinsically different from the discussion of the existence of winning strategies in determined games. In particular, strategic reasoning as in normal form games is not applicable.
- The common player in the two games acts as a conduit for transfer of information from one game to the other; thus *game composition* is essential for such reasoning. The example illustrates that playing several instances of the same game may mean something very different from repeated games.
- The common player can be a resource bounded agent who cannot analyse the entire game structure and compute the winning strategy (even if it exists). The player thus mimics the moves of an "expert" in order to win one of the constituent games.

In general, when extensive form games are played in parallel, with one player participating in several games simultaneously, such an information transfer from one game to the other is possible. In general, since strategies are structured in extensive form games, they can make use of such information in a non-trivial manner.

In the context of agent-based systems, agents are supposed to play several interactive roles at the same time. Hence when interaction is modelled by games (as in the case of negotiations, auctions, social dilemma games, market games, etc.) such parallel games can assume a great deal of importance. Indeed, a prominent feature of an agent in such a system is the ability to *learn* and transferring strategic moves from one game to the other can be of importance as one form of learning.

Indeed, sequential composition of games can already lead to interesting situations. Consider player A playing a game against B, and after the game is over, playing another instance of the *same* game against player C. Now each of the leaf nodes of the first game carries important historical information about play in the game, and A can strategize differently from each of these nodes in the second game, thus reflecting learning again. Negotiation games carry many such instances of history-based strategizing.

What is needed is an algebra of game composition in which the addition of a parallel operator can be studied in terms of how it interacts with the other operators like choice and sequential composition. This is reminiscent of process calculi, where equivalence of terms in such algebras is studied in depth.

In this paper, we follow the seminal work of Parikh ([12]) on **propositional game logic**. We use dynamic logic for game expressions but extended with parallel composition; since we wish to take into account game structure, we work with extensive form games embedded in Kripke structures rather than with effectivity functions. In this framework, we present a complete axiomatization of the logic and show that the satisfiability problem for the logic is decidable.

The interleaving operator has been looked at in the context of program analysis in terms of dynamic logic [1]. The main technical difficulty addressed in the paper is that parallel composition is not that of sequences (as typically done in process calculi) but that of trees. The main modality of the logic is an assertion of the form $\langle g, i \rangle \alpha$ which asserts, at a state s, that a tree t in the "tree language" associated with g is enabled at s, and that player i has a strategy (subtree) in it to ensure α. Parallel composition is not compositional in the standard logical sense: the semantics of $g_1 \| g_2$ is not given in terms of the semantics of g_1 and g_2 considered as wholes, but by going into their structure. Therefore, defining the enabled-ness of a strategy as above is complicated. Note that the branching structure we consider is quite different from the intersection operator in dynamic logic [8,6,11] and is closer to the paradigm of concurrent dynamic logic [14].

For ease of presentation, we first present the logic with only sequential and parallel composition and discuss technicalities before considering iteration, which adds a great deal of complication. Note that the dual operator, which is important in Parikh's game logic is not relevant here, since we wish to consider games between several players played in parallel.

Related Work

Games have been extensively studied in temporal and dynamic logics. For concurrent games, this effort was pioneered by work on Alternating time temporal logic (ATL) [3], which considers selective quantification over paths. Various extension of ATL was subsequently proposed, these include ones in which strategies can be named and explicitly referred to in the formulas of the logic [18,2,19]. Parikh's work on propositional game logics [12] initiated the study of game structures in terms of algebraic properties. Pauly [13] has built on this to reason about abilities of coalitions of players. Goranko draws parallels between Pauly's coalition logic and ATL [7]. Van Benthem uses dynamic logic to describe games and strategies [16]. Strategic reasoning in terms of a detailed notion of agency has been studied in the *stit* framework [10,4,5].

Somewhat closer in spirit is the work of [17] where van Benthem and co-authors develop a logic to reason about simultaneous games in terms of a parallel operator. The reasoning is based on powers of players in terms of the outcome states that can be ensured. Our point of departure is in considering extensive form game trees explicitly and looking at interleavings of moves of players in the tree structure.

2 Preliminaries

2.1 Extensive Form Games

Let $N = \{1, \ldots, n\}$ denote the set of players, we use i to range over this set. For $i \in N$, we often use the notation $\bar{\imath}$ to denote the set $N \setminus \{i\}$. Let Σ be a finite set of action symbols representing moves of players, we let a, b range over Σ. For a set X and a finite sequence $\rho = x_1 x_2 \ldots x_m \in X^*$, let $\mathit{last}(\rho) = x_m$ denote the last element in this sequence.

Game trees: Let $\mathbb{T} = (S, \Rightarrow, s_0)$ be a tree rooted at s_0 on the set of vertices S and $\Rightarrow : (S \times \Sigma) \to S$ is a *partial* function specifying the edges of the tree. The tree $\mathbb{T}$ is said to be finite if S is a finite set. For a node $s \in S$, let $\overrightarrow{s} = \{s' \in S \mid s \overset{a}{\Rightarrow} s' \text{ for some } a \in \Sigma\}$, $moves(s) = \{a \in \Sigma \mid \exists s' \in S \text{ with } s \overset{a}{\Rightarrow} s'\}$ and $E_T(s) = \{(s, a, s') \mid s \overset{a}{\Rightarrow} s'\}$. By $E_T(s) \times x$ we denote the set $\{((s, x), a, (s', x)) \mid (s, a, s') \in E_T(s)\}$. The set $x \times E_T(s)$ is defined similarly. A node s is called a leaf node (or terminal node) if $\overrightarrow{s} = \emptyset$. The depth of a tree is the length of the longest path in the tree.

An extensive form game tree is a pair $T = (\mathbb{T}, \widehat{\lambda})$ where $\mathbb{T} = (S, \Rightarrow, s_0)$ is a tree. The set S denotes the set of game positions with s_0 being the initial game position. The edge function $\Rightarrow$ specifies the moves enabled at a game position and the turn function $\widehat{\lambda} : S \to N$ associates each game position with a player. Technically, we need player labelling only at the non-leaf nodes. However, for the sake of uniform presentation, we do not distinguish between leaf nodes and non-leaf nodes as far as player labelling is concerned. An extensive form game tree $T = (\mathbb{T}, \widehat{\lambda})$ is said to be finite if $\mathbb{T}$ is finite. For $i \in N$, let $S^i = \{s \mid \widehat{\lambda}(s) = i\}$ and let $frontier(T)$ denote the set of all leaf nodes of T. Let $S^L_T = frontier(T)$ and $S^{NL}_T = S \setminus S^L_T$. For a tree $T = (S, \Rightarrow, s_0, \widehat{\lambda})$ we use $head(T)$ denote the depth one tree generated by taking all the outgoing edges of s_0.

A play in the game T starts by placing a token on s_0 and proceeds as follows: at any stage if the token is at a position s and $\widehat{\lambda}(s) = i$ then player i picks an action which is enabled for her at s, and the token is moved to s' where $s \overset{a}{\Rightarrow} s'$. Formally a play in T is simply a path $\rho : s_0 a_1 s_1 \cdots$ in $\mathbb{T}$ such that for all $j > 0$, $s_{j-1} \overset{a_j}{\Rightarrow} s_j$. Let $Plays(T)$ denote the set of all plays in the game tree T.

2.2 Strategies

A strategy for player $i \in N$ is a function μ^i which specifies a move at every game position of the player, i.e. $\mu^i : S^i \to \Sigma$. A strategy μ^i can also be viewed as a subtree of T where for each player i node, there is a unique outgoing edge and for nodes belonging to players in $\bar{\imath}$, every enabled move is included. Formally we define the strategy tree as follows: For $i \in N$ and a player i strategy $\mu^i : S^i \to \Sigma$ the strategy tree $T_{\mu^i} = (S_{\mu^i}, \Rightarrow_{\mu^i}, s_0, \widehat{\lambda}_{\mu^i})$ associated with μ is the least subtree of T satisfying the following property: $s_0 \in S_{\mu^i}$,

– For any node $s \in S_{\mu^i}$,
  - if $\widehat{\lambda}(s) = i$ then there exists a unique $s' \in S_{\mu^i}$ and action a such that $s \overset{a}{\Rightarrow}_{\mu^i} s'$.
  - if $\widehat{\lambda}(s) \neq i$ then for all s' such that $s \overset{a}{\Rightarrow} s'$, we have $s \overset{a}{\Rightarrow}_{\mu^i} s'$.

Let $\Omega^i(T)$ denote the set of all strategies for player i in the extensive form game tree T. A play $\rho : s_0 a_0 s_1 \cdots$ is said to be consistent with μ^i if for all $j \geq 0$ we have $s_j \in S^i$ implies $\mu^i(s_j) = a_j$.

2.3 Composing Game Trees

We consider sequential and parallel composition of game trees. In the case of sequences, composing them amounts to concatenation and interleaving. Concatenating trees is less straightforward, since each leaf node of the first is now a root of the second tree. Interleaving trees is not the same as a tree obtained by interleaving paths from the two trees, since we wish to preserve choices made by players.

Sequential composition: Suppose we are given two finite extensive form game trees $T_1 = (S_1, \Rightarrow_1, s_1^0, \widehat{\lambda}_1)$ and $T_2 = (S_2, \Rightarrow_2, s_2^0, \widehat{\lambda}_2)$. The sequential composition of T_1 and T_2 (denoted $T_1; T_2$) gives rise to a game tree $T = (S, \Rightarrow, s_0, \widehat{\lambda})$, defined as follows: $S = S_1^{NL} \cup S_2$, $s_0 = s_1^0$,

- $\widehat{\lambda}(s) = \widehat{\lambda}_1(s)$ if $s \in S_1^{NL}$ and $\widehat{\lambda}(s) = \widehat{\lambda}_2(s)$ if $s \in S_2$.
- $s \overset{a}{\Rightarrow} s'$ iff:
  - $s, s' \in S_1^{NL}$ and $s \overset{a}{\Rightarrow}_1 s'$, or
  - $s, s' \in S_2$ and $s \overset{a}{\Rightarrow}_2 s'$, or
  - $s \in S_1^{NL}, s' = s_2^0$ and there exists $s'' \in S_1^L$ such that $s \overset{a}{\Rightarrow}_1 s''$.

In other words, the game tree $T_1; T_2$ is generated by pasting the tree T_2 at all the leaf nodes of T_1. The definition of sequential composition can be extended to a set of trees $\mathcal{T}_2$ (denoted $T_1; \mathcal{T}_2$) with the interpretation that at each leaf node of T_1, a tree $T_2 \in \mathcal{T}_2$ is attached.

Parallel composition: The parallel composition of T_1 and T_2 (denoted $T_1 \| T_2$) yields a set of trees. A tree $t = (S, \Rightarrow, s_0, \widehat{\lambda})$ in the set of trees $T_1 \| T_2$ provided: $S \subseteq S_1 \times S_2$, $s_0 = (s_1^0, s_2^0)$,

- For all $(s, s') \in S$:
  - $E_T((s, s')) = E_{t_1}(s) \times s'$ and $\widehat{\lambda}(s, s') = \widehat{\lambda}_1(s)$, or
  - $E_T((s, s')) = s \times E_{t_2}(s')$ and $\widehat{\lambda}(s, s') = \widehat{\lambda}_2(s')$.
- For every edge $s_1 \overset{a}{\Rightarrow}_1 s_1'$ in t_1, there exists $s_2 \in S_2$ such that $(s_1, s_2) \overset{a}{\Rightarrow} (s_1', s_2)$ in t.
- For every edge $s_2 \overset{a}{\Rightarrow}_2 s_2'$ in t_2, there exists $s_1 \in S_1$ such that $(s_1, s_2) \overset{a}{\Rightarrow} (s_1, s_2')$ in t.

3 Examples

Consider the trees T_1 and T_2 given in Figure 1. The sequential composition of T_1 and T_2 (denoted $T_1; T_2$) is shown in Figure 2. This is obtained by pasting the tree T_2 at all the leaf nodes of T_1.

Now consider two finite extensive form game trees T_4 and T_5 given in figure 3. Each game is played between two players, player 2 is common in both games.

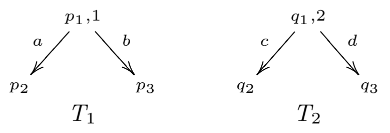


Fig. 1. Atomic games

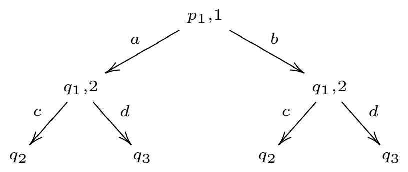


Fig. 2. $T_1; T_2$

Note that we are talking about different instances of the same game (as evident from the similar game trees) played between different pairs of players with a player in common. Consider the interleaving of T_4 and T_5 where player 1 moves first in T_4, followed by 2 and 3 in T_5, and then again coming back to the game T_4, with the player 2-moves. This game constitutes a valid tree in the set of trees defined by $T_4 \| T_5$ and is shown in Figure 4.

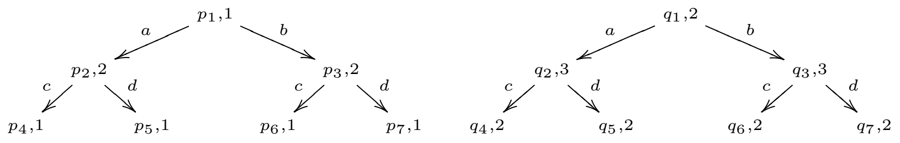
Due to space constraints, we have not provided the names for each of the states in the parallel game tree, but they are quite clear from the context. The game starts with player 1 moving from p_1 in T_4 to p_2 or p_3. Then the play moves to the game T_5, where player 2 moves to q_2 or q_3, followed by the moves of player 3. After that, the play comes back to T_4, where player 2 moves once again.

These games clearly represent toy versions of "playing against two Grandmasters simultaneously". Players 1 and 3 can be considered as the Grandmasters, and 2 as the poor mortal. Let us now describe the copycat strategy that can be used by player 2, when the two games are played in parallel. The simultaneous game (figure 4), starts with player 1 making the first move a, say in the game tree T_4 (from (p_1, q_1)) to move to (p_2, q_1). Player 2 then copies this move in game T_5, to move to (p_2, q_2). The game continues in T_5, with player 3 moving to (p_2, q_4), say. Player 2 then copies this move in T_4 (playing action c) to move to (p_4, q_4). This constitutes a play of the game, where player 2 copies the moves of players 1 and 3, respectively.

Evidently, if player 1 has a strategy in T_4 to achieve a certain objective, whatever be the moves of player 2, following the same strategy, player 2 can attain the same objective in T_5.

Parallel composition can also be performed with respect to games structures which are not the same. Consider the game trees T_6 and T_7 given in Figure 5.

An interleaved game where each game is played alternatively starting from the game T_6 can be represented by the game tree in Figure 6.



(a) T_4 (b) T_5

Fig. 3. Atomic games

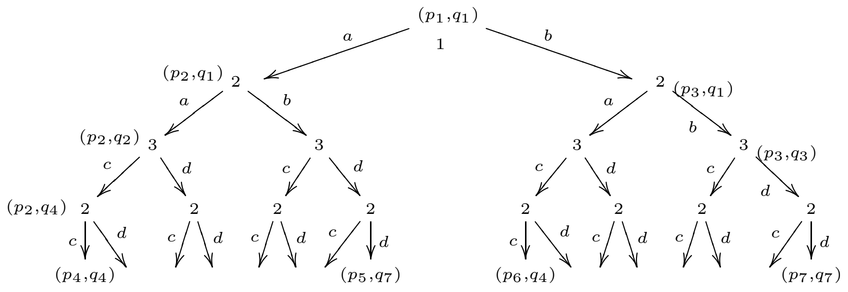


Fig. 4. Game tree T

4 The Logic

For a finite set of action symbols Σ, let $\mathcal{T}(\Sigma)$ be a countable set of finite extensive form game trees over the action set Σ which is closed under subtree inclusion. That is, if $T \in \mathcal{T}(\Sigma)$ and T' is a subtree of T then $T' \in \mathcal{T}(\Sigma)$. We also assume that for each $a \in \Sigma$, the tree consisting of the single edge labelled with a is in $\mathcal{T}(\Sigma)$. Let $\mathbb{H}$ be a countable set and h, h' range over this set. Elements of $\mathbb{H}$ are referred to in the formulas of the logic and the idea is to use them as names for extensive form game trees in $\mathcal{T}(\Sigma)$. Formally we have a map $\nu : \mathbb{H} \to \mathcal{T}(\Sigma)$ which given any name $h \in \mathbb{H}$ associates a tree $\nu(h) \in \mathcal{T}(\Sigma)$. We often abuse notation and use h to also denote $\nu(h)$ where the meaning is clear from the context.

4.1 Syntax

Let P be a countable set of propositions, the syntax of the logic is given by:

$$\Gamma := h \mid g_1; g_2 \mid g_1 \cup g_2 \mid g_1 \| g_2$$

$$\Phi := p \in P \mid \neg\alpha \mid \alpha_1 \vee \alpha_2 \mid \langle g, i \rangle \alpha$$

where $h \in \mathbb{H}$ and $g \in \Gamma$.

In Γ, the atomic construct h specifies a finite extensive form game tree. Composite games are then constructed using the standard dynamic logic operators along with the parallel operator. $g_1 \cup g_2$ denotes playing g_1 or g_2. Sequential composition is denoted by $g_1; g_2$ and $g_1 \| g_2$ denotes the parallel composition of games.

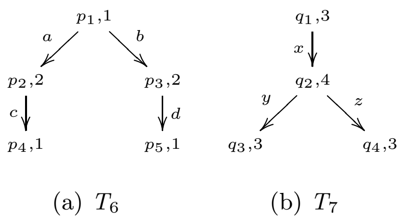


Fig. 5. Atomic games

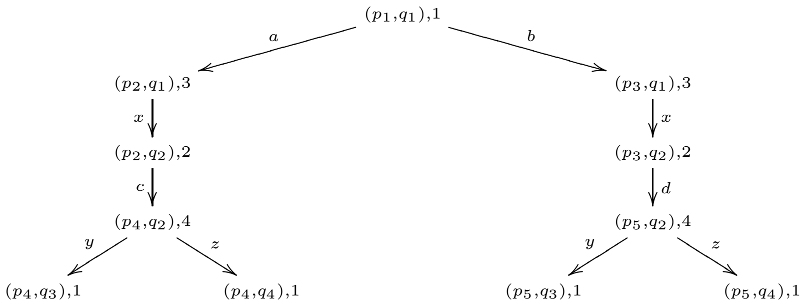


Fig. 6. A game tree in $T_6 \| T_7$

The main connective $\langle g, i \rangle \alpha$ asserts at state s that a tree in g is enabled at s and that player i has a strategy subtree in it at whose leaves α holds.

4.2 Semantics

A model $M = (W, \rightarrow, \widehat{\lambda}, V)$ where W is the set of states (or game positions), $\rightarrow \subseteq W \times \Sigma \times W$ is the move relation, $V : W \rightarrow 2^P$ is a valuation function and $\widehat{\lambda} : W \rightarrow N$ is a player labelling function. These can be thought of as standard Kripke structures whose states correspond to game positions along with an additional player labelling function. An extensive form game tree can be thought of as *enabled* at a certain state, say s of a Kripke structure, if we can embed the tree structure in the tree unfolding of the Kripke structure rooted at s. We make this notion more precise below.

Enabling of trees: For a game position $u \in W$, let T_u denote the tree unfolding of M rooted at u. We say the game h is enabled at a state u if the structure $\nu(h)$ can be embedded in T_u with respect to the enabled actions and player labelling. Formally this can be defined as follows:

Given a state u and $h \in \mathbb{H}$, let $T_u = (S^s_M, \Rightarrow_M, \widehat{\lambda}_M, s)$ and $\nu(h) = T_h = (S_h, \Rightarrow_h, \widehat{\lambda}_h, s_{h,0})$. The restriction of T_u with respect to the game tree h (denoted $T_u \upharpoonright h$) is the subtree of T_s which is generated by the structure specified by T_h.

The restriction is defined inductively as follows: $T_u \upharpoonright h = (S, \Rightarrow, \widehat{\lambda}, s_0, f)$ where $f : S \to S_h$. Initially $S = \{s\}$, $\widehat{\lambda}(s) = \widehat{\lambda}_M(s)$, $s_0 = s$ and $f(s_0) = s_{h,0}$.

For any $s \in S$, let $f(s) = t \in S_h$. Let $\{a_1, \dots, a_k\}$ be the outgoing edges of t, i.e. for all $j : 1 \leq j \leq k$, $t \overset{a_j}{\Rightarrow}_h t_j$. For each a_j, let $\{s_j^1, \dots, s_j^m\}$ be the nodes in S_M^s such that $s \overset{a_j}{\Rightarrow}_M s_j^l$ for all $l : 1 \leq l \leq m$. Add nodes $s_j^1, \dots, s_j^m$ to S and the edges $s \overset{a_j}{\Rightarrow} s_j^l$ for all $l : 1 \leq l \leq m$. Also set $\widehat{\lambda}(s_j^l) = \widehat{\lambda}_M(s_j^l)$ and $f(s_j^l) = t_j$.

We say that a game h is enabled at u (denoted *enabled*(h, u)) if the tree $T_u \upharpoonright h = (S, \Rightarrow, \widehat{\lambda}, s_0, f)$ satisfies the following properties: for all $s \in S$,

– $moves(s) = moves(f(s))$,
– if $moves(s) \neq \emptyset$ then $\widehat{\lambda}(s) = \widehat{\lambda}_h(f(s))$.

Interpretation of atomic games: To formally define the semantics of the logic, we need to first fix the interpretation of the compositional games constructs. In the dynamic logic approach, for each game construct g and player i we would associate a relation $R_g^i \subseteq (W \times 2^W)$ which specifies the outcome of a winning strategy for player i. However due to the ability of being able to interleave game positions, in this setting we need to keep track of the actual tree structure rather just the "input-output" relations, which is closer in spirit to what is done in process logics [9] . Thus for a game g and player i we define the relation $R_g^i \subseteq 2^{(W \times W)^*}$. For a pair $\mathbf{x} = (u, w) \in W \times W$ and a set of sequences $Y \in 2^{(W \times W)^*}$ we define $(u, w) \cdot Y = \{(u, w) \cdot \rho \mid \rho \in Y\}$. For $j \in \{1, 2\}$ we use $\mathbf{x}[j]$ to denote the j-th component of $\mathbf{x}$.

For each atomic game h and each state $u \in W$, we define $R_h^i(u)$ in a bottom-up manner in such a way that whenever h is enabled at u, $R_h^i(u)$ encodes the set of all available strategies (cf. Section 2.2) for player i in the game h enabled at u. The collection of all such strategies that a player i can have, whenever the game h is enabled at some state $u \in W$ is given by R_h^i.

Let $h = (S, \Rightarrow, s_0, \widehat{\lambda})$ be a depth 1 tree with $moves(s_0) = \{a_1, \dots, a_k\}$ and for all $s \neq s_0$, $moves(s) = \emptyset$. For $i \in N$ and a state $u \in W$, we define $R_h^i(u) \subseteq 2^{(W \times W)^*}$ as follows:

– If $\widehat{\lambda}(s_0) = i$ then $R_h^i(u) = \{X_j \mid enabled(h, u) \text{ and } X_j = \{(u, w_j)\}$ where $u \xrightarrow{a_j} w_j\}$.
– if $\widehat{\lambda}(s_0) \in \overline{\imath}$ then $R_h^i(u) = \{\{(u, w_j) \mid enabled(h, u) \text{ and } \exists a_j \in moves(s_0)$ with $u \xrightarrow{a_j} w_j\}\}$.

For $g \in \Gamma$, let $R_g^i = \bigcup_{u \in W} R_g^i(u)$.

For a tree $h = (S, \Rightarrow, s_0, \widehat{\lambda})$ such that $depth(h) > 1$, we define $R_h^i(u)$ as,

– if $\widehat{\lambda}(s_0) = i$ then $R_h^i(u) = \{\{(u, w) \cdot Y\} \mid \exists X \in R_{head(h)}^i$ with $(u, w) \in X, u \xrightarrow{a_j} w$ and $Y \in R_{h_{a_j}}^i\}$
– if $\widehat{\lambda}(s_0) \in \overline{\imath}$ then $R_h^i(u) = \{\{(u, w) \cdot Y \mid \exists X \in R_{head(h)}^i$ with $(u, w) \in X, u \xrightarrow{a_j} w$ and $Y \in R_{h_{a_j}}^i\}\}$.

Remark: Note that a set $X \in R_h^i$ can contain sequences such as $(u,w)(v,x)$ where $w \neq v$. Thus in general sequence of pairs of states in X need not represent a subtree of T_u for some $u \in W$. We however need to include such sequences since if h is interleaved with another game tree h', a move enabled in h' could make the transition from w to v. A sequence $\varrho \in X$ is said to be legal if whenever $(u,w)(v,x)$ is a subsequence of ϱ then $w = v$. A set $X \subseteq 2^{(W\times W)^*}$ is a valid tree if for all sequence $\varrho \in X$, ϱ is legal and X is prefix closed. For X which is a valid tree we have the property that for all $\varrho, \varrho' \in X$, $\mathit{first}(\varrho)[1] = \mathit{first}(\varrho')[1]$. We denote this state by $\mathit{root}(X)$. We also use $\mathit{frontier}(X)$ to denote the frontier nodes, i.e. $\mathit{frontier}(X) = \{\mathit{last}(\varrho)[2] \mid \varrho \in X\}$.

For a game tree h, although every set $X \in R_h^i$ need not be a valid tree, we can associate a tree structure with X (denoted $\mathfrak{T}(X)$) where the edges are labelled with pairs of the form (u,w) which appears in X. Conversely given $W \times W$ edge labelled finite game tree $\mathfrak{T}$, we can construct a set $X \subseteq 2^{(W\times W)^*}$ by simply enumerating the paths and extracting the labels of each edge in the path. We denote this translation by $\mathfrak{f}(\mathfrak{T})$. We use these two translations in what follows:

Interpretation of composite games: For $g \in \Gamma$ and $i \in N$, we define $R_g^i \subseteq 2^{(W\times W)^*}$ as follows:

- $R^i_{g_1 \cup g_2} = R^i_{g_1} \cup R^i_{g_2}$.
- $R^i_{g_1;g_2} = \{\mathfrak{f}(\mathfrak{T}(X);\mathcal{T}) \mid X \in R^i_{g_1}$ and $\mathcal{T} = \{\mathfrak{T}(X_1),\ldots,\mathfrak{T}(X_k)\}$ where $\{X_1,\ldots,X_k\} \subseteq R^i_{g_2}\}$.
- $R^i_{g_1\|g_2} = \{\mathfrak{f}(\mathfrak{T}(X_1)\|\mathfrak{T}(X_2)) \mid X_1 \in R^i_{g_1}$ and $X_2 \in R^i_{g_2}\}$.

The truth of a formula $\alpha \in \Phi$ in a model M and a position u (denoted $M,u \models \alpha$) is defined as follows:

- $M,u \models p$ iff $p \in V(u)$.
- $M,u \models \neg\alpha$ iff $M,u \not\models \alpha$.
- $M,u \models \alpha_1 \vee \alpha_2$ iff $M,u \models \alpha_1$ or $M,u \models \alpha_2$.
- $M,u \models \langle g,i\rangle\alpha$ iff $\exists X \in R^i_g$ such that X constitutes a valid tree, $\mathit{root}(X) = u$ and for all $w \in \mathit{frontier}(X)$, $M,w \models \alpha$.

A formula α is satisfiable if there exists a model M and a state u such that $M,u \models \alpha$.

Let h_1 and h_2 be the game trees T_4 and T_5 given in Figure 3. The tree in which the moves of players are interleaved in lock-step synchrony is one of the trees in the semantics of $h_1\|h_2$. This essentially means that at every other stage if a depth one tree is enabled then after that the same tree structure is enabled again, except for the player labelling. Given the (finite) atomic trees, we can write a formula α_{LS} which specifies this condition. If the tree h is a minimal one, i.e. of depth one given by $(S, \Rightarrow, s_0, \widehat{\lambda})$, α_{LS_h} can be defined as, $\bigwedge_{a_j \in moves(s_0)}(\langle a_j\rangle\top \wedge [a_j](\wedge_{a_j \in moves(s_0)}\langle a_j\rangle\top)$.

If player 1 has a strategy (playing a, say) to achieve certain objective ϕ in the game h_1, player 2 can play (copy) the same strategy in h_2 to ensure ϕ. This phenomenon can be adequately captured in the interleaved game structure, where player 2 has a strategy (viz. playing a) to end in those states of the game $h_1 \| h_2$, where player 1 can end in h_1. So we have that, whenever h_1 and $h_1 \| h_2$ are enabled and players can move in lock-step synchrony with respect to the game h_1 (or, h_2), $\langle h_1, 1 \rangle \phi \rightarrow \langle h_1 \| h_2, 2 \rangle \phi$ holds.

5 Axiom System

The main technical contribution of this paper is a sound and complete axiom system. Firstly, note that the logic extends standard PDL. For $a \in \Sigma$ and $i \in N$, let T^i_a be the tree defined as: $T^i_a = (S, \Rightarrow, s_0, \widehat{\lambda})$ where $S = \{s_0, s_1\}$, $s_0 \overset{a}{\Rightarrow} s_1$, $\widehat{\lambda}(s_0) = i$ and $\widehat{\lambda}(s_1) \in N$. Let t^i_a be the name denoting this tree, i.e. $\nu(t^i_a) = T^i_a$. For each $a \in \Sigma$ we define,

– $\langle a \rangle \alpha = \bigwedge_{i \in N} (\mathbf{turn}_i \supset \langle t^i_a, i \rangle \alpha)$.

From the semantics it is easy to see that we get the standard interpretation for $\langle a \rangle \alpha$, i.e. $\langle a \rangle \alpha$ holds at a state u iff there is a state w such that $u \overset{a}{\rightarrow} w$ and α holds at w.

Enabling of trees: The crucial observation is that the property of whether a game is enabled can be described by a formula of the logic. Formally, for $h \in \mathbb{H}$ such that $\nu(h) = (S, \Rightarrow, s_0, \widehat{\lambda})$ and $moves(s_0) \neq \emptyset$ and an action $a \in moves(s_0)$, let h_a be the subtree of T rooted at a node s' with $s_0 \overset{a}{\Rightarrow} s'$. The formula $h^{\checkmark}$ (defined below) is used to express the fact that the tree structure $\nu(h)$ is enabled and $head^{\checkmark}_h$ to express that $head(\nu(h))$ is enabled. This is defined as,

– If $\nu(h)$ is atomic then $h^{\checkmark} = \top$ and $head^{\checkmark}_h = \top$.
– If $\nu(h)$ is not atomic and $\widehat{\lambda}(s_0) = i$ then
  • $h^{\checkmark} = \mathbf{turn}_i \wedge (\bigwedge_{a_j \in moves(s_0)} (\langle a_j \rangle \top \wedge [a_j] h^{\checkmark}_{a_j}))$.
  • $head^{\checkmark}_h = \mathbf{turn}_i \wedge (\bigwedge_{a_j \in moves(s_0)} \langle a_j \rangle \top)$.

Due to the ability to interleave choices of players, we also need to define for a composite game expression g, the initial (atomic) game of g and the game expression generated after playing the initial atomic game (or in other words the residue). We make this notion precise below:

Definition of init

– $init(h) = \{h\}$ for $h \in \mathrm{G}$
– $init(g_1; g_2) = init(g_1)$ if $g_1 \neq \epsilon$ else $init(g_2)$.
– $init(g_1 \cup g_2) = init(g_1) \cup init(g_2)$.
– $init(g_1 \| g_2) = init(g_1) \cup init(g_2)$.

Definition of residue

- $h\backslash h = \epsilon$ and $\epsilon\backslash h = \epsilon$.
- $(g_1; g_2)\backslash h = \begin{cases} (g_1\backslash h); g_2 & \text{if } g_1 \neq \epsilon. \\ (g_2\backslash h) & \text{otherwise.} \end{cases}$
- $(g_1 \cup g_2)\backslash h = \begin{cases} (g_1\backslash h) \cup (g_2\backslash h) & \text{if } h \in \mathit{init}(g_1) \text{ and } h \in \mathit{init}(g_2). \\ g_1\backslash h & \text{if } h \in \mathit{init}(g_1) \text{ and } h \notin \mathit{init}(g_2). \\ g_2\backslash h & \text{if } h \in \mathit{init}(g_2) \text{ and } h \notin \mathit{init}(g_1). \end{cases}$
- $(g_1 \| g_2)\backslash h = \begin{cases} (g_1\backslash h \| g_2) \cup (g_1 \| g_2\backslash h) & \text{if } h \in \mathit{init}(g_1) \text{ and } h \in \mathit{init}(g_2). \\ (g_1\backslash h \| g_2) & \text{if } h \in \mathit{init}(g_1) \text{ and } h \notin \mathit{init}(g_2). \\ (g_1 \| g_2\backslash h) & \text{if } h \in \mathit{init}(g_2) \text{ and } h \notin \mathit{init}(g_1). \end{cases}$

The translation used to express the property of enabling of trees in terms of standard PDL formulas also suggest that the techniques developed for proving completeness of PDL can be applied in the current setting. We base our axiomatization of the logic on the "reduction axioms" methodology of dynamic logic. The most interesting reduction axiom in our setting would naturally involve the parallel composition operator. Intuitively, for game expressions g_1, g_2, a formula α and a player $i \in N$ the reduction axiom for $\langle g_1 \| g_2, i\rangle\alpha$ need to express the following properties:

- There exists an atomic tree $h \in \mathit{init}(g_1 \| g_2)$ such that $\mathit{head}(\nu(h))$ is enabled.
- Player i has a strategy in $\mathit{head}(\nu(h))$ which when composed with a strategy in the residue ensures α. We use $\mathit{comp}^i(h, g_1, g_2, \alpha)$ to denote this property and formally define it inductively as follows:

Suppose $h = (S, \Rightarrow, s_0, \widehat{\lambda})$ where $A = \mathit{moves}(s_0) = \{a_1, \ldots, a_k\}$.

- If $h \in \mathit{init}(g_1)$, $h \in \mathit{init}(g_2)$ and
  - $\widehat{\lambda}(s_0) = i$ then $\mathit{comp}^i(h, g_1, g_2, \alpha) = \bigvee_{a_j \in A}(\langle a_j\rangle\langle(h_{a_j}; (g_1\backslash h))\|g_2\rangle\alpha \vee \langle a_j\rangle\langle g_1\|(h_{a_j}; (g_2\backslash h))\rangle\alpha)$.
  - $\widehat{\lambda}(s_0) \in \bar{\imath}$ then $\mathit{comp}^i(h, g_1, g_2, \alpha) = \bigwedge_{a_j \in A}([a_j]\langle(h_{a_j}; (g_1\backslash h))\|g_2\rangle\alpha \vee [a_j]\langle g_1\|(h_{a_j}; (g_2\backslash h))\rangle\alpha)$.
- If $h \in \mathit{init}(g_1)$, $h \notin \mathit{init}(g_2)$ and
  - $\widehat{\lambda}(s_0) = i$ then $\mathit{comp}^i(h, g_1, g_2, \alpha) = \bigvee_{a_j \in A}(\langle a_j\rangle\langle(h_{a_j}; (g_1\backslash h))\|g_2\rangle\alpha)$.
  - $\widehat{\lambda}(s_0) \in \bar{\imath}$ then $\mathit{comp}^i(h, g_1, g_2, \alpha) = \bigwedge_{a_j \in A}([a_j]\langle(h_{a_j}; (g_1\backslash h))\|g_2\rangle\alpha)$.
- if $h \in \mathit{init}(g_2)$, $h \notin \mathit{init}(g_1)$ and
  - $\widehat{\lambda}(s_0) = i$ then $\mathit{comp}^i(h, g_1, g_2, \alpha) = \bigvee_{a_j \in A}(\langle a_j\rangle\langle g_1\|(h_{a_j}; (g_2\backslash h))\rangle\alpha)$.
  - $\widehat{\lambda}(s_0) \in \bar{\imath}$ then $\mathit{comp}^i(h, g_1, g_2, \alpha) = \bigwedge_{a_j \in A}([a_j]\langle g_1\|(h_{a_j}; (g_2\backslash h))\rangle\alpha)$.

Note that the semantics for parallel composition allows us to interleave subtrees of g_2 within g_1 (and vice versa). Therefore in the definition of comp^i at each stage after an action a_j, it is important to perform the sequential composition of the subtree h_{a_j} with the residue of the game expression.

The axiom schemes

(A1) Propositional axioms:
  (a) All the substitutional instances of tautologies of PC.
  (b) $\mathbf{turn}_i \equiv \bigwedge_{j \in \bar{\imath}} \neg \mathbf{turn}_j$.

(A2) Axiom for single edge games:
  (a) $\langle a \rangle(\alpha_1 \vee \alpha_2) \equiv \langle a \rangle \alpha_1 \vee \langle a \rangle \alpha_2$.
  (b) $\langle a \rangle \mathbf{turn}_i \supset [a]\mathbf{turn}_i$.

(A3) Dynamic logic axioms:
  (a) $\langle g_1 \cup g_2, i \rangle \alpha \equiv \langle g_1, i \rangle \alpha \vee \langle g_2, i \rangle \alpha$.
  (b) $\langle g_1 ; g_2, i \rangle \alpha \equiv \langle g_1, i \rangle \langle g_2, i \rangle \alpha$.
  (c) $\langle g_1 \| g_2, i \rangle \alpha \equiv \bigvee_{h \in init(g_1 \| g_2)} head_h^{\checkmark} \wedge comp^i(h, g_1, g_2, \alpha)$.

(A4) $\langle h, i \rangle \alpha \equiv h^{\checkmark} \wedge \downarrow_{(h,i,\alpha)}$.

For $h \in \mathbb{H}$ with $\nu(h) = T = (S, \Rightarrow, s_0, \widehat{\lambda})$ we define $\downarrow_{(h,i,\alpha)}$ as follow:

$$- \downarrow_{(h,i,\alpha)} = \begin{cases} \alpha & \text{if } moves(s_0) = \emptyset. \\ \bigvee_{a \in \Sigma} \langle a \rangle \langle h_a, i \rangle \alpha & \text{if } moves(s_0) \neq \emptyset \text{ and } \widehat{\lambda}(s_0) = i. \\ \bigwedge_{a \in \Sigma} [a] \langle h_a, i \rangle \alpha & \text{if } moves(s_0) \neq \emptyset \text{ and } \widehat{\lambda}(s_0) \in \bar{\imath}. \end{cases}$$

Inference rules

$$(MP)\ \frac{\alpha, \quad \alpha \supset \beta}{\beta} \qquad (NG)\ \frac{\alpha}{[a]\alpha}$$

Axioms (A1) and (A2) are self explanatory. Axiom (A3) constitutes the reduction axioms for the compositional operators. Note that unlike in PDL sequential composition in our setting corresponds to composition over trees. The following proposition shows that the usual reduction axiom for sequential composition remains valid.

Proposition 5.1. *The formula $\langle g_1 ; g_2, i \rangle \alpha \equiv \langle g_1, i \rangle \langle g_2, i \rangle \alpha$ is valid.*

Proof. Suppose $\langle g_1 ; g_2, i \rangle \alpha \supset \langle g_1, i \rangle \langle g_2, i \rangle \alpha$ is not valid. This means there exists a model M and a state u such that $M, u \models \langle g_1 ; g_2, i \rangle \alpha$ and $M, u \not\models \langle g_1, i \rangle \langle g_2, i \rangle \alpha$. From semantics we get $\exists X \in R^i_{g_1;g_2}$ such that X is a valid tree, $root(X) = u$ and for all $w \in frontier(X)$ we have $M, u \models \alpha$. By definition, X is of the form $\mathfrak{f}(\mathfrak{T}(Y); \mathcal{T})$ where $Y \in R^i_{g_1}$ and $\mathcal{T} = \{\mathfrak{T}(X_1), \ldots, \mathfrak{T}(X_k)\}$ with $\{X_1, \ldots, X_k\} \subseteq R^i_{g_2}\}$. Since X is a valid tree we have $Y, X_1, \ldots, X_k$ are valid trees. Thus we get that for all $j : 1 \leq j \leq k$, $M, root(X_j) \models \langle \xi_2, i \rangle \alpha$ and from semantics we have $M, u \models \langle g_1, i \rangle \langle g_2, i \rangle \alpha$ which gives the required contradiction.

A similar argument which makes use of the definition of R^i_g and the semantics shows that $\langle g_1, i \rangle \langle g_2, i \rangle \alpha \supset \langle g_1 ; g_2, i \rangle \alpha$ is valid.

5.1 Completeness

To show completeness, we prove that every consistent formula is satisfiable. Let α_0 be a consistent formula, and $CL(\alpha_0)$ denote the subformula closure of α_0. In addition to the usual subformula closure we also require the following: if $\langle h, i\rangle\alpha \in CL(\alpha_0)$ then $g^{\checkmark}, \downarrow_{(h,i,\alpha)} \in CL(\alpha_0)$ and if $\langle g_1 \| g_2, i\rangle\alpha \in CL(\alpha_0)$ then $\bigwedge_{h\in init(g_1\|g_2)} head_h^{\checkmark}$, $comp^i(h, g_1, g_2, \alpha) \in CL(\alpha_0)$.

Let $AT(\alpha_0)$ be the set of all maximal consistent subsets of $CL(\alpha_0)$, referred to as atoms. We use u, w to range over the set of atoms. Each $u \in AT(\alpha_0)$ is a finite set of formulas, we denote the conjunction of all formulas in u by $\widehat{u}$. For a nonempty subset $X \subseteq AT(\alpha_0)$, we denote by $\widetilde{X}$ the disjunction of all $\widehat{u}, u \in X$. Define a transition relation on $AT(\alpha_0)$ as follows: $u \xrightarrow{a} w$ iff $\widehat{u} \wedge \langle a\rangle\widehat{w}$ is consistent. Let the model $M = (W, \longrightarrow, V)$ where $W = AT(\alpha_0)$ and the valuation function V is defined as $V(w) = \{p \in P \mid p \in w\}$. Once the model is defined, the semantics (given earlier) specifies relation R^i_g. The following lemma asserts the consistency condition on elements of R^i_g.

Lemma 5.1. *For all $i \in N$, for all $h \in \mathbb{H}$, for all $X \subseteq (W \times W)^*$ with $\mathcal{X} = frontier(X)$, for all $u \in W$ the following holds:*

1. *if X is a valid tree with $root(X) = u$ and $X \in R^i_h$ then $\widehat{u} \wedge \langle h, i\rangle\widetilde{\mathcal{X}}$ is consistent.*
2. *if $\widehat{u} \wedge \langle h, i\rangle\widetilde{\mathcal{X}}$ is consistent then there exists a X' which is a valid tree with $frontier(X') \subseteq \mathcal{X}$ and $root(X') = u$ such that $X' \in R^i_h$.*

Proof. A detailed proof is given in the appendix. It essentially involves showing that the game h is enabled at the state u and that there is a strategy for player i in $T_u \upharpoonright h$ represented by the tree X whose frontier nodes are $\mathcal{X}$. The strategy tree X is constructed in stages starting at u. For any path of the partially constructed strategy tree if the paths ends in a position of player i then the path is extended by guessing a unique outgoing edge. If the position belongs to a player in $\bar{\imath}$ then all edges are taken into account.

Lemma 5.2. *For all $i \in N$, for all $g \in \Gamma$, for all $X \subseteq (W \times W)^*$ with $\mathcal{X} = frontier(X)$ and $u \in W$, if $\widehat{u} \wedge \langle h, i\rangle\widetilde{\mathcal{X}}$ is consistent then there exists X' which is a valid tree with $frontier(X') \subseteq \mathcal{X}$ and $root(X') = u$ such that $X' \in R^i_h$.*

Proof is given in the appendix.

Lemma 5.3. *For all $\langle g, i\rangle\alpha \in CL(\alpha_0)$, for all $u \in W$, $\widehat{u} \wedge \langle g, i\rangle\alpha$ is consistent iff there exists $X \in R^i_g$ which is a valid tree with $root(X) = u$ such that $\forall w \in frontier(X)$, $\alpha \in w$.*

Proof. ($\Rightarrow$) Follows from lemma 5.2.
($\Leftarrow$) Suppose there exists $X \in R^i_g$ which is a valid tree with $root(X) = u$ such that $\forall w \in frontier(X)$, $\alpha \in w$. We need to show that $\widehat{u} \wedge \langle g, i\rangle\alpha$ is consistent, this is done by induction on the structure of g.

– The case when $g = h$ follows from lemma 5.1. For $g = g_1 \cup g_2$ the result follows from axiom (A3a).
– $g = g_1; g_2$: Since $X \in R^i_{g_1;g_2}$, $\exists Y$ with $root(Y) = u$ and $frontier(Y) = \{v_2, \ldots, v_k\}$, there exist sets $X_1, \ldots, X_k$ where for all $j : 1 \leq j \leq k$, $root(X_j) = v_j$, $\bigcup_{j=1,\ldots,k} frontier(X_j) = frontier(X)$, $X_j \in R^i_{g_2}$ and $Y \in R^i_{g_1}$. By induction hypothesis, for all j, $\widehat{v}_j \wedge \langle g_2 \rangle \alpha$ is consistent. Since v_j is an atom and $\langle g_2, i\rangle\alpha \in CL(\alpha_0)$, we get $\langle g_2, i\rangle\alpha \in v_j$. Again by induction hypothesis we have $\widehat{u} \wedge \langle g_1, i\rangle\langle g_2, i\rangle\alpha$ is consistent. Hence from (A3b) we have $\widehat{u} \wedge \langle g_1; g_2, i\rangle\alpha$ is consistent.
– $g = g_1 \| g_2$: Let $h \in init(g_1 \| g_2)$, and $h = (S, \Rightarrow, s_0, \widehat{\lambda})$. We have three cases depending on whether h is the initial constituent game in g_1 and g_2. We look at the case when $h \in init(g_1)$ and $h \notin init(g_2)$, the arguments for the remaining cases are similar. Let $A = moves(s_0) = \{a_1, \ldots, a_k\}$. By semantics, since $enabled(h, u)$ holds we have $moves(u) = A$. We also get there exists $Y_j \in R^i_{t_{a_j};(g_1 \backslash h)\|g_2}$ where $\bigcup_{j=1,\ldots,k} frontier(Y_j) = frontier(X)$. Suppose $\widehat{\lambda}(s_0) = \bar{\imath}$, by performing a second induction on the depth of X we can argue that $\widehat{u} \wedge (\bigwedge_{a_j \in A}([a_j]\langle(t_{a_j}; (g_1 \backslash h))\|g_2\rangle\alpha)$ is consistent. Therefore from axiom (A3c) we have $\widehat{u} \wedge \langle g_1 \| g_2\rangle\alpha$ is consistent.

This leads us to the following theorem from which we can deduce the completeness of the axiom system.

Theorem 5.1. *For all formulas α_0, if α_0 is consistent then α_0 is satisfiable.*

Dedidability: Given a formula α_0, let $\mathfrak{H}(\alpha_0)$ be the set of all atomic game terms appearing in α_0. Let $\mathfrak{T}(\alpha_0) = \{\nu(h) \mid h \in \mathfrak{H}(\alpha_0)\}$ and $\mathfrak{m} = \max_{T \in \mathfrak{T}(\alpha_0)} |T|$. For any finite tree T, we define $|T|$ to be the number of vertices and edges in T. It can be verified that $|CL(\alpha_0)|$ is linear in $|\alpha_0|$ and therefore we have $|AT(\alpha_0)| = \mathcal{O}(2^{|\alpha_0|})$. The states of the model M constitutes atoms of α_0 and therefore we get that if α_0 is satisfiable then there is a model whose size is at most exponential in $|\alpha_0|$. The relation R^i_g can be explicitly constructed in time $\mathcal{O}(2^{|M|^{\mathfrak{m}}})$. Thus we get the following corollary.

Corollary 5.1. *The satisfiability problem for the logic is decidable.*

6 Discussion

Iteration

An obvious extension of the logic is to add an operator for (unbounded) iteration of sequential composition. The semantics is slightly more complicated since we are dealing with trees. One needs to define it in terms of a least fixed point operator (as seen in [12]). Under this interpretation, the standard dynamic logic axiom for iteration remains valid: $\langle g^*, i\rangle\alpha \equiv \alpha \vee \langle g, i\rangle\langle g^*, i\rangle\alpha$.

We also have the familiar induction rule for dynamic logic which asserts that when α is invariant under g so it is with the iteration of g.

$$(IND)\ \frac{\langle g, i\rangle\alpha \supset \alpha}{\langle g^*, i\rangle\alpha \supset \alpha}$$

Note that the completeness proof (in the presence of interleaving) gets considerably more complicated now. Firstly, the complexity of $g\backslash h$ is no longer less than that of g so we cannot apply induction directly for parallel composition. In general when we consider $g_1^*\|g_2^*$, the interleaving critically depends on how many iterations are chosen in each of the components. The technique is to consider a graph for every g as follows: add an edge labelled h from g to $g\backslash h$. This is a finite graph, and we can show that the enabling of g at a state s corresponds to the existence of an embedding of this graph at s. In effect, the unfolding of the parallel composition axiom asserts the existence of this subgraph, and the rest of the proof uses the induction rule as in the completeness proof for dynamic logic. We omit the detailed proof here since it is technical and lengthy.

Strategy Specifications

Throughout the paper we have been talking of existence of strategies in compositional games. It would be more interesting to specify strategies explicitly in terms of their properties as done in [15]. In the presence of parallel composition, this adds more value to the analysis since apart from specifying structural conditions which ensures the ability for players to copy moves, we can also specify the exact sequence of moves which are copied across games. The basic techniques used here can be extended to deal with strategy specification. However, it would be more interesting to come up with compositional operators for strategy specifications which can naturally exploit the interleaving semantics.

Acknowledgements. We thank the anonymous referees for their valuable comments and suggestions. The second author thanks the Netherlands Institute for Advanced Study in the Humanities and Social Sciences for its support.

Appendix

Lemma 5.1. For all $i \in N$, for all $h \in \mathbb{H}$, for all $X \subseteq (W \times W)^*$ with $\mathcal{X} = \mathit{frontier}(X)$, for all $u \in W$ the following holds:

1. if X is a valid tree with $\mathit{root}(X) = u$ and $X \in R_h^i$ then $\widehat{u} \wedge \langle h, i\rangle\widetilde{\mathcal{X}}$ is consistent.
2. if $\widehat{u} \wedge \langle h, i\rangle\widetilde{\mathcal{X}}$ is consistent then there exists a X' which is a valid tree with $\mathit{frontier}(X') \subseteq \mathcal{X}$ and $\mathit{root}(X') = u$ such that $X' \in R_h^i$.

Proof. Let $h = (S, \Rightarrow, s_0, \widehat{\lambda})$. If $moves(s_0) = \emptyset$ then from axiom (A4) we get $\langle h, i\rangle\alpha \equiv \beta \wedge \alpha$ and the lemma holds. Let $moves(s_0) = \{a_1, \ldots, a_k\}$ and $\widehat{\lambda}(s_0) = i$.

Suppose $X \in R^i_h$, since X is a valid tree and $enabled(head(h), u)$ holds, there exist sets $Y_1, \ldots, Y_k$ such that for all $j : 1 \leq j \leq k$, $w_j = root(Y_j)$ and $u \xrightarrow{a_j} w_j$. Since u is an i node we have that the strategy should choose a w_j such that $u \xrightarrow{a_j} w_j$ and $X' \in R^i_{h_{a_j}}$ where $X = (u, w_j) \cdot X'$. By induction hypothesis we have $\widehat{w_j} \wedge \langle h_{a_j}, i\rangle\widetilde{\mathcal{X}}$ is consistent. Hence from axiom (A4) we conclude $\widehat{u} \wedge \langle h, i\rangle\widetilde{\mathcal{X}}$ is consistent.

Suppose $\widehat{u} \wedge \langle h, i\rangle\widetilde{\mathcal{X}}$ is consistent. From axiom (A4) it follows that there exists $w_1, \ldots, w_k$ such that for all $j : 1 \leq j \leq k$, we have $u \xrightarrow{a_j} w_j$ and hence $enabled(h, u)$ holds. Let $\mathcal{X} = \{v_1, \ldots, v_m\}$, from axiom (A4) we have $\widehat{u} \wedge (\bigvee_{a \in \Sigma} \langle a\rangle\langle h_a, i\rangle\widetilde{\mathcal{X}})$ is consistent. Hence we get that there exists w_j such that $u \xrightarrow{a_j} w_j$ and $\widehat{w_j} \wedge \langle h_a, i\rangle\widetilde{\mathcal{X}}$ is consistent. By induction hypothesis there exists X' which is a valid tree with $frontier(X') \subseteq \mathcal{X}$, $root(X') = w_j$ and $X' \in R^i_{h_a}$. By definition of R^i we get $(u, w_j) \cdot X' \in R^i_h$.

Let $\widehat{\lambda}(s_0) = \overline{\imath}$ and suppose $X \in R^i_h$. Since $enabled(head(h), u)$ holds and X is a valid tree, there exist sets $Y_1, \ldots, Y_k$ such that for all $j : 1 \leq j \leq k$, $w_j = root(Y_j)$ and $u \xrightarrow{a_j} w_j$. Since u is an $\overline{\imath}$ node, any strategy of i need to have all the branches at u (by definition of strategy). Thus we get: for all w_j with $u \xrightarrow{a_j} w_j$, there exists X_j with $root(X_j) = w_j$ such that $X_j \in R^i_h$ and $X = \bigcup_{j=1,\ldots,k}(u, w_j) \cdot X_j$. By induction hypothesis and the fact that $\mathcal{X}_j = frontier(X_j) \subseteq \mathcal{X}$, we have $\widehat{w_j} \wedge \langle h, i\rangle\widetilde{\mathcal{X}}$ is consistent. Hence from axiom (A4) we get $\widehat{u} \wedge \langle h, i\rangle\widetilde{\mathcal{X}}$ is consistent.

Likewise, using axiom (A4) we can show that if $\widehat{u} \wedge \langle h, i\rangle\widetilde{\mathcal{X}}$ is consistent then there exists a X' which is a valid tree with $frontier(X') \subseteq \mathcal{X}$ and $root(X') = u$ such that $X' \in R^i_h$.

Lemma 5.2. For all $i \in N$, for all $g \in \Gamma$, for all $X \subseteq (W \times W)^*$ with $\mathcal{X} = frontier(X)$ and $u \in W$, if $\widehat{u} \wedge \langle h, i\rangle\widetilde{\mathcal{X}}$ is consistent then there exists X' which is a valid tree with $frontier(X') \subseteq \mathcal{X}$ and $root(X') = u$ such that $X' \in R^i_h$.

Proof. By induction on the structure of g.

- $g = h$: The claim follows from Lemma 5.1 item 2.
- $g = g_1 \cup g_2$: By axiom (A3a) we get $\widehat{u} \wedge \langle g_1, i\rangle\widetilde{\mathcal{X}}$ is consistent or $\widehat{u} \wedge \langle g_2, i\rangle\widetilde{\mathcal{X}}$ is consistent. By induction hypothesis there exists X_1 which is a valid tree with $frontier(X_1) \subseteq \mathcal{X}$ and $root(X_1) = u$ such that $(u, X_1) \in R^i_h$ or there exists X_2 which is a valid tree with $frontier(X_2) \subseteq \mathcal{X}$ and $root(X_2) = u$ such that $X_2 \in R^i_h$. Hence we have $X_1 \in R^i_{g_1 \cup g_2}$ or $X_2 \in R^i_{g_1 \cup g_2}$.
- $g = g_1; g_2$: By axiom (A3b), $\widehat{u} \wedge \langle g_1, i\rangle\langle g_2, i\rangle\widetilde{\mathcal{X}}$ is consistent. Hence $\widehat{u} \wedge \langle g_1, i\rangle(\bigvee(\widehat{w} \wedge \langle g_2, i\rangle\widetilde{\mathcal{X}}))$ is consistent, where the join is taken over all $w \in \mathcal{Y} = \{w \mid w \wedge \langle g_2, i\rangle\widetilde{\mathcal{X}}$ is consistent $\}$. So $\widehat{u} \wedge \langle g_1, i\rangle\widetilde{\mathcal{Y}}$ is consistent. By induction hypothesis, there exists Y' which is a valid tree with $\mathcal{Y}' = frontier(Y') \subseteq \mathcal{Y}$ and $root(Y') = u$ such that $(u, Y') \in R^i_{g_1}$. We also have that for all $w \in \mathcal{Y}$, $\widehat{w} \wedge \langle g_2, i\rangle\widetilde{\mathcal{X}}$ is consistent. Therefore we get for all $w_j \in \mathcal{Y}' = \{w_1, \ldots, w_k\}$,

$\widehat{w}_j \wedge \langle g_2, i\rangle\widetilde{\mathcal{X}}$ is consistent. By induction hypothesis, there exists X_j which is a valid tree with $\mathcal{X}_j = \mathit{frontier}(X_j) \subseteq \mathcal{X}$ and $\mathit{root}(\mathcal{X}_j) = w_j$ such that $X_j \in R^i_{g_2}$. Let X' be the tree in $Y'; \{X_j \mid j = 1, \dots, k\}$ obtained by pasting X_j to the leaf node w_j in Y'. We get $X' \in R^i_{g_1;g_2}$.

– $g = g_1 \| g_2$: Note that for all $g \in \Gamma$ and $h \in \mathit{head}(g)$, the complexity of $g \backslash h$ is less than that of g. Therefore by making use of axiom (A3c) we can show that there exists X' with $\mathit{frontier}(X') \subseteq \mathcal{X}'$ and $\mathit{root}(X') = u$ such that $X' \in R^i_h$.

References

1. Abrahamson, K.: Decidability and expressiveness of logics of processes. PhD thesis, Dept. of Computer Science, Univ. of Washington (1980)
2. Ågotnes, T.: Action and knowledge in alternating time temporal logic. Synthese 149(2), 377–409 (2006)
3. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM 49, 672–713 (2002)
4. Broersen, J.: CTL.STIT: Enhancing ATL to express important multi-agent system verification properties. In: Proceedings of AAMAS 2010. ACM Press, New York (2010)
5. Broersen, J., 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)
6. 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)
7. Goranko, V.: Coalition games and alternating temporal logics. In: Proceedings of TARK 2001, pp. 259–272 (2001)
8. Harel, D.: Dynamic logic. In: Handbook of Philosophical Logic, vol. 2, pp. 496–604 (1984)
9. Harel, D., Kozen, D., Parikh, R.: Process logic: Expressiveness, decidability, completeness. Journal of Computer and System Sciences 25(2), 144–170 (1982)
10. Horty, J.: Agency and Deontic Logic. Oxford University Press, Oxford (2001)
11. Lange, M., Lutz, C.: 2-EXPTIME lower bounds for propositional dynamic logics with intersection. Journal of Symbolic Logic 70(4), 1072–1086 (2005)
12. Parikh, R.: The logic of games and its applications. Annals of Discrete Mathematics 24, 111–140 (1985)
13. Pauly, M.: Logic for Social Software. PhD thesis, Univ. of Amsterdam (2001)
14. Peleg, D.: Concurrent dynamic logic. Journal of the ACM 34(2), 450–479 (1987)
15. Ramanujam, R., Simon, S.: Dynamic logic on games with structured strategies. In: Proceedings of KR 2008, pp. 49–58. AAAI Press, Menlo Park (2008)
16. van Benthem, J.: Extensive games as process models. Journal of Logic Language and Information 11, 289–313 (2002)
17. van Benthem, J., Ghosh, S., Liu, F.: Modelling simultaneous games with dynamic logic. Synthese (Knowledge, Rationality and Action) 165, 247–268 (2008)
18. van der Hoek, W., Jamroga, W., Wooldridge, M.: A logic for strategic reasoning. In: Proceedings of AAMAS 2005, pp. 157–164 (2005)
19. Walther, D., van der Hoek, W., Wooldridge, M.: Alternating-time temporal logic with explicit strategies. In: Proceedings of TARK 2007, pp. 269–278 (2007)

Exploring the Boundary of Half Positionality[*]

Alessandro Bianco, Marco Faella, Fabio Mogavero, and Aniello Murano

Università degli Studi di Napoli "Federico II", Italy
{alessandrobianco,mfaella,mogavero,murano}@na.infn.it

Abstract. Half positionality is the property of a language of infinite words to admit positional winning strategies, when interpreted as the goal of a two-player game on a graph. Such problem applies to the automatic synthesis of controllers, where positional strategies represent efficient controllers. As our main result, we describe a novel sufficient condition for half positionality, more general than what was previously known. Moreover, we compare our proposed condition with several others, proposed in the recent literature, outlining an intricate network of relationships, where only few combinations are sufficient for half positionality.

1 Introduction

Games are widely used in computer science as models to describe multi-agent systems, or the interaction between a system and its environment [KVW01, McN93, Tho95, Zie98]. Usually, the system is a component that is under the control of its designer and the environment represents all the components the designer has no direct control of. In this context, a game allows the designer to easily check whether the system can force some desired behavior (or avoid an undesired one), independently of the choices of the other components. Further, game algorithms may automatically synthesize a design that obtains the desired behavior.

We consider games played by two players on a finite graph, called *arena*. The arena models the interaction between the entities involved: a node represents a state of the interaction, and an edge represents progress in the interaction. We consider turn-based games, i.e. games where each node is associated with only one player, who is responsible for choosing the next node. A sequence of edges in the graph represents a run of the system. Player 0 wants to force the system to follow an infinite run with a desired property, expressed as a language of infinite words called *goal*. The objective of player 1 is the opposite. In this context, a *strategy* for a player is a predetermined decision that the player makes on all possible finite paths ending with a node associated to that player. A strategy is *winning* for a player if it allows him to force a desired path no matter what strategy his opponent uses. A key property of strategies is the amount of memory that they require, in order to choose their next move. The simplest strategies do not need to remember the past history of the game, i.e., their choices only depend on the current state in the game. Such strategies are called *positional*.

We are interested in determining the existence of a winning strategy for one of the players, and possibly compute an effective representation of such a strategy. To this

[*] Work partially supported by MIUR PRIN Project n.2007-9E5KM8.

aim, suitable techniques have been developed when the desired behavior of a player is specified in particular forms (see [KVW01] for temporal logic specifications, and [EJ91, McN93, Mos91] for parity conditions). In synthesis problems, only positional strategies may be suitable to concrete implementation, due to space constraints. In fact, in principle even a positional strategy, which is a function from states to moves, needs an amount of storage that is proportional to the size of the state-space of the system. Symbolic representations can mitigate such issues [Cac02]. For this reason, it is useful to know when a given goal guarantees that if player 0 (respectively, player 1) has a winning strategy then he has a positional one. This property is called *half positionality* (in the following, HP) for player 0 (resp., player 1). If a goal is HP for both players, the goal is called *full positional* (FP). Notice that HP is more important than FP in the synthesis applications we are referring to. In these applications, player 0 represents the controller to be synthesized and player 1 the environment. Hence, we are only interested in obtaining simple winning strategies for one of the two players, namely for player 0.

Full positionality has been studied and characterized: in [EJ91, McN93, Mos91], it was proved that the parity winning conditions are full positional and in [GZ05] Zielonka and Gimbert defined a complete characterization for full positional determined goals on finite arenas. In that paper, it is proven that a goal is FP if and only if both the goal and its complement satisfy two properties called *monotonicity* and *selectivity*. On the other hand, a goal (but not its complement) being monotone and selective is not sufficient for HP. Moreover, HP has been specifically investigated by Kopczyński in [Kop07, Kop06]. There, the author defines sufficient conditions for a goal to be HP on all finite arenas. However, no characterization of half positional goals has been found so far. Positionality of games with infinitely many moves has been studied in [CN06, Gra04].

In his work, Kopczyński proves that if a goal is *concave* and *prefix-independent* then it is HP. In this paper, we investigate half positionality on finite arenas and we provide a novel sufficient condition for a goal to be HP on all finite arenas. We prove that if a goal is *strongly monotone* and *strongly concave*, then it is HP. As the names suggest, strong monotonicity is derived by the notion of monotonicity in [GZ05] and strong concavity refines the notion of concavity defined in [Kop06]. We prove that our condition constitutes an improvement over that defined in [Kop06], because it allows to classify as HP a broader set of goals. Several examples show that our condition is somewhat robust, in the sense that it is not trivial to further strengthen the result.

Overview. The rest of the paper is organized as follows. In Section 2, we introduce some preliminary notation. In Section 3, we introduce and define the new properties of goals sufficient to ensure half positionality. We prove that such properties describe a wider set of goals than the properties in [Kop06] and we show that some weaker conditions are not sufficient. In Section 4, we prove that our conditions are not necessary to half-positionality. In Section 5, we analyze the conditions of [GZ05], relating them to half positionality. We show that natural stronger forms of such conditions are not sufficient, and we conclude by defining a characterization for half-positionality on game graph whose nodes belong all to one player only. Finally, we provide some conclusions is Section 6.

2 Preliminaries

Let X be a set and i be a positive integer. By X^i we denote the Cartesian product of X with itself i times and by X^* (resp., X^ω) the set of finite (resp., infinite) sequences of elements of X. The set X^* also contains the *empty word* ε. A *finite language* (resp. *infinite language*) on the alphabet X is a subset of X^* (resp. X^ω). A *finite state automaton* is a tuple (X, Q, δ, q_0, F) where X is an alphabet, Q a set of states, $q_0 \in Q$ an initial state, $F \subseteq Q$ a set of final states and $\delta : Q \times X \to 2^Q$ a transition function. A *run* of the automaton on a sequence $x_1 \dots x_k \in X^*$, is a sequence $q_0 \dots q_k \in Q^*$ such that for each $i \in \{1, \dots, k\}$ we have $q_i \in \delta(q_{i-1}, x_i)$. A word $x \in X^*$ is said *accepted* by the automaton if there exists a run $q_0, \dots, q_k$ on x ending in a final state $q_k \in F$. A language is said *regular* iff there exists a finite state automaton that accepts all and only the words belonging to it. Moreover, by $\mathbb{N}$ we denote the set of non-negative integers.

For a non-negative integer k, let $[k] = \{0, 1, \dots, k\}$. A *word* on the alphabet $[k]$ is a finite or infinite sequence of elements of $[k]$, a *language* over the alphabet $[k]$ is a set of words over $[k]$. For each element $i \in [k]$, we often use i to denote the language $\{i\}$, when the meaning is clear from the context.

Arenas. A *k-colored arena* is a tuple $A = (V_0, V_1, v_{\text{ini}}, E)$, where V_0 and V_1 are a partition of a finite set V of *nodes*, $v_{\text{ini}} \in V$ is the *initial node*, and $E \subseteq V \times [k] \times V$ is a set of *colored edges* such that for each node $v \in V$ there is at least one edge exiting from v. A colored edge $e = (u, a, v) \in E$ represents a connection *colored* with a from the node u, named *source* of e, to the node v, named *destination* of e. In the following, we simply call a k-colored arena an *arena*, when k is clear from the context. For a node $v \in V$, we call ${}_vE = \{(v, a, w) \in E\}$ and $E_v = \{(w, a, v) \in E\}$ the sets of edges exiting and entering v, respectively.

For a color $a \in [k]$, we denote by $E(a) = \{(v, a, w) \in E\}$ the set of edges colored with a. A *finite path* ρ is a finite sequence of edges $\{(v_i, a_i, v_{i+1})\}_{i \in \{0, \dots, n-1\}}$, and its *length* $|\rho|$ is the number of edges it contains. We use $\rho(i)$ to indicate the i-th edge of ρ. Sometimes, we write the path ρ as $v_0 v_1 \dots v_n$, when the colors are not important. An *infinite path* is defined analogously, i.e., it is an infinite sequence of edges $\{(v_i, a_i, v_{i+1})\}_{i \in \mathbb{N}}$. For a path (finite or infinite) ρ and an integer i, we denote by $\rho^{\leq i}$ the *prefix* of ρ containing i edges. The *color sequence* of a finite (resp. infinite) path $\rho = \{(v_i, c_i, v_{i+1})\}_{i \in \{0, \dots, n-1\}}$ (resp. $\rho = \{(v_i, c_i, v_{i+1})\}_{i \in \mathbb{N}}$) is the sequence $Col(\rho) = \{c_i\}_{i \in \{0, \dots, n-1\}}$ (resp. $Col(\rho) = \{c_i\}_{i \in \mathbb{N}}$) of the colors of the edges of ρ. For two color sequences $x, y \in [k]^\omega$, the *shuffle* of x and y, denoted by $x \otimes y$ is the language of all the words $z_1 z_2 z_3 \dots \in [k]^\omega$, such that $z_1 z_3 \dots z_{2h+1} \dots = x$ and $z_2 z_4 \dots z_{2h} \dots = y$, where $z_i \in [k]^*$ for all $i \in \mathbb{N}$. For two languages $M, N \subseteq [k]^\omega$, the *shuffle* of M and N is the set $M \otimes N = \cup_{n \in N, m \in M} m \otimes n$.

Games. A *k-colored game* is a pair $G = (A, W)$, where $A = (V_0, V_1, v_{\text{ini}}, E)$ is a k-colored arena and $W \subseteq [k]^\omega$ is a set of color sequences called *goal*. By $\overline{W}$ we denote the set $[k]^\omega \setminus W$. Informally, we assume that the game is played by two players, referred to as player 0 and player 1. The players construct a path starting at v_{ini} on the arena A; such a path is called *play*. Once the partial play reaches a node $v \in V_0$, player 0 chooses an edge exiting from v and extends the play with this edge; once the partial play reaches a node $v \in V_1$, player 1 makes a similar choice. Player 0's aim is to make the play have color sequence in W, while player 1's aim is to make the play have color sequence in $\overline{W}$. We now define some notation in order to formalize the previous intuitive description. For $h \in \{0,1\}$, let $E_h = \{(v,c,w) \in E \mid w \in V_h\}$ be the set of edges ending into nodes of player h. Moreover, let ε be the empty word. A *strategy* for player h is a function $\sigma_h : \varepsilon \cup (E^* E_h) \to E$ such that, if $\sigma_h(e_0 \dots e_n) = e_{n+1}$, then the destination of e_n is the source of e_{n+1}, and if $\sigma_h(\varepsilon) = e$, then the source of e is v_{ini}. Intuitively, σ_h fixes the choices of player h for the entire game, based on the previous choices of both players. The value $\sigma_h(\varepsilon)$ is used to choose the first edge in the game. A strategy σ_h is *positional* iff its choices depend only on the last node of the partial play, i.e., for all partial plays ρ and ρ' with the same last node, it holds that $\sigma_h(\rho) = \sigma_h(\rho')$. A play $\{e_i\}_{i \in \mathbb{N}} \in E^\omega$ is *consistent* with a strategy σ_h iff *(i)* if $v_{\text{ini}} \in V_h$ then $e_0 = \sigma_h(\varepsilon)$, and *(ii)* for all $i \in \mathbb{N}$, if $e_i \in E_h$ then $e_{i+1} = \sigma_h(e_0 \dots e_i)$. An infinite play ρ is *winning* for player 0 (resp. player 1) iff $Col(\rho) \in W$ (resp. $Col(\rho) \notin W$). Note that, given two strategies, σ for player 0 and τ for player 1, there exists only one play consistent with both of them. This is due to the fact that the two strategies univocally determine the next edge at every step of the play. We call such a play $P_G(\sigma,\tau)$. A strategy for player h is *winning* iff all plays consistent with that strategy are winning for player h. A game is *determined* iff one of the two players has a winning strategy. A goal is *determined* iff all games $G = (A,W)$ are determined.

Concavity and Prefix Independence. A goal is said prefix independent if the adding or removing of a finite prefix on an infinite color sequence does not change the winning value of the sequence itself. Formally, a goal $W \subseteq [k]^\omega$ is *prefix independent* iff for all color sequences $x \in [k]^\omega$, and all finite words $z \in [k]^*$, $x \in W$ iff $zx \in W$. Following [Kop06], a goal is concave if switching infinitely often between two infinite color sequences, does not yield a better color sequence for player 0. Formally, a goal W is *concave* iff, for all words $x,y \in [k]^\omega$ and $z \in x \otimes y$, it holds that if $z \in W$ then $x \in W$ or $y \in W$. A goal W is *half positional* on an arena A iff, for all games $G = (A,W)$, if player 0 has a winning strategy then he has a positional winning strategy. A goal W is *half positional* iff it is half positional on all arenas A. As proved by Kopczyński, concave and prefix independence properties are sufficient conditions for half positionality.

Theorem 1 ([Kop06]). *All concave and prefix-independent goals are determined and half-positional.*

In the following, for a goal W and a pair of sets $M,N \in [k]^\omega$ we use the notation $M \leq_W N$ to mean that if M contains a winning word then N contains a winning word too, and the notation $M <_W N$ to mean that M contains only losing words and N contains at least a winning word. For ease of reading, when the goal W is clear from the contest, we write $M < N$ and $M \leq N$, respectively, for $M <_W N$ and $M \leq_W N$. With the following two lemmas, we reformulate the definition of concavity and prefix independence in terms of languages, rather than of single words.

Lemma 1. *A goal $W \subseteq [k]^\omega$ is prefix-independent iff for all color sequences $x \in [k]^*$ and sets of color sequences $M \subseteq [k]^\omega$ we have that $xM \leq M$ and $M \leq xM$.*

Proof. Suppose that W is prefix independent. If M contains a winning word m, then xM contains the winning word xm, and we have both $xM \leq M$ and $M \leq xM$. If M contains only losing words m, then xM contains only losing words xm and we have both $xM \leq M$ and $M \leq xM$.

Suppose now that, for all languages $M \subseteq [k]^{\omega}$, we have $xM \leq M$ and $M \leq xM$. Moreover, suppose by contradiction that $W \neq xW$. Then, there exists a word m such that $xm \notin W$. Hence, for the language $M = \{m\}$ we do not have $M \leq xM$. $\square$

Lemma 2. *A goal $W \subseteq [k]^{\omega}$ is concave iff for all languages $M,N \subseteq [k]^{\omega}$ we have that $M \otimes N \leq M \cup N$.*

Proof. Suppose that W is concave. For all $M,N \subseteq \overline{W}$, we have that $M \otimes N \subseteq \overline{W}$. So, for all languages $M,N \in [k]^{\omega}$, if M or N contains a winning word in W, we have in both cases $M \otimes N \leq M \cup N$; conversely, if M and N contain only losing words, by hypothesis, so does $M \otimes N$. Hence, we have that $M \otimes N \leq M \cup N$.

Suppose now that for all languages $M,N \subseteq [k]^{\omega}$ we have $M \otimes N \leq M \cup N$. Then, if M and N contain only losing words, $M \otimes N$ must contain only losing words too. Thus, for all $M,N \in \overline{W}$ we have that $M \otimes N \subseteq \overline{W}$. $\square$

3 Novel Properties of Goals

In this section, we present two properties of goals: *strong monotonicity* and *strong concavity*. Their aim is to refine the properties of prefix-independence and concavity in such a way they still imply half positionality for player 0. The property of monotonicity was first defined in [GZ05]. It states that two color sequences with a common prefix cannot exchange their winning value by switching to another prefix.

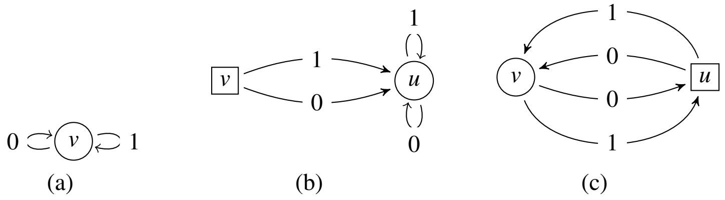


Fig. 1. Three game arenas

Definition 1. *A goal $W \subseteq [k]^{\omega}$ is* monotone *iff for all words $x \in [k]^*$ and all regular languages $M,N \subseteq [k]^{\omega}$ it holds that $xM < xN$ implies that for all $y \in [k]^*$ it is $yM \leq yN$.*

Here, we define a stronger version of monotonicity, it asks that the above property should hold even on non-regular languages.

Definition 2. *A goal $W \subseteq [k]^{\omega}$ is* strongly monotone *if, for all words $x \in [k]^*$, $m,n \in [k]^{\omega}$, such that $xm \notin W$ and $xn \in W$, for all $y \in [k]^*$ it holds that either $ym \notin W$ or $yn \in W$.*

In the following we make use of an equivalent definition of strong monotonicity that operates on languages.

Lemma 3. *A goal $W \subseteq [k]^\omega$ is strongly monotone iff, for all words $x \in [k]^*$ and languages $M, N \subseteq [k]^\omega$, it holds that $xM < xN$ implies that for all $y \in [k]^*$ it is $yM \leq yN$.*

Strong monotonicity represents a weakening of the property of prefix independence that requires, instead, that the winning nature of a word does not change by changing a finite prefix. Indeed, the following lemma holds.

Lemma 4. *All prefix-independent goals are strongly monotone. Moreover, there is a goal which is strongly monotone, but not prefix-independent.*

Proof. For the first part, we have by hypothesis that, for all $x \in [k]^*$, and $M \subseteq [k]^\omega$, it holds that $M \leq xM \leq M$. Now, take two languages $M, N \subseteq [k]^\omega$, and suppose that there exists an $x \in [k]^*$ such that $xM < xN$, then for all $y \in [k]^*$ we have $yM \leq M \leq xM \leq xN \leq N \leq yN$.

For the second part, let $k = 1$, a strongly monotone and prefix-dependent goal is given by the language of all words containing at least one 0, i.e., $W = [k]^*0[k]^\omega$. It is easy to see that the goal is not prefix-independent, because the word 1^ω is losing while the word 01^ω is winning. We show that the goal is strongly monotone. Consider two languages $M, N \subseteq [k]^*$, and suppose that there exists an $x \in [k]^*$ such that $xM < xN$, then xN contains a winning word and xM contains only losing words. Observe first that x cannot contain 0, or else all words in xM would be winning. So $x \in 1^*$, there exists a word in N that contains 0, and all words in M contain only 1's. So, for each $y \in [k]^*$, there is always a word in yN containing 0. Since yN contains a winning word, we have $yM \leq yN$. □

We investigate the usefulness of strong monotonicity. First, we show that strong monotonicity cannot replace prefix-independence in the hypotheses of Theorem 1.

Lemma 5. *There is a strongly monotone and concave goal which is not half-positional.*

Proof. For $k = 1$, the strongly monotone and concave goal is $W = [k]^*01^\omega$. We prove first that the goal is strongly monotone and concave. A word is losing if and only if it is either 1^ω or it does not have 1^ω as a suffix. Let $x \in [k]^*$, $n, m \in [k]^\omega$ with $xn, xm \notin W$. There are two situations to discuss. First, assume that x does not contain 0. Then, n and m may be both 1^ω in which case $x(m \otimes n) = 1^\omega$ or at least one between n and m contains 0 infinitely often, thus the shuffle of n and m contains only words that pick colors from both the sequences infinitely often and thus only words that contain 0 infinitely often. So, $x(m \otimes n)$ contains losing word even in this case. Instead, assume that x contains 0. Then, n and m contain 0 infinitely often and the same reasoning above applies. So the goal is concave. Let $x \in [k]^*$, $n, m \in [k]^\omega$ such that $xm \notin W$ and $xn \in W$. We prove strong monotonicity by showing that for all $y \in [k]^*$ it holds that $ym \notin W$ or $yn \in W$. We again distinguish two cases. First, assume that x does not contain a 0. Then, n contains 0 and a suffix 1^ω thus for every $y \in [k]^*$, we have $yn \in W$ since it contains 0 and a suffix 1^ω. Instead, assume that x contains a 0. Then, m contains 0 infinitely often, thus for every $y \in [k]^*$ the word $ym \notin W$ since it contains 0 infinitely often. The above goal is not half-positional in the following arena $(\{v\}, \emptyset, v, \{(v,0,v),(v,1,v)\})$ (Fig. 1(a)), in such a game graph player 0 wins by choosing at least once the edge with color 0 and then always the edge with color 1. □

Observe that, in the previous counterexample, the key element that does not allow half positionality is the fact that player 0 prefers switching between two different behaviors finitely often and then progressing indefinitely along one of them. However, concavity just requires that player 0 prefers following a fixed behavior rather than switching between two different ones *infinitely often*. Thus, we introduce a modification to the property of concavity, requiring not only that alternating infinitely often between two losing words yields a losing word, but also that alternating *finitely* often between two losing words and then progressing along one of them yields a losing word.

Definition 3. *For two color sequences $x, y \in [k]^\omega$, the* strong shuffle *of x and y, denoted by $x \otimes_s y$ is the language containing*

1. *the set $x \otimes y$;*
2. *the words $z_1 z_2 \dots z_l z' \in [k]^\omega$, for odd l, $z_i \in [k]^*$ and $z' \in [k]^\omega$, such that it holds $x = z_1 z_3 \dots z_l z'$ and $y = z_2 z_4 \dots z_{l-1} y'$, for some $y' \in [k]^\omega$;*
3. *the words $z_1 z_2 \dots z_l z' \in [k]^\omega$, for even l, $z_i \in [k]^*$ and $z' \in [k]^\omega$, such that it holds $x = z_1 z_3 \dots z_{l-1} x'$ and $y = z_2 z_4 \dots z_l z'$, for some $x' \in [k]^\omega$.*

For two languages $M, N \subseteq [k]^\omega$, the strong shuffle *of M and N is the set $M \otimes_s N = \cup_{n \in N, m \in M} (m \otimes_s n)$.*

Definition 4. *A goal $W \subseteq [k]^\omega$ is* strongly concave *iff, for all words $x \in [k]^*$, $n, m \in [k]^\omega$, and $z \in x(m \otimes_s n)$, it holds that if $z \in W$ then either $xn \in W$ or $xm \in W$.*

It is immediate to see that a strongly concave goal is concave too. In the following, we make use of an equivalent definition of strong concavity that operates on languages.

Lemma 6. *A goal $W \subseteq [k]^\omega$ is strongly concave iff, for all words $x \in [k]^*$ and languages $M, N \subseteq [k]^\omega$, it holds that $x(M \otimes_s N) \leq xM \cup xN$.*

Even the property of strong concavity is not sufficient to ensure half positionality.

Lemma 7. *There is a strongly concave goal which is not half-positional.*

Proof. For $k = 1$ the strongly concave goal is $W = 0^\omega \cup 1^\omega$. Two losing words n and m contain at least an occurrence of the color 1 and an occurrence of the color 0, thus every word in their strong shuffle will contain at least an occurrence of color 1 and an occurrence of color 0 and it will be losing. So the strong concavity of the goal is proved. The above goal is not half-positional in the following 2-colored arena $(\{u\}, \{v\}, v, \{(v,0,u), (v,1,u), (u,0,u), (u,1,u)\})$, showed in Figure 1(b). In this arena player 0 wins the game by choosing forever the edge $(u,0,u)$ or the edge $(u,1,u)$ depending on what color was chosen by player 1 to reach u from v. □

In the previous counterexample, by choosing a different prefix, player 1 can exchange the winning nature of the following choices of player 0. That is why strong monotonicity is essential since it somehow allows player 0 to operate while forgetting the past decisions taken by player 1.

We argue now that the two introduced properties of strong monotonicity and strong concavity are strictly less restrictive than the properties of prefix independence and concavity.

Lemma 8. *Concave and prefix-independent goals are strongly monotone and strongly concave.*

Proof. By Lemma 4 we already have that a prefix-independent goal is strongly monotone. It remains to show that a concave and prefix-independent goal is strongly concave.

For a language $M \subseteq [k]^\omega$, let $\mathit{suff}(M)$ and $\mathit{pref}(M)$ be the sets of suffixes and prefixes of words in M, respectively. By concavity, for all $M,N \subseteq [k]^\omega$ we have $M \otimes N \leq M \cup N$ and by prefix independence we have for all $M \in [k]^\omega$ and for all $x \in [k]^*$ $M \leq xM \leq M$. Take any word $x \in [k]^*$, and any two languages $M,N \subseteq [k]^\omega$. Then we have $x(M \otimes_s N) = x(M \otimes N) \cup x \cdot \mathit{pref}(M \otimes N) \cdot \mathit{suff}(N) \cup x \cdot \mathit{pref}(M \otimes N) \cdot \mathit{suff}(M)$. First, by prefix independence and then by concavity we have $x(M \otimes N) \leq M \otimes N \leq M \cup N \leq x(M \cup N) = xM \cup xN$. Then, $x \cdot \mathit{pref}(M \otimes N) \cdot \mathit{suff}(T) \leq \mathit{suff}(T) \leq xT \leq xM \cup xN$, where $T \in \{M,N\}$. So, we have $x(M \otimes_s N) \leq xM \cup xN$. □

Lemma 9. *There exists a strongly monotone and strongly concave goal which is not prefix independent.*

Proof. Let $k = 1$, the goal is given by the set of words that either start with 1, or start with 0 and contain infinitely many 0's, i.e., $W = 0(1^*0)^\omega \cup 1[k]^*$. It is easy to see that the goal is not prefix-independent: indeed, for $M = 1^\omega$ we have that $0M \leq M$, but not $M \leq 0M$ since M contains only winning words and $0M$ only losing ones.

Next, we prove that the goal is strongly monotone. Consider $M,N \subseteq [k]^*$ and $x \in [k]^*$ and suppose that $xM < xN$, so xN contains a winning word and xM contains only losing ones. Observe that x does not start with 1, otherwise all words in xM would be winning. So, there are two situations to discuss: $x = \varepsilon$ or x starts with 0. If $x = \varepsilon$ then all words in M starts with 0 and have a suffix equal to 1^ω. Now for all $y \in 1[k]^*$ we have $yM \leq yN$ since all the words in all languages are winning; for all $y \in 0^*[k]^*$ we have $yM \leq yN$ because all the words in yM are losing since they start with 0 and have a suffix 1^ω. If instead x starts with 0 then there exists a word $n \in N$ that contains infinitely many 0, for every $y \in [k]^*$ the word yn will contain infinitely many 0 and it will be winning, thus for all $y \in [k]^*$ we will have $yM \leq yN$.

Now we prove that the goal is strongly concave. Consider $x \in [k]^*$, $M,N \subseteq [k]^\omega$ and $K \subseteq [k]^*$. We want to prove that $x(M \otimes_s N) \leq xM \cup xN$. If the r.h.s. of the inequality contains a winning word, the inequality trivially holds. So, suppose that the r.h.s. does not contain a winning word, so it cannot be $x \in 1[k]^*$ but it must be $x \in 0[k]^* \cup \{\varepsilon\}$. If x starts with 0, every word in M,N contains a suffix 1^ω and all words in $M \otimes_s N$ contain a suffix 1^ω. So, $M \otimes_s N$ contains only losing words. If $x = \varepsilon$, every word in M,N contains a suffix 1^ω and starts with 0, so all words in $M \otimes_s N$ contain a suffix 1^ω and start with 0, and therefore they are losing. □

4 A Sufficient Condition for Half Positionality

In this section, we prove that determinacy, strong monotonicity and strong concavity are sufficient but not necessary conditions to half positionality for player 0.

Theorem 2. *All determined, strongly monotone and strongly concave goals are half-positional.*

Proof. The proof proceeds by induction on the number of edges exiting from the nodes controlled by player 0 in the game arena. As a base case in the graph G for each node controlled by player 0 there exists only one exiting edge. In such a graph player 0 has only one possible strategy which is positional. So, the result is trivially true. Suppose that in the arena there are n edges exiting from nodes of player 0 and that, for all graphs with at most $n-1$ edges exiting from nodes of player 0, if player 0 has a winning strategy he has a positional one. Let t be a node of player 0 in G such that there is more than one edge exiting from t. We can partition the set of edges exiting from t in two disjoint non-empty sets E_α and E_β. Let G_α and G_β be the two subgraphs obtained from G by removing the edges of E_β and E_α, respectively. There are two cases to discuss.

First, suppose that in G_α or G_β player 0 has a winning strategy. Then, by inductive hypothesis he has a positional winning strategy. It is easy to see that such a strategy is winning in G too. Indeed, since player 0 controls the node t, he is able to force the play to stay always in G_α or G_β. Suppose now that player 0 has no winning strategy in G_α and in G_β. We prove the thesis by showing that player 0 has no winning strategy in G. By determinacy, there exist two strategies τ_α and τ_β winning for player 1 in G_α and G_β, respectively.

Let σ be a strategy of player 0 in G, we show that there exists a strategy of player 1 in G winning in G against σ. If one of the plays $P(\sigma,\tau_\alpha)$ or $P(\sigma,\tau_\beta)$ does not pass through t then that play is in G_α and G_β and so it is winning for player 1 who is using his winning strategy on one of the graphs.

Suppose now that both of the above plays pass through t. Let x_α and x_β be respectively the color sequences of the prefixes of $P(\sigma,\tau_\alpha)$ and $P(\sigma,\tau_\beta)$, up to the first occurrence of t. Let M_α and M_β be the sets of color sequences of suffixes after respectively a prefix x_α and x_β of plays consistent respectively with τ_α and τ_β. Observe that $x_\alpha M_\alpha$ and $x_\beta M_\beta$ contain plays consistent respectively with τ_α in G_α and τ_β in G_β, and such plays are losing for player 0. We prove now that either $x_\alpha M_\beta$ or $x_\beta M_\alpha$ contains only losing words for player 0. Indeed, if $x_\alpha M_\beta$ contains a winning word, we have that $x_\alpha M_\alpha < x_\alpha M_\beta$, since $x_\alpha M_\alpha$ contains only plays losing for player 0. Then, by strong monotonicity we have that, for all $y \in C^*$, it holds $yM_\alpha \leq yM_\beta$ and hence $x_\beta M_\alpha \leq x_\beta M_\beta$. Since $x_\beta M_\beta$ contains only losing words, so does $x_\beta M_\alpha$.

Suppose without loss of generality that $x_\beta M_\alpha$ contains only losing words. Then, we construct the strategy τ'_α, which behaves like τ_α on all partial plays which do not have a prefix x_β. When the partial play has a prefix x_β, it behaves like τ_α when it sees x_α in place of x_β. More formally $\tau'_\alpha(x_\beta\pi) = \tau_\alpha(x_\alpha\pi)$, and in the other cases $\tau'_\alpha(\pi) = \tau_\alpha(\pi)$. Let $\tau'_\beta = \tau_\beta$. We construct a strategy τ in G: at the beginning the strategy behaves like τ_β; when the play passes through t, depending on what subgraph the last edge from t chosen by player 0 belongs to, the strategy τ behaves like τ'_α or τ'_β when they are applied only to the initial prefix up to t and all the loops from t to t, where the first edge belongs to G_α or G_β, respectively.

Formally, for all prefixes π that do not pass through t, we have $\tau(\pi) = \tau_\beta(\pi)$; if π_{i,γ_i} is a loop from t to t with first edge in G_{γ_i}, for all prefixes $\pi = x\pi_{1,\gamma_1},\ldots,\pi_{n,\gamma_n}\pi_\gamma$, we have $\tau'(\pi) = \tau'_\gamma(x(\prod_{\gamma_i=\gamma}\pi_{i,\gamma_i}\pi_\gamma))$. The play $P(\sigma,\tau)$ coincides with $P(\sigma,\tau_\beta)$ up to t, so it has a prefix with color sequence x_β. After that prefix, the play develops in parallel and alternates pieces of two plays: one in G_β consistent with τ_β, and the other in G_α

consistent with τ'_α. So, the color sequence of the two suffixes are respectively in M_β and in M_α.[1] Hence, the color sequence of the suffix after x_β of the play $P(\sigma,\tau)$ lies in the shuffle of M_α and M_β. By strong concavity we have that $Col(P(\sigma,\tau)) \in x_\beta(M_\alpha \otimes_s M_\beta) \leq x_\beta M_\alpha \cup x_\beta M_\beta$. Since both $x_\beta M_\alpha$ and $x_\beta M_\beta$ contain only losing words, we have that $Col(P(\sigma,\tau))$ is a losing word for player 0. Hence, for all strategies σ of player 0 there exists a strategy τ of player 1 winning over 0. We conclude that player 0 has no winning strategy. $\square$

Since strongly concavity implies concavity, the following result states that the conditions appearing as the hypothesis of the previous theorem and of Theorem 1 are not a complete characterizations for half positional goals.

Lemma 10. *There exists a goal that is half positional but not concave.*

Proof. The half positional goal is $W = [k]^*1[k]^*1[k]^\omega$. The goal states that player 0 tries to make color 1 occur at least twice. It is half positional because in every point in a play player 0 does not need to look at the past, but just tries to form a path that passes through as many edges colored with 1 as possible. For a more formal proof, see Lemma 14.

We show that the goal is not concave: let $x = \varepsilon$, $n, m = 10^\omega$, then we have $xn, xm \notin W$, but $t = 110^\omega \in m \otimes n$ with $xt \in W$, hence the goal is not concave. $\square$

5 Selectivity

Here, we discuss how the properties presented in [GZ05] as a characterization of full positionality relate to half positionality. We already presented monotonicity in the previous section. The second property introduced in [GZ05] is similar to the property of strong concavity with the exception that in the shuffle the interleaving of words is allowed only at certain points.

Definition 5. *Let $M \subseteq [k]^*$. Then, with the notation $\langle M \rangle$ we define the set of all words $m \in [k]^\omega$ such that every prefix of m is a prefix of a word in M.*

Definition 6. *A goal W is* selective *iff for all $x \in [k]^*$ and for all regular languages $M, N, K \subseteq [k]^*$ we have that $x\langle (M \cup N)^* K \rangle \leq x\langle M^* \rangle \cup x\langle N^* \rangle \cup x\langle K \rangle$.*

The two conditions of selectivity and monotonicity provide a complete characterization of full positionality. Precisely, a goal W is full positional iff both W and $\overline{W}$ are selective and monotone [GZ05]. The proof makes use of the fact that, assuming that player 1 uses a positional strategy, player 0 can play on the graph induced by that strategy, and hence construct paths whose prefixes are recognizable by the automaton described by the game graph. We investigated the hypothesis that monotonicity and selectivity of W were sufficient to half positionality. However, the two conditions are not directly applicable, since they operate on regular languages. Indeed, when player 1 can use a non-positional strategy, the path constructed by player 0 is taken from a simple graph no more and it

[1] Note that it is possible that one of the two suffixes does not progress indefinitely.

does not belong to a language recognized by an automaton. Hence, we strengthened the conditions of monotonicity and selectivity in order to take into account all possible paths that could be formed by player 0 together with a non-positional strategy of player 1.

Definition 7. *A goal W is* strongly selective *iff for all $x \in [k]^*$ and for all languages $M,N,K \subseteq [k]^*$ we have that $x\langle (M \cup N)^* K\rangle \leq x\langle M^*\rangle \cup x\langle N^*\rangle \cup x\langle K\rangle$.*

Selectivity and strong selectivity represent two weaker properties than strong concavity.

Lemma 11. *Every strongly concave goal is strongly selective.*

Proof. For all words $x \in [k]^*$, for all languages $M,N,K \subseteq [k]^*$, we have that $x\langle (M \cup N)^* K\rangle \subseteq x((\langle M^*\rangle \otimes_s \langle N^*\rangle) \otimes_s \langle K\rangle) \leq x\langle M^*\rangle \cup x\langle N^*\rangle \cup x\langle K\rangle$.

Unfortunately, the strong versions of selectivity and concavity proved not to be sufficient conditions to half positionality[2].

Lemma 12. *There is a strongly monotone and strongly selective goal which is not half-positional.*

Proof. Let $k \in \mathbb{N}$, for all colors $i \in [k]$ and finite paths π, let $|\pi|_i$ be the number of edges colored by i on π, and let $|\pi|$ be the number of edges in π. Moreover for all $n \in \mathbb{N}$ let $\pi^{\leq n}$ be the prefix of length n of π. The strongly monotone and strongly selective goal is the set W of all the infinite words m such that, for all colors $i \in [k]$, the limit $\lim_{n\to+\infty} \frac{|m^{\leq n}|_i}{|m^{\leq n}|}$ exists and is finite. The goal is prefix independent. Indeed, let $\pi = x\pi'$ then for all $i \in [k]^*$ we have $\lim_{n\to+\infty} \frac{|\pi'^{\leq n}|_i}{|\pi'^{\leq n}|} = \lim_{n\to+\infty} \frac{|\pi^{\leq n+|x|}|_i - |x|_i}{|\pi^{\leq n+|x|}| - |x|} = \lim_{m\to+\infty} \frac{|\pi^{\leq m}|_i}{|\pi^{\leq m}|}$. The goal is also strongly selective. Indeed, suppose by contradiction that there exist a sequence $x \in [k]^*$, and three languages $M,N,K \subseteq [k]^*$ such that $x\langle (M \cup N)^* K\rangle$ contains one winning word and $x\langle M^*\rangle \cup x\langle N^*\rangle \cup x\langle K\rangle$ contains only losing words. In this case, M and N must be empty else any periodic word $\pi = m^\omega \in M^* \cup N^*$ with $m \in M \cup N$ has a finite limit $\lim_{n\to+\infty} \frac{|\pi^{\leq n}|_i}{|\pi^{\leq n}|} = \frac{|m|_i}{|m|}$, for all colors i. So, the set $\langle x(M \cup N)^* K\rangle = x\langle K\rangle$ and contains only losing words which is a contradiction. The above goal is not half-positional in the following arena $(\{u\},\{v\},u,\{(v,0,u),(v,1,u),(u,0,v),(u,1,v)\})$ with $k = 1$ (Fig 1(c)). Player 0 can win with a strategy with memory by choosing from V' to V the opposite of the color that player 1 chose from V to V' right before, thus yielding a path in $[k]^*(10)^\omega$ which has limit $\frac{1}{2}$ for both colors. However if player 0 uses a positional strategy, it will only choose one color from V' to V, let suppose without loss of generality that he chooses color 0. The player 1 can force a path $\pi = \prod_{i=0}^{+\infty}(00)^{2^i}(10)^{2^i}$. Then we have $|\prod_{i=0}^{l}(00)^{2^i}(10)^{2^i}| = \sum_{i=0}^{l} 4 \cdot 2^i = 4(2^{l+1}-1)$, and $|(\prod_{i=0}^{l-1}(00)^{2^i}(10)^{2^i}) \cdot (00)^{2^l}| = 4(2^l + 2^{l-1} - 1)$. Moreover, $|\prod_{i=0}^{l}(00)^{2^i}(10)^{2^i}|_1 = \sum_{i=0}^{l} \cdot 2^i = (2^{l+1}-1)$, and $|(\prod_{i=0}^{l-1}(00)^{2^i}(10)^{2^i}) \cdot (00)^{2^l}|_1 = \sum_{i=0}^{l-1} \cdot 2^i = 2^l - 1$. So we have $\frac{|\prod_{i=0}^{l}(00)^{2^i}(10)^{2^i}|_1}{|\prod_{i=0}^{l}(00)^{2^i}(10)^{2^i}|} = \frac{1}{4}$, moreover $\frac{|(\prod_{i=0}^{l-1}(00)^{2^i}(10)^{2^i}) \cdot (00)^{2^l}|_1}{|(\prod_{i=0}^{l-1}(00)^{2^i}(10)^{2^i}) \cdot (00)^{2^l}|} = \frac{2^l-1}{3(2^l-1)+2(2^l)} = \frac{2^l-1}{5(2^l)-3} = \frac{2^l-\frac{3}{5}}{5(2^l)-3} - \frac{\frac{2}{5}}{5(2^l)-3} < \frac{1}{5}$. This shows that in the limit $\frac{|\pi^{\leq n}|_1}{|\pi^{\leq n}|}$ oscillates between $\frac{1}{4}$ and something less than $\frac{1}{5}$. □

[2] We thank Zielonka and Gimbert for pointing out the counterexample.

Although the following theorem is obtained easily from the techniques developed in [GZ05], we think that it is worth mentioning that half positionality on arenas controlled only by player 0 is equivalent to the selectivity of the goal. Since the selectivity is similar in a way to strong concavity, we show that strong concavity is a condition useful to assert that, on decisions independent from player 1, player 0 prefers a fixed behavior rather than switching between two different ones. We prove the above statement by making use of the following lemma proved in [GZ05].

Lemma 13 ([GZ05]). *Let A be a finite co-accessible*[3] *automaton recognizing a language $L \subset [k]^*$ and having starting state q. Then, $\langle L\rangle$ is the set of infinite color sequences on the graph of A starting in q.*

Theorem 3. *A goal is selective iff it is half-positional on all arenas controlled by player* 0*.*

Proof. [only if] Suppose that a goal W is half-positional on all game graph controlled by player 0 but non-selective. Let $x \in [k]^*$ and $M,N,K \subseteq [k]^*$ be three recognizable languages such that $x\langle (M\cup N)^*K\rangle \not\subseteq x\langle M^*\rangle \cup x\langle N^*\rangle \cup x\langle K\rangle$. This means that there is a winning word in $x\langle (M\cup N)^*K\rangle$ and $x\langle M^*\rangle \cup x\langle N^*\rangle \cup x\langle K\rangle$ contains only losing words. Let G_x, G_M, G_N be the minimized finite automata recognizing the languages $\{x\}, M, N$, respectively, and having only one starting state with no transition returning to it and one final state with no transition exiting from it. Let G_K be the minimized finite automaton recognizing the language K, having only one starting state with no transition returning to it. We construct the game graph G by combining together the graphs G_x, G_M, G_N, G_K. Precisely we glue together the final state of G_x, the initial and final states of G_M and G_N and the initial state of G_K in a new node t. Observe that, by gluing together the initial and final states, the automata G_M, G_N recognize M^* and N^*, respectively. The initial state of G is the starting state of G_x. Thus the graph G recognizes the language $x(M\cup N)^*K$. Hence by Lemma 13, every infinite path in G is in $\langle x(M\cup N)^*K\rangle = x\langle (M\cup N)^*K\rangle$. Since this set contains a winning word, there is a winning strategy for player 0. However, if player 0 uses a positional strategy he cannot win. Indeed, player 0 reaches first the node t by constructing the color sequence x on G_x. In the node t player 0 chooses once and for all which of the subgraphs G_M, G_N, G_K he will use, so the infinite play will be of the form xm where m is an infinite path in G_M,G_N or G_K. By Lemma 13, $xm \in x\langle M^*\rangle \cup x\langle N^*\rangle \cup x\langle K\rangle$. But this set contains only losing words. Hence, xm is losing.

[if] Suppose that a goal W is selective, we prove by induction on the number of edges exiting from the nodes of the arena G controlled by player 0 that if there exists a winning strategy for player 0 then there exists a positional one. As base case there exists only one edge exiting from the nodes of G, hence player 0 has only one strategy, which is trivially positional. Suppose that in the arena there are n edges exiting from nodes of player 0 and that for all graphs with at most $n-1$ edges exiting from nodes of player 0, if player 0 has a winning strategy he has a positional one. Let t be a node of player 0 in G such that there is more than one edge exiting from t. We can partition the set

[3] An automaton is co-accessible iff from every state there is a path reaching an accepting state. It's easy to see that a minimized automaton is co-accessible.

of edges exiting from t in two disjoint non-empty sets E_α and E_β. Let G_α and G_β be the two subgraphs obtained from G by removing the edges of E_β and E_α, respectively. There are two cases to discuss. First, suppose that either in G_α or G_β player 0 has a winning strategy. Then, by inductive hypothesis he has a positional winning strategy. It is easy to see that such a strategy is winning in G too, indeed player 0 is able to play always in G_α or G_β since he controls every node.

Suppose now that player 0 has no winning strategy in G_α and in G_β. We prove the thesis by showing that player 0 has no winning strategy in G. Let M_α and M_β be the sets of all finite color sequences from t to t and K_α and K_β be the sets of all finite color sequences starting from t, in G_α and G_β, respectively. Such sets are regular languages: M_α and M_β are recognized by the automata having respectively G_α and G_β as state graphs, with starting node t and accepting set $\{t\}$. The sets K_α and K_β are the languages accepted by the automata with state graphs G_α and G_β, respectively, with starting node t and accepting set given by all the states.

Suppose now by contradiction that there exists a winning strategy for player 0 in G. Then this strategy will form a winning path π. Such a path cannot be in G_α or G_β, or else player 0 has a winning strategy in one of those subgraphs. So the path is in G and passes through t. Let x be the shortest prefix of π ending in t, then π belongs to the set $x\langle(M_\alpha \cup M_\beta)^*(K_\alpha \cup K_\beta)\rangle$, since it starts with x, then either loops forever from t to t in G_α and G_β, or possibly ends with an infinite path that never comes back to t. However, for $\gamma \in \{\alpha,\beta\}$, the sets $x\langle M_\gamma^*\rangle$ and $x\langle K_\gamma\rangle$ contain only paths in G_γ, so they are losing. Thus, we have $x\langle(M \cup N)^*K\rangle \not\leq x\langle M^*\rangle \cup x\langle N^*\rangle \cup x\langle K\rangle$, which contradicts selectivity.

□

After discussing monotonicity and selectivity we can formally complete the proof of Lemma 10.

Lemma 14. *Let $k = 1$, the goal $W = [k]^*1[k]^*1[k]^\omega$ is full positional.*

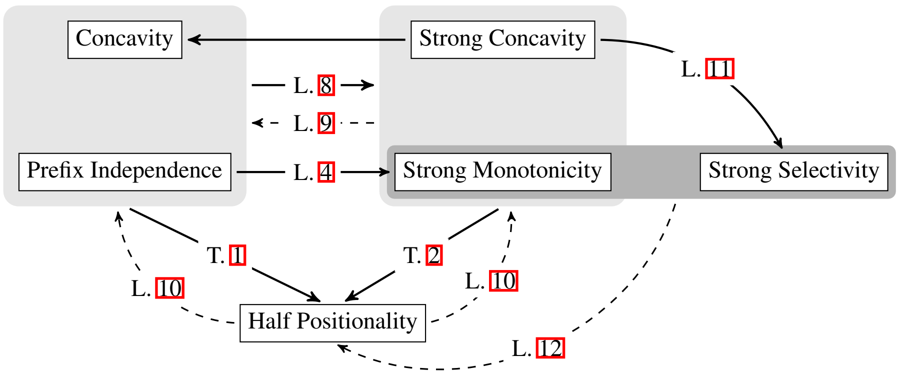


Fig. 2. Summary of results. Continuous arrows represent a holding implication and dashed ones a false one. Arrows are labeled with the corresponding lemma or theorem. Moreover, a gray box represents a conjunction of conditions.

Proof. Using the characterization of [GZ05], we prove the statement by showing that the goals W and $\overline{W}$ are selective and monotone. Observe that W is the set of all the words having at least two 1's and $\overline{W}$ is the set of all the words contain at most one 1.

1. W is selective. Suppose by contradiction that W is not selective. Then, there exist $x \in [k]^*$ and $M, N, K \subseteq [k]^*$ such that $x\langle (M \cup N)^* K \rangle = x\langle (M \cup N)^* \rangle \cup x(M \cup N)^* \langle K \rangle$ contains a winning word and $x\langle M^* \rangle \cup x\langle N^* \rangle \cup x\langle K \rangle$ contains only losing words. Observe that no word in M or N contains 1, or else if $m \in M \cup N$ contains a 1, $xm^\omega \in x\langle M^* \rangle \cup x\langle N^* \rangle$ contains infinitely many 1's and it is a winning word. So, the words in the set $x\langle (M \cup N)^* \rangle$ do not contain 1 and they are losing. Moreover, since $x\langle K \rangle$ does not contain more than one 1, the words in $x(M \cup N)^* \langle K \rangle$ do not contain more than one 1 and they are all losing too. So, the set $x\langle (M \cup N)^* K \rangle$ contains only losing words, hence a contradiction.
2. W is monotone. Suppose by contradiction that W is not monotone. Then there exist $x, y \in [k]^*$ and $M, N \subseteq [k]^*$ such that $xM < xN$ and $yN < yM$. So, xM and yN contain only losing words, xN and yM contain a winning word. If x contains more than one 1, all words in the first two sets are losing, hence a contradiction. If x contains one 1, then no word in M contains 1. However, there is a winning word in yM, so y contains two 1's. Hence, yN contain only winning words, which is a contradiction. If x does not contain a 1, there is a word in N with two 1's. Hence, yN contains at least a winning word, which is again a contradiction.
3. $\overline{W}$ is selective. Suppose by contradiction that $\overline{W}$ is not selective. Then, there exist $x \in [k]^*$ and $M, N, K \subseteq [k]^*$ such that $x\langle (M \cup N)^* K \rangle = x\langle (M \cup N)^* \rangle \cup x(M \cup N)^* \langle K \rangle$ contains a winning word and $x\langle M^* \rangle \cup x\langle N^* \rangle \cup x\langle K \rangle$ contains only losing words. Observe that no word in M or N does not contain 1, else if $m \in M \cup N$ does not contain a 1, $xm^\omega \in x\langle M^* \rangle \cup x\langle N^* \rangle$ does not contain 1's and it is a winning word. So the words in the set $x\langle (M \cup N)^* \rangle$ contain infinitely many 1's and they are losing. Moreover, since $x\langle K \rangle$ contains more than one 1, the words in $x(M \cup N)^* \langle K \rangle$ contain more than one 1 and they are all losing. So, the set $x\langle (M \cup N)^* K \rangle$ contains only losing words, hence a contradiction.
4. $\overline{W}$ is monotone. Suppose by contradiction that $\overline{W}$ is not monotone. Then there exist $x, y \in [k]^*$ and $M, N \subseteq [k]^*$ such that $xM < xN$ and $yN < yM$. So, xM and yN contain only losing words, xN and yM contain a winning word. If x contains more than one 1, all words in the first two sets are winning, hence a contradiction. If x contains one 1, then there is a word in N that does not contain 1's. Since yN contains only losing words, y contains more than one 1. So, all words in yM are losing, hence a contradiction. If x does not contain 1, then all words in M contain more than one 1, so all words in yM are losing, hence a contradiction. □

6 Conclusions

In this paper, we defined a new sufficient condition for half-positionality on finite arenas, which turns out to be strictly weaker (i.e., broader) than that defined by Kopczyński in [Kop06], as long as determined goals are considered. We discussed the conditions presented in [GZ05] for full-positionality and we proved that a stronger partial form of them does not ensure half positionality.

The main open problem left by this research is the formulation of a complete characterization of half-positionality. Another interesting question for further research is whether or not the properties of strong monotonicity and strong concavity imply determinacy. The answer to this question may simplify the statement of Theorem 2 by removing the hypothesis of determinacy. Finally, another open problem consists in developing algorithms for checking whether a goal, given in input in some effective way such as an automaton or a temporal logic formula, satisfies the conditions outlined in this paper and is therefore HP. Such an algorithm may be used as a preliminary step in controller synthesis tools, in order to estimate the amount of memory that the synthesized controller will need.

References

[Cac02] Cachat, T.: Symbolic strategy synthesis for games on pushdown graphs. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 704–715. Springer, Heidelberg (2002)

[CN06] Colcombet, T., Niwinski, D.: On the positional determinacy of edge-labeled games. Theor. Comput. Sci. 352(1-3), 190–196 (2006)

[EJ91] Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: FOCS 1991, pp. 368–377. IEEE Computer Society Press, Los Alamitos (1991)

[Gra04] Gradel, E.: Positional determinancy of infinite games. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 4–18. Springer, Heidelberg (2004)

[GZ05] Gimbert, H., Zielonka, W.: Games where you can play optimally without any memory. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 428–442. Springer, Heidelberg (2005)

[Kop06] Kopczyński, E.: Half-positional determinancy of infinite games. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 336–347. Springer, Heidelberg (2006)

[Kop07] Kopczyński, E.: Omega-regular half-positional winning conditions. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 41–53. Springer, Heidelberg (2007)

[KVW01] Kupferman, O., Vardi, M.Y., Wolper, P.: Module checking. Information and Computation 164, 322–344 (2001)

[McN93] McNaughton, R.: Infinite games played on finite graphs. Annals of Pure and Applied Logic 65, 149–184 (1993)

[Mos91] Mostowski, A.W.: Games with forbidden positions. Technical Report 78, Uniwersytet Gdański, Instytut Matematyki (1991)

[Tho95] Thomas, W.: On the synthesis of strategies in infinite games. In: Mayr, E.W., Puech, C. (eds.) STACS 1995. LNCS, vol. 900, pp. 1–13. Springer, Heidelberg (1995)

[Zie98] Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. J. of Theor. Comp. Sci. 200(1-2), 135–183 (1998)

Proof-Theoretic Analysis of Rationality for Strategic Games with Arbitrary Strategy Sets

Jonathan A. Zvesper[1] and Krzysztof R. Apt[2,3]

[1] Oxford University Computing Laboratory, Parks Road, Oxford OX1 3QD, UK
[2] Centre for Mathematics and Computer Science (CWI), Science Park 123, 1098 XG Amsterdam, The Netherlands
[3] University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands

Abstract. In the context of strategic games, we provide an axiomatic proof of the statement

(Imp) Common knowledge of rationality implies that the players will choose only strategies that survive the iterated elimination of strictly dominated strategies.

Rationality here means playing only strategies one believes to be best responses. This involves looking at two formal languages. One, $\mathcal{L}_O$, is first-order, and is used to formalise optimality conditions, like avoiding strictly dominated strategies, or playing a best response. The other, $\mathcal{L}_\nu$, is a modal fixpoint language with expressions for optimality, rationality and belief. Fixpoints are used to form expressions for common belief and for iterated elimination of non-optimal strategies.

Keywords: epistemic analysis, logic, fixpoints, rationalizability.

1 Introduction

There are two main sorts of solution concepts for strategic games: *equilibrium* concepts and what might be called "*effective*" concepts. One interpretation of the equilibrium concepts, for example Nash equilibrium, tacitly presupposes that a game is played repeatedly (see, e.g. [13, page 14]). Thus the standard condition for Nash equilibrium in terms of the knowledge or beliefs of the players [3] – the so-called "epistemic analysis" of Nash equilibrium – includes a requirement that players know the other players' strategy choices.

Consider the left-hand game in Figure 1, in which each player has two choices L and R and both players get payoff of 1 if they coordinate, and 0 otherwise. Then there are two Nash equilibria[1]: both play L or both play R. But this does not translate by itself into an effective strategy for either player reasoning in isolation, without some exogenous information.

In contrast, *effective* solution concepts, for example the iterated elimination of strictly dominated strategies, are compatible with such a "one-shot" interpretation of the game. Thus the epistemic analysis of the iterated elimination of

[1] A Nash equilibrium in a two-player game is a pair (s_1, s_2) of strategies, one for each player such that s_1 is a best response to s_2 and vice-versa.

	L	R
L	1, 1	0, 0
R	0, 0	1, 1

	L	R
U	1, 1	1, 0
D	0, 0	0, 1

Fig. 1. Two strategic games

strictly dominated strategies does not require that the players know each other's strategy choice.

A strategy s_i is strictly dominated if there is an alternative strategy t_i such that no matter what the opponent does, t_i is (strictly) better for i than s_i. Say that a player is *sd*-rational if he never plays a strategy that he believes to be strictly dominated. What the iterated elimination of strictly dominated strategies does in general require, see [4], is then that players have *common true belief* that each other is rational, that is: they are rational, believe that all are rational, believe that all believe that all are rational, etc.

In the right-hand game in Figure 1, the column player, on first looking at her choices L or R is, superficially, in the same situation as before: choose L and risk the opponent playing D or choice R and risk the opponent playing U. However, this time the row player can immediately dismiss playing D on the grounds that U will *always* be better, no matter what the column player does. So if the column player knows (or believes) this, then he cannot *rationally* play R, and so must play L.

In this paper we study the logical form of epistemic characterisation results of this second kind, so we give formal proof-theoretic principles to justify some given effective or algorithmic process in terms of common belief of some form of rationality. We will introduce two formal languages. One, $\mathcal{L}_O$, is a first-order language, that can be used to define 'optimality conditions'. Avoiding playing a strictly dominated strategy is an example of an 'optimality condition'. Another one is choosing a best response.

However, as observed in [2] for all such notions there are two versions: 'local' and 'global'. Notice that in our informal description of when s_i is strictly dominated by t_i we did not specify *where* i is allowed to choose alternative strategies from. In particular, since we are thinking of an iterated procedure, if t_i has been eliminated already then it would seem unreasonable to say that i should consider it. That intuition yields the *local* definition; the *global* definition states the opposite: that player i should always consider his original strategy set from the full game when looking to see if a strategy is dominated.

A motivation for looking at global versions of optimality notions is that they are often mathematically better behaved. On finite games the iterations for various local and global versions coincide [1], but on infinite games they can differ. In a nutshell: an optimality condition ϕ_i for player i is *global* if i does not 'forget', during the iterated elimination process, what strategies he has available in the whole game. The distinction is clarified in the respective definitions in $\mathcal{L}_O$.

An optimality condition ϕ induces an optimality operator O_ϕ on the complete lattice of restrictions (roughly: the subgames) of a given game. Eliminating non-ϕ-optimal strategies can be seen as the calculation of a fixpoint of the corresponding operator O_ϕ. Furthermore, common belief is characterised as a fixpoint (cf. Note 3 below). Viewed from the appropriate level of abstraction, in terms of fixpoints of operators, this connection between common belief of rationality and the iterated elimination of non-optimal strategies becomes clear.

We define a language $\mathcal{L}_\nu$ that describes things from this higher level of abstraction. Each optimality condition defines a corresponding notion of *rationality*, which means playing a strategy that one *believes* to be ϕ-optimal. $\mathcal{L}_\nu$ is a modal fixpoint language with modalities for *belief* and *optimality*, and so can express connections between optimality, rationality and (common) belief.

We say that an operator O on an arbitrary lattice $(D, \subseteq)$ is ***monotonic*** when for all $A, B \in D$, if $A \subseteq B$ then $O(A) \subseteq O(B)$. The global versions of relevant optimality operators, in particular of the operators corresponding to the best response and strict dominance, are monotonic. This is immediately verifiable in $\mathcal{L}_O$ by observing that the relevant definition is *positive*.

Our first result is a syntactic proof of the following result, where ϕ is a *monotonic* optimality condition:[2]

Theorem 1. *Common true belief of ϕ-rationality entails all played strategies survive the iterated elimination of non-ϕ-optimal strategies.*

Although this theorem relies on a rule for fixpoint calculi that is only sound for monotonic operators, the semantics of the language $\mathcal{L}_\nu$ allows also for arbitrary ***contracting*** operators, i.e. such that for all A, $O(A) \subseteq A$. We are therefore able to look at what more is needed in order to justify the following statement (cf. [4, Proposition 3.10]), where *gbr*-rationality means avoiding avoiding strategies one believes to be never best responses in the global sense:

Theorem 2. (**Imp**) *Common true belief of gbr-rationality implies that the players will choose only strategies that survive the iterated elimination of strictly dominated strategies.*

This theorem connects a global notion of *gbr*-rationality with a local one, referred to in the iterated elimination operator. Our language allows for arbitrary contracting operators, and their fixpoints to be formed, and we exhibit one sound rule connecting the resulting fixpoints with monotonic fixpoints.

Our theorems hold for arbitrary games, and the resulting potentially transfinite iterations of the elimination process. The syntactic approach clarifies the logical underpinnings of the epistemic analysis. It shows that the use of transfinite iterations can be naturally captured in $\mathcal{L}_\nu$, at least when the relevant operators are monotonic, by a single inference rule that involves greatest fixpoints.

The relevance of monotonicity in the context of epistemic analysis of finite strategic games has already been pointed out in [5], where the connection is

[2] By "common true belief" we mean a common belief that is correct. In particular, common knowledge entails common true belief.

also noted between the iterated elimination of non-optimal strategies and the calculation of the fixpoint of the corresponding operator.

To our knowledge, although several languages have been suggested for reasoning about strategic games (e.g. [7]), none use explicit fixpoints (except, as we mentioned, for some suggestions in [5]) and none use arbitrary optimality operators.

Therefore they are not appropriate for reasoning at the level of abstraction that we suggest when studying the epistemic foundations of these "effective" solution concepts. For example while [7, Section 13] does provide some analysis of the logical form of the argument that common knowledge of one kind of rationality implies not playing strategies that are strictly dominated, the fixpoint reasoning is done at the meta-level. What [7] provides is a proof schema, that shows how, for any finite game, and any natural number n, to give a proof that common knowledge of rationality entails not playing strategies that are eliminated in n rounds of elimination of non-optimal strategies.

The more general and elegant reasoning principle is captured by using fixpoint operators and optimality operators. Another important advantage to our approach is that we are not restricted in our analysis to finite games. This means in particular that our logical analysis covers the mixed extension of any finite game.

Our use of transfinite iterations is motivated by the original finding of [12], where a two-player game is constructed for which the ω_0 (the first infinite ordinal) and $\omega_0 + 1$ iterations of the rationalizability operator of [6] differ.

2 Games and the Language $\mathcal{L}_O$

A ***strategic game*** is a tuple $(T_1, \dots, T_n, <_1, \dots, <_n)$, where $\{1, \dots, n\}$ are the players and each T_i is player i's set of strategies, and $<_i$ is player i's preference relation, which is a total linear order over the set of ***strategy profiles*** $T = \prod_{i=1}^{n} T_i$. Note that we assume arbitrary games, rather than restricting to games in which T is finite. To depict games it is sometimes easier, as we did in Figure 1, to write down a number for the players' "payoffs", rather than just a preference ordering. We use some standard notation from game theory, writing s_{-i} for $(s_1, \dots s_{i-1}, s_{i+1}, \dots s_n)$ and (s_i, t_{-i}) for the strategy profile $(t_1, \dots t_{i-1}, s_i, t_{i+1}, \dots s_n)$, as well as S_{-i} for $\prod_{j \neq i} S_j$. A ***restriction*** of the game $(T_1, \dots, T_n, <_1, \dots, <_n)$ is a sequence $S = (S_1, \dots, S_n)$ with $S_i \subseteq T_i$ for all players i, i.e. a (possibly empty) subgame in which the payoff information is left out.

The language we use for specifying optimality conditions is a first-order language, with variables $V = \{x, y, z, \dots\}$, a monadic predicate C, a constant o and a family of n ternary relation symbols $\cdot \geq^i_\cdot \cdot$, where $i \in [1..n]$. So $\mathcal{L}_O$ is given by the following inductive definition:

$$\phi ::= C(a) \mid a \geq^i_c b \mid \neg\phi \mid \phi \wedge \phi \mid \exists x \phi,$$

where $i \in [1..n]$ and $\{a, b, c\} \subseteq V \cup \{o\}$.

We use the standard abbreviations $\rightarrow$ and $\vee$, further abbreviate $\neg\, a \geq^i_c b$ to $b >^i_c a$, $\forall x\phi$ to $\neg\exists x\neg\phi$, $\exists x(C(x) \wedge \phi)$ to $\exists x \in C\,\phi$, and $\forall x(C(x) \rightarrow \phi)$ to $\forall x \in C\,\phi$.

An *optimality model* (G, G', s) is a triple consisting of a strategic game $G = (T_1, \ldots, T_n, <_1, \ldots, <_n)$, a restriction G' of G, and a strategy profile $s \in T$. G will be used to interpret the predicate C, and s will be the interpretation of o. An *assignment* for (G, G', s) is a function α assigning a strategy profile in T to each variable, and s to o. The ternary satisfaction relation $\vDash$ between optimality models, assignments and formulas of $\mathcal{L}_O$ is defined inductively as follows, where α is an assignment for (G, G', s), and $\nvDash$ the complement of $\vDash$:

$$\begin{array}{lll}
(G, G', s) \vDash_\alpha C(x) & \Leftrightarrow & \forall i \in \{1, \ldots, n\},\ (\alpha(x))_i \in G'_i \\
(G, G', s) \vDash_\alpha x \geq^i_z y & \Leftrightarrow & (\alpha(x)_i, \alpha(z)_{-i}) \geq_i (\alpha(y)_i, \alpha(z)_{-i}) \\
(G, G', s) \vDash_\alpha \neg\phi & \Leftrightarrow & (G, G', s) \nvDash_\alpha \phi \\
(G, G', s) \vDash_\alpha \phi_1 \wedge \phi_2 & \Leftrightarrow & (G, G', s) \vDash_\alpha \phi_1 \text{ and } (G, G', s) \vDash_\alpha \phi_2 \\
(G, G', s) \vDash_\alpha \exists x\phi & \Leftrightarrow & \text{there is } \alpha' : (G, G', s) \vDash_{\alpha'} \phi \text{ and} \\
 & & \forall y \in V \text{ with } x \neq y,\ \alpha(y) = \alpha'(y)
\end{array}$$

If for any assignment α for G we have $(G, G', s) \vDash_\alpha \phi$ then we write $(G, G', s) \vDash \phi$. A variable x occurs ***free*** in ϕ if it is not under the scope of a quantifier $\exists x$; a formula is ***closed*** if it has no free variables.

An ***optimality condition*** for player i is a closed $\mathcal{L}_O$-formula in which all the occurrences of the atomic formulas $a \geq^j_c b$ are with j equal to i. Intuitively, an optimality condition ϕ_i for player i is a way of specifying what it means for i's strategy in o to be an 'OK' choice for i given that i's opponents will play according to C_{-i} and that i's alternatives are C_i.

In particular, we are interested in the following optimality conditions:

– $lsd_i := \forall y \in C\, \exists z \in C\, o \geq^i_z y$,
– $gsd_i := \forall y\, \exists z \in C\, o \geq^i_z y$,
– $gbr_i := \exists z \in C\, \forall y\, o \geq^i_z y$.

The optimality conditions listed define some fundamental notions from game theory: lsd_i says that o_i is not *locally* strictly dominated in the context of C; gsd_i says that o_i is not *globally* strictly dominated in the context of C; and gbr_i says that o_i is globally a best response in the context of C.

The distinction between local and global properties, studied further in [2], is clarified below. It important for us here because the global versions, in contrast to the local ones, satisfy a syntactic property to be defined shortly.

First, as an illustration of the difference between gbr_i and gsd_i, consider the game in Figure 2. Call that game H, with the row player 1 and the column player 2. Then we have

$$(H, (T_1, T_2), (D, R)) \vDash gsd_1,$$

but

$$(H, (T_1, T_2), (D, R)) \vDash \neg gbr_1.$$

	L	R
U	$2,1$	$0,0$
M	$0,1$	$2,0$
D	$1,0$	$1,2$

Fig. 2. An illustration of the difference between strict dominance and best response

The local notions are such that when the 'context' restriction C consists of a singleton strategy for a player i, then that strategy is locally optimal. So for example

$$(H, (\{U, M\}, \{R\}), (U, R)) \vDash lsd_2,$$

whereas

$$(H, (\{U, M\}, \{R\}), (U, R)) \vDash \neg gsd_2.$$

We say that an optimality condition ϕ_i is ***positive*** when any sub-formula of the form $C(z)$, with z any variable, occurs under the scope of an even number of negation signs ($\neg$). Note that both gbr_i and gsd_i are positive, while lsd_i is not. As we will see in a moment, positive optimality conditions induce monotonic optimality operators, and monotonicity will be the condition required of optimality operators in Theorem 1 relating common knowledge of ϕ-rationality with the iterated elimination of non-ϕ strategies.

3 Optimality Operators

Henceforth let $G = (T_1, \dots, T_n, <_1, \dots, <_n)$ be a fixed strategic game. Recall that a *restriction* of the game G is a sequence $S = (S_1, \dots, S_n)$ with $S_i \subseteq T_i$ for all players i. We will interpret optimality conditions as *operators* on the lattice of the restrictions of a game ordered by component-wise set inclusion:

$$(S_1, \dots, S_n) \subseteq (S'_1, \dots, S'_n) \text{ iff } S_i \subseteq S'_i \text{ for all } i \in [1..n].$$

Given a sequence ϕ giving an optimality condition ϕ_i for each player i, we introduce an ***optimality operator*** O_ϕ defined by

$$O_\phi(S) = \prod_{i=1}^{n} \{s_i \in S_i \mid \phi_i(s_i, S).\}$$

Consider now an operator O on an arbitrary complete lattice $(D, \subseteq)$ with largest element $\top$. We say that an element $S \in D$ is a ***fixpoint*** of O if $S = O(S)$ and a ***post-fixpoint*** of O if $S \subseteq O(S)$.

We define by transfinite induction a sequence of elements O^α of D, for all ordinals α:

- $O^0 := \top$,
- $O^{\alpha+1} := O(O^\alpha)$,
- for limit ordinals β, $O^\beta := \bigcap_{\alpha<\beta} O^\alpha$.

We call the least α such that $O^{\alpha+1} = O^{\alpha}$ the ***closure ordinal*** of O and denote it by α_O. We call then O^{α_O} the ***outcome of*** (iterating) O and write it alternatively as O^{∞}.

Not all operators have fixpoints, but the monotonic and contracting ones (already defined in the introduction) do:

Note 1. Consider an operator O on $(D, \subseteq)$.

(i) If O is contracting or monotonic, then it has an outcome, i.e., O^{∞} is well-defined.
(ii) The operator $\overline{O}$ defined by $\overline{O}(X) := O(X) \cap X$ is contracting.
(iii) If O is monotonic, then the outcomes of O and $\overline{O}$ coincide.

Proof. For (i), it is enough to know that for every set D there is an ordinal α such that there is no injective function from α to D.

Note that the operators O_ϕ are by definition contracting, and hence all have outcomes. Furthermore, it is straightforward to verify that if ϕ_i is positive for all players i, then O_ϕ is monotonic.

The following classic result due to [14] also forms the basis of the soundness of some part of the proof systems we consider.[3]

Tarski's Fixpoint Theorem. For every monotonic operator O on $(D, \subseteq)$

$$O^{\infty} = \nu O = \cup\{S \in D \mid S \subseteq O(S)\},$$

where νO is the largest fixpoint of O.

We shall need the following lemma, which is crucial in connecting iterations of arbitrary contracting operators with those of monotonic operators. It also ensures the soundness of one of the proof rules we will introduce.

Lemma 1. *Consider two operators O_1 and O_2 on $(D, \subseteq)$ such that*

– *for all $S \in D$, $O_1(S) \subseteq O_2(S)$,*
– *O_1 is monotonic.*

Then $O_1^{\infty} \subseteq \overline{O_2}^{\infty}$.

Proof. By Note 1(*i*) the outcomes of O_1 and $\overline{O_2}$ exist.

We prove now by transfinite induction that for all α

$$\overline{O_1}^{\alpha} \subseteq \overline{O_2}^{\alpha}$$

from which the claim follows, since by Note 1(*iii*) we have $O_1^{\infty} = \overline{O_2}^{\infty}$.

By the definition of the iterations we only need to consider the induction step for a successor ordinal. So suppose the claim holds for some α.

The second assumption implies that $\overline{O_1}$ is monotonic. We have the following string of inclusions and equalities, where the first inclusion holds by the induction hypothesis and monotonicity of $\overline{O_1}$ and the second one by the first assumption

$$\overline{O_1}^{\alpha+1} = \overline{O_1}(\overline{O_1}^{\alpha}) \subseteq \overline{O_1}(\overline{O_2}^{\alpha}) = O_1(\overline{O_2}^{\alpha}) \cap \overline{O_2}^{\alpha} \subseteq O_2(\overline{O_2}^{\alpha}) \cap \overline{O_2}^{\alpha} = \overline{O_2}^{\alpha+1}.$$

[3] We use here its 'dual' version in which the iterations start at the largest and not at the least element of a complete lattice.

4 Beliefs and the Modal Fixpoint Language $\mathcal{L}_\nu$

Recall that G is a game $(T_1, \ldots, T_n, P_1, \ldots, P_n)$. A ***belief model*** for G is a tuple $(\Omega, \overline{s}_1, \ldots, \overline{s}_n, P_1, \ldots, P_n)$, with Ω a non-empty set of 'states', and for each player i, $\overline{s}_i : \Omega \to T_i$ and $P_i : \Omega \to 2^\Omega$. The P_i's are *possibility correspondences* cf. [4]. The idea of a possibility correspondence P_i is that if the actual state is ω then $P_i(\omega)$ is the set of states that i considers possible: those that i considers might be the actual state.

Subsets of Ω are called ***events***. A player i *believes* an event E if that event holds in every state that i considers possible. Thus at the state ω, player i believes E iff $P_i(\omega) \subseteq E$.

Given some event E we write G_E to denote the restriction of G determined by E:

$$(G_E)_i = \{s_i \in T_i \mid \exists u \in E : \overline{s}_i(u) = s_i\}.$$

In the rest of this section we present a formal language $\mathcal{L}_\nu$ that will be interpreted over belief models. To begin, we consider the simpler language $\mathcal{L}$, the formulas of which are defined inductively as follows, where $i \in [1..n]$:

$$\psi ::= rat_{\phi_i} \mid \psi \wedge \psi \mid \neg\psi \mid \Box_i \psi \mid O_{\phi_i}\psi,$$

with ϕ_i an optimality condition for player i. We abbreviate the formula $\bigwedge_{i\in[1..n]} rat_{\phi_i}$ to rat_ϕ, $\bigwedge_{i\in[1..n]} \Box_i\psi$ to $\Box\psi$ and $\bigwedge_{i\in[1..n]} O_{\phi_i}\psi$ to $O_\phi\psi$.

Formulas of $\mathcal{L}$ are interpreted as events in (i.e. as subsets of the domain of) belief models. Given a belief model $(\Omega, \overline{s}_1, \ldots, \overline{s}_n, P_1, \ldots, P_n)$ for G, we define the ***interpretation function*** $[\![\cdot]\!] : \mathcal{L} \to \mathcal{P}(\Omega)$ as follows:

– $[\![rat_{\phi_i}]\!] = \{\omega \in \Omega \mid \phi_i(\overline{s}_i(\omega), G_{P_i(\omega)})\}$,
– $[\![\phi \wedge \psi]\!] = [\![\phi]\!] \cap [\![\psi]\!]$,
– $[\![\neg\psi]\!] = \Omega - [\![\psi]\!]$,
– $[\![\Box_i\psi]\!] = \{\omega \in \Omega \mid P_i(\omega) \subseteq [\![\psi]\!]\}$,
– $[\![O_{\phi_i}\psi]\!] = \{\omega \in \Omega \mid (G, G_{[\![\psi]\!]}, \overline{s}_i(\omega)) \vDash \phi_i\}$.

$P_i(\omega)$ gives the set of states that i considers possible at ω, so $[\![rat_{\phi_i}]\!]$ is the event that player i is ϕ_i-rational, since it means that i's strategy is optimal according to ϕ_i in the context that the player considers it possible that he is in. The semantic clause for $\Box_i$ was mentioned at the begin of this section and is familiar from epistemic logic: $[\![\Box_i\psi]\!]$ is the event that player i believes the event $[\![\psi]\!]$. $[\![O_{\phi_i}\psi]\!]$ is the event that player i's strategy is optimal according to the optimality condition ϕ_i, in the context of the restriction $G_{[\![\psi]\!]}$.

Then clearly $[\![rat_\phi]\!]$ is the event that every player i is ϕ_i-rational; $[\![O_\phi\psi]\!]$ is the event that every player's strategy is ϕ_i-optimal in the context of the restriction $G_{[\![\psi]\!]}$; and $[\![\Box\psi]\!]$ is the event that every player believes the event $[\![\psi]\!]$ to hold.

Although $\mathcal{L}$ can express some connections between our formal definitions of optimality rationality and beliefs, it could be made more expressive. The language could be extended with, for example, atoms s_i expressing the event that the strategy s_i is chosen. This choice is made for example in [7], where modal languages for reasoning about games are defined. The language we introduce is not parametrised by the game, and consequently can unproblematically be used to reason about games with arbitrary strategy sets.

We will use our language to talk about fixpoint notions: common belief and iterated elimination of non-optimal strategies. Let us therefore explain what is meant by ***common belief***. Common belief of an event E is the event that all players believe E, all players believe that they believe E, all players believe that they believe that…, and so on. Formally, we define $\mathcal{CB}(E)$, the event that E is commonly believed, inductively:

$$\begin{aligned}\mathcal{B}_1(E) &= \{\omega \in \Omega \mid \forall i \in [1..n],\, P_i(\omega) \subseteq E\}\\ \mathcal{B}_{m+1}(E) &= \mathcal{B}_1(\mathcal{B}_m(E))\\ \mathcal{CB}(E) &= \bigcap_{m>0} \mathcal{B}_m(E)\end{aligned}$$

Notice that $\mathcal{B}_1(E)$ is the event that everybody believes that E (indeed, we have $\mathcal{B}_1[\![\psi]\!] = [\![\Box\psi]\!]$), $\mathcal{B}_2(E)$ is the event that everybody believes that everybody believes that E, etc.

'Common belief' is called 'common knowledge' when for all players i and all states $\omega \in \Omega$, we have $\omega \in P_i(\omega)$. In such a case the players have never ruled out the current state, and so it is legitimate to interpret $\Box_i\psi$ as 'i knows that ψ'.

Both common knowledge and common belief are known to have equivalent characterisations as fixpoints, and we will exploit this below in defining them in the modal fixpoint language which we now specify.

We extend the vocabulary of $\mathcal{L}$ with a single set variable denoted by X and the contracting fixpoint operator νX. (The corresponding extension of first-order logic by the dual, inflationary fixpoint operator μX was first studied in [8].) Modulo one caveat the resulting language $\mathcal{L}_\nu$ is defined as follows:

$$\psi ::= rat_{\phi_i} \mid (\psi \wedge \psi) \mid \neg\psi \mid \Box_i\psi \mid O_{\phi_i}\psi \mid \nu X.\psi$$

The caveat is the following:

– ϕ must be ***ν-free***, which means that it does not contain any occurrences of the νX operator.

This restriction is not necessary but simplifies matters and is sufficient for our considerations.

To extend the interpretation function $[\![\cdot]\!]$ to $\mathcal{L}_\nu$, we must keep track of the variable X. Therefore we first extend the function $[\![\cdot]\!] : \mathcal{L} \to \mathcal{P}(\Omega)$ to a function $[\![\cdot \mid \cdot]\!] : \mathcal{L}_\nu \times \mathcal{P}(\Omega) \to \mathcal{P}(\Omega)$ by padding it with a dummy argument. We give one clause as an example:

– $[\![\Box_i\psi \mid E]\!] = \{\omega \in \Omega \mid P_i(\omega) \subseteq [\![\psi \mid E]\!]\}$.

We use this extra argument in the semantic clause for the variable X:

– $[\![X \mid E]\!] = E$.

Those formulas whose semantics we have so far given define operators. More specifically, for each of them $[\![\psi \mid \cdot]\!]$ is an operator on the powerset $\mathcal{P}(\Omega)$ of Ω. We use this to define the clause for νX:

– $[\![\nu X.\psi \mid E]\!] = ([\![\psi \wedge X \mid \cdot]\!])^{\infty}$.

When X does not occur free in ψ, we have $[\![\psi \mid E]\!] = [\![\psi \mid F]\!]$ for any events E and F, so in these cases we can write simply $[\![\psi]\!]$. Note that $[\![\nu X.\psi]\!]$ is well-defined since for all E we have $[\![\psi \wedge X \mid E]\!] = [\![\psi \mid E]\!] \cap [\![X \mid E]\!] \subseteq E$, so the operator $[\![\psi \wedge X \mid \cdot]\!]$ is contracting.

We say that a formula ψ of $\mathcal{L}_\nu$ is ***positive in*** X when each occurrence of X in ψ is under the scope of an even number of negation signs ($\neg$), and under the scope of an optimality operator O_{ϕ_i} only if ϕ_i is positive.

Note 2. When ψ is positive, the operator $[\![\psi \mid \cdot]\!]$ is monotonic.

Then by Tarski's Fixpoint Theorem and Note 1(*iii*) we can use the following alternative definition of $[\![\nu X.\psi]\!]$ in terms of post-fixpoints:

$$[\![\nu X.\psi]\!] = \bigcup \{E \subseteq \Omega \mid E \subseteq [\![\psi \mid E]\!]\}.$$

Let us mention some properties the language $\mathcal{L}_\nu$ can express. First notice that common belief is definable in $\mathcal{L}_\nu$ using the νX operator. An analogous characterization of common knowledge is in [9, Section 11.5].

Note 3. Let ψ be a formula of $\mathcal{L}$. Then $[\![\nu X.\Box(X \wedge \psi)]\!]$ is the event that the event $[\![\psi]\!]$ is common belief.

From now on we abbreviate the formula $\nu X.\Box(X \wedge \psi)$ with ψ a formula of $\mathcal{L}$ to $\Box^*\psi$. So $\mathcal{L}_\nu$ can define common belief. Moreover, as the following observation shows, it can also define the iterated elimination of non-optimal strategies.

Note 4. In the game determined by the event $[\![\nu X.O_\phi X]\!]$, every player selects a strategy which survives the iterated elimination of non-ϕ-optimal strategies.

Proof. It follows immediately from the following equivalence, which is obtained by unpacking the relevant definitions:

$$G_{[\![O_\phi X \wedge X \mid E]\!]} = O_\phi(G_E).$$

5 Proof Systems

Consider the following formula:

$$(rat_\phi \wedge \Box^* rat_\phi) \rightarrow \nu X.O_\phi X. \qquad (1)$$

By Notes 3 and 4, we see that (1) states that: true common belief that the players are ϕ-rational entails that each player selects a strategy that survives the iterated elimination of non-ϕ-optimal strategies.

In the rest of this section we will discuss a simple proof system in which we can derive (1). We will use an axiom and rule of inference for the fixpoint operator taken from [11] and one axiom for rationality analogous to the one called in [7] an "implicit definition" of rationality. We give these in Figure 3, where, crucially, *ψ is positive in X, and all the ϕ_i's are positive.* We denote here by $\psi[X \mapsto \chi]$ the formula obtained from ψ by substituting each occurrence of the variable X with the formula χ. Assuming given some standard proof rules for propositional reasoning, we add the axioms and rule given in Figure 3 to obtain the system **P**.

Axiom schemata

$$rat_\phi \to (\Box\chi \to O_\phi\chi) \quad ratDis$$

$$\nu X.\psi \to \psi[X \mapsto \nu X.\psi] \; \nu Dis$$

Rule of inference

$$\frac{\chi \to \psi[X \mapsto \chi]}{\chi \to \nu X.\psi} \; \nu Ind$$

Fig. 3. Proof system **P**

A formula is a ***theorem*** of a proof system if it is derivable from the axioms and rules of inference. An $\mathcal{L}_\nu$-formula ψ is ***valid*** if for every belief model $(\Omega, \ldots)$ for G we have $[\![\psi]\!] = \Omega$. We now establish the soundness of the proof system **P**, that is, that its theorems are valid.

Lemma 2. *The proof system **P** is sound.*

Proof. We show the validity of the axiom *ratDis*:

Let $(\Omega, \overline{s}_1, \ldots, \overline{s}_n, P_i, \ldots, P_n)$ be a belief model for G. We must show that $[\![rat_\phi \to (\Box\chi \to O_\phi\chi)]\!] = \Omega$. That is, that for any χ the inclusion $[\![rat_\phi]\!] \cap [\![\Box\chi]\!] \subseteq [\![O_\phi\chi]\!]$ holds. So take some $\omega \in [\![rat_\phi]\!] \cap [\![\Box\chi]\!]$. Then for every $i \in [1..n]$, $\phi_i(\overline{s}_i(\omega), G_{P_i(\omega)})$, and $P_i(\omega) \subseteq [\![\chi]\!]$. So by monotonicity of ϕ_i, $\phi_i(\overline{s}_i(\omega), G_{[\![\chi]\!]})$, i.e. $\omega \in [\![O_{\phi_i}\chi]\!]$ as required.

The axioms νDis and the rule νInd were introduced in [11]; they formalise, respectively, the following two consequences of Tarski's Fixpoint Theorem concerning a monotonic operator F:

– νF is a post-fixpoint of F, i.e., $\nu F \subseteq F(\nu F)$ holds,
– if Y is a post-fixpoint of F, i.e., $Y \subseteq F(Y)$, then $Y \subseteq \nu F$.

Next, we establish the already announced claim.

Theorem 1. *The formula (1) is a theorem of the proof system **P**.*

Proof. The following formulas are instances of the axioms *ratDis* (with $\psi := \Box^* rat_\phi \wedge rat_\phi$) and νDis (with $\psi := \Box(X \wedge rat_\phi)$) respectively:

$$rat_\phi \to (\Box(\Box^* rat_\phi \wedge rat_\phi) \to O_\phi(\Box^* rat_\phi \wedge rat_\phi)), \tag{2}$$

$$\Box^* rat_\phi \to \Box((\Box^* rat_\phi) \wedge rat_\phi). \tag{3}$$

Putting these two together via some propositional logic, we obtain

$$((\Box^* rat_\phi) \wedge rat_\phi) \to O_\phi((\Box^* rat_\phi) \wedge rat_\phi),$$

which is of the right shape to apply the rule νInd (with $\chi := \Box^* rat_\phi \wedge rat_\phi$ and $\psi := O_\phi X$). We then obtain

$$(\Box^* rat_\phi \wedge rat_\phi) \to \nu X.O_\phi X,$$

which is precisely the formula (1).

Corollary 1. *The formula (1) is valid.*

It is interesting to note that no axioms or rules for the modalities $\Box$ or O were needed in order to derive (1), other than those connecting them with rationality. In particular, no introspection is required on the part of the players, nor indeed is the K axiom $\Box(\varphi \wedge \psi) \leftrightarrow (\Box\varphi \wedge \Box\psi)$ needed.

In the language $\mathcal{L}_\nu$, the rat_{ϕ_i} are in effect propositional constants. We might instead define them in terms of the $\Box_i$ and O_{ϕ_i} modalities but to this end we would need to extend the language $\mathcal{L}_\nu$. One way to do this is to use a quantified modal language, allowing quantifiers over set variables, so extending $\mathcal{L}_\nu$ by allowing formulas of the form $\forall X \varphi$. Such quantified modal logics are studied in [10]. It is straightforward to extend the semantics to this larger class of formulas:

$$[\![\forall X \varphi \mid E]\!] = \{\omega \in \Omega \mid \forall F \subseteq \Omega,\, \omega \in [\![\varphi \mid F]\!]\}.$$

In the resulting language each rat_{ϕ_i} constant is definable by a formula of this second-order language:

$$rat_{\phi_i} \equiv \forall X(\Box_i X \to O_{\phi_i} X). \tag{4}$$

The following observation then shows correctness of this definition.

Note 5. For all $i \in [1..n]$ the formula (4) is valid in the semantics sketched.

To complete our proof-theoretic analysis we augment the proof system **P** with the following proof rule where we assume that χ is positive in X, but where ψ is an arbitrary ν-free $\mathcal{L}_\nu$-formula:

$$\frac{\chi \to \psi}{\nu X.\chi \to \nu X.\psi}\ Incl$$

The soundness of this rule is a direct consequence of Lemma 1.

To formalize the statement **Imp** we need two optimality conditions, gbr_i and lsd_i.

To link the proof systems for the languages $\mathcal{L}_O$ and $\mathcal{L}_\nu$ we add the following proof rule, where each ϕ_i and ψ_i is an optimality condition in $\mathcal{L}_O$, and $O_\phi X \to O_\psi X$ is a formula of $\mathcal{L}_\nu$.

$$\frac{\phi_i \to \psi_i, i \in [1..n]}{O_\phi X \to O_\psi X}\ Link$$

The soundness of this rule is a direct consequence of the semantics of the formulas $O_\phi X$ and $O_\psi X$.

We denote the system obtained from **P** by adding to it the above two proof rules and standard first-order logic rules concerning the formulas in the language $\mathcal{L}_O$, like

$$\frac{\exists y\, \forall x \phi}{\forall x\, \exists y \phi}$$

by **R**. We can now formalize the statement **Imp** as follows:

$$(rat_{gbr} \wedge \Box^* rat_{gbr}) \to \nu x.O_{lsd}x. \tag{5}$$

The following result then shows that this formula can be formally derived in the considered proof system.

Theorem 2. *The formula (5) is a theorem of the proof system **R**.*

Proof. The properties gbr_i are monotonic, so the following implication is an instance of (1):

$$(rat_{gbr} \wedge \Box^* rat_{gbr}) \to \nu x.O_{gbr}x.$$

Further, since the implication $gbr_i \to lsd_i$ holds, we get by the *Link* rule

$$\nu x.O_{gbr}x \to \nu x.O_{lsd}x,$$

from which (5) follows.

Corollary 2. *The formula (5) is valid.*

6 Summary

We have studied the logical form of epistemic characterisation results, for arbitrary (including infinite) strategic games, of the form "common knowledge of ϕ-rationality entails playing according to the iterated elimination of non-ϕ' properties". A main contribution of this work is in revealing, by giving syntactic proofs, the reasoning principles involved in two cases: firstly when $\phi = \phi'$ (Theorem 1), and secondly when ϕ entails ϕ' (Theorem 2). In each case the result holds when ϕ is monotonic. The language $\mathcal{L}_\nu$ that we used to formalise this reasoning is to our knowledge novel in combining optimality operators with fixpoint notions. Such a combination is natural when studying such characterisation results, since common knowledge and iterated elimination are both fixpoint notions.

The language $\mathcal{L}_\nu$ is parametric in the optimality conditions used by players. It is therefore built on the top of a first-order language $\mathcal{L}_O$ used to define syntactically optimality conditions relevant for our analysis.

References

1. Apt, K.R.: Relative strength of strategy elimination procedures. Economics Bulletin 3(21), 1–9 (2007), `http://economicsbulletin.vanderbilt.edu/Abstract.asp?PaperID=EB-07C70015`
2. Apt, K.R.: The many faces of rationalizability. Berkeley Electronic Journal of Theoretical Economics 7(1), 38 pages (2007)
3. Aumann, R.J., Brandenburger, A.: Epistemic conditions for nash equilibrium. Econometrica 63(5), 1161–1180 (1995)
4. Battigalli, P., Bonanno, G.: Recent results on belief, knowledge and the epistemic foundations of game theory. Research in Economics 53, 149–225 (1999)
5. Benthem, J.v.: Rational dynamics and epistemic logic in games. International Game Theory Review 9(1), 13–45 (2007) (Erratum reprint, 9(2), 377–409)
6. Bernheim, B.D.: Rationalizable strategic behavior. Econometrica 52, 1007–1028 (1984)
7. Bruin, B.d.: Explaining Games: On the logic of game theoretic explanations. PhD thesis, ILLC, Amsterdam (2004)
8. Dawar, A., Grädel, E., Kreutzer, S.: Inflationary fixed points in modal logics. ACM Transactions on Computational Logic, TOCL 5(2), 282–315 (2004)
9. Fagin, R., Halpern, J.Y., Vardi, M., Moses, Y.: Reasoning about knowledge. MIT Press, Cambridge (1995)
10. Fine, K.: Propositional quantifiers in modal logic. Theoria 36, 336–346 (1970)
11. Kozen, D.: Results on the propositional mu-calculus. Theoretical Computer Science 27(3), 333–354 (1983)
12. Lipman, B.L.: A note on the implications of common knowledge of rationality. Games and Economic Behaviour 6, 114–129 (1994)
13. Osborne, M.J., Rubinstein, A.: A Course in Game Theory. MIT Press, Cambridge (1994)
14. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5, 285–309 (1955)

An Agent Language with Destructive Assignment and Model-Theoretic Semantics

Robert Kowalski and Fariba Sadri

Department of Computing, Imperial College London, 180 Queens Gate, London SW7 2AZ
{rak,fs}@doc.ic.ac.uk

Abstract. In this paper we present an agent language that combines agent functionality with an action theory and model-theoretic semantics. The language is based on abductive logic programming (ALP), but employs a simplified state-free syntax, with an operational semantics that uses destructive assignment to manipulate a database, which represents the current state of the environment. The language builds upon the ALP combination of logic programs, to represent an agent's beliefs, and integrity constraints, to represent the agent's goals. Logic programs are used to define macro-actions, intensional predicates, and plans to reduce goals to sub-goals including actions. Integrity constraints are used to represent reactive rules, which are triggered by the current state of the database and recent agent actions and external events. The execution of actions and the assimilation of observations generate a sequence of database states. In the case of the successful solution of all goals, this sequence, taken as a whole, determines a model that makes the agent's goals and beliefs all *true*.

Keywords: abductive logic programming, agent languages, model-theoretic semantics.

1 Introduction

Practical agent languages, many of which were originally inspired by the use of modal logic specifications of an agent's Beliefs, Desires and Intentions, have largely abandoned their original model-theoretic semantics in favour of operational semantics. They employ procedural representations and perform destructive assignment on "beliefs" that represent the current state of the agent's environment.

ALP (abductive logic programming) agents [12] have both an operational semantics and a model-theoretic semantics. However, they represent the agent's observations in a non-destructive database and explicitly represent and reason about time or state, using a formal action theory such as the event calculus, with the consequent inefficiencies of reasoning with explicit frame axioms.

In this paper we present a language, LPS, that combines a declarative semantics based on ALP with the features of practical agent languages, including the use of destructive assignment and a syntax that does not refer to time or state. The semantics of LPS can be viewed in terms of Kripke possible world structures, as in Transaction (*TR*) Logic [2]. However in *TR* Logic, the truth of sentences is defined along paths of possible worlds. In LPS, the possible worlds are combined into a single model with state arguments in the spirit of the situation calculus and Golog [14].

The database is structured as a deductive database, with extensional predicates that are represented explicitly and with intentional predicates that are defined by logic programs. Actions and observations are structured by means of an action theory that defines the preconditions of actions and the effects of actions and external events on the extensional predicates of the database. Intentional predicates are modified as ramifications of changes to the extensional predicates.

The frame problem is avoided by employing destructive change of state, without the use of frame axioms. The inefficiencies of planning from first principles are avoided, by using plan libraries to achieve intended consequences of actions, and by using the action theory only to transform one state of the database to the next, implementing consequences of the agent's actions and external observations.

In contrast with many agent languages, *TR* Logic and Golog, but as in ALP agents, LPS highlights the distinction between maintenance goals (or reactive rules), represented as integrity constraints, and beliefs, represented as logic programs. The approach is based upon our earlier attempt to combine similar features of production systems with the model-theory of ALP [12]. We retain the name LPS, introduced in an earlier paper [13], and which stands for Logic-based Production System language, because we treat production rules and agent plans in the same way.

In the remainder of the paper, we present motivating examples and background, and then the syntax, operational semantics and model-theoretic semantics of LPS. We assume the reader is familiar with logic programs, SLD resolution and the minimal model semantics of Horn clauses.

1. 1 Motivating Examples

The vocabulary of LPS includes both ordinary stateless predicates, as well as *fluents*, which are extensional and intensional predicates, and *actions*, which are atomic actions, macro-actions, and external events, which are observable by the agent. The semantics (or internal syntax) of a fluent P has an additional argument $P(T)$, indicating that P holds in the state T (or at the time T). Atomic and macro actions A have two additional arguments $A(T_1, T_2)$, indicating that the action A takes place from T_1 to T_2. The semantics of atomic actions and events A happening in the transition from state T to $T+1$ is given by $A(T, T+1)$.

The surface syntax of LPS does not have explicit state arguments. Instead, as well as the ordinary conjunction $\wedge$, it has two other conjunctions whose meaning is defined in terms of states. The formal syntax and semantics will be given in Section 3. But, in the meanwhile, the semantics of the two conjunctions in the following examples can be understood as follows:

$P : Q$, where both P and Q are fluents, means $P(T) \wedge Q(T)$.
$P : A$, where P is a fluent and A is an action means $P(T_1) \wedge A(T_1, T_2)$.
$A : P$, where A is an action and P is a fluent means $A(T_1, T_2) \wedge P(T_2)$.
$P ; Q$, where both P and Q are fluents, means $P(T_1) \wedge Q(T2) \wedge T_1 \leq T_2$.
$P ; A$, where P is a fluent and A is an action means $P(T_1) \wedge A(T_2, T_3) \wedge T_1 \leq T_2$.
$A ; B$, where both A and B are actions, means $A(T_1, T_2) \wedge B(T_3, T_4) \wedge T_2 \leq T_3$.

Below we illustrate our approach by giving examples formalized in the LPS language.

Example 1: We consider an online shopping scenario, similar to the running example produced by the W3C RIF Working Group on rule interchange[1]. Reactive rules are used to welcome a customer when she logs in, and to take payment and issue confirmation when she checks out:

login(X) : customer(X) → welcome(X).
checkout(X): customer(X): shop-cart(X, ID, Value) →
take-payment(X, ID, Value) ; confirm(X, ID, Value).

The goals generated by the reactive rules are solved by macro-actions, in which a customer is *welcomed* with an appropriate offer. A new customer is welcomed by an offer of a promotional item. A gold customer is welcomed by an offer of a promotional item that is similar to an item recommended by her profile:

welcome(X) ← status(X, new): promotional-item(Y): offer(X, Y).
welcome(X) ← status(X, gold): promotional-item(Y): profile(X, Z):
similar(Y, Z): offer(X, Y).

The semantics (i.e. the state-based translation) of the first reactive rule and the first macro-action definition are:

$login(X, T\text{-}1, T) \wedge customer(X, T) \rightarrow welcome(X, T1, T2) \wedge T \le T1.$
$welcome(X, T, T1) \leftarrow status(X, new, T) \wedge promotional\text{-}item(Y, T) \wedge$
$offer(X, Y, T, T1).$

Example 2: The following is a reformulation in LPS of an example given in [4].
Reactive Rule: If a room is dirty clean it
is-dirty(Room) → clean(Room).
Macro-actions definitions:
clean(Room) ← goto(Room); vacuum(Room).
goto(Y) ← pos(Y).
goto(Y) ← pos(X) : different(X, Y) : adjacent(X, Z): step(X, Z); goto(Y).

Here *is-dirty* and *pos* are extensional predicates, *adjacent* and *different* are state-independent predicates, *vacuum* and *step* are atomic actions, and *clean* and *goto* are macro-actions. The action *step(X,Y)* causes a change in location and the action *vacuum(Room)* causes a change in the status of the *Room* via the *action theory*:

terminates(step(X,Y), pos(X)) and *initiates(step(X,Y), pos(Y)),*
terminates(vacuum(Room), is-dirty(Room)) and
initiates(vacuum(Room), is-clean(Room)).

The semantics of the last macro-action definition is:

$goto(Y, T1, T3) \leftarrow pos(X, T1) \wedge different(X, Y) \wedge adjacent(X,Z) \wedge step(X, Z, T1, T1+1) \wedge goto(Y, T2, T3) \wedge T1+1 \le T2.$

[1] http://www.w3.org/2005/rules/wiki/RIF_Working_Group visited in July 2009.

The LPS operational semantics (the LPS cycle) works as follows: The condition *is-dirty(Room)* of the reactive rule is checked against a database that represents the current state of the environment. For all instantiations σ for which the condition is true, the goal *clean(Room)*σ is added to the agent's goals. Each goal is then planned for by planning rules or macro-actions, the resulting atomic actions are executed, and each such execution (destructively) updates the database. In general, the planning and action executions can be interleaved, provided any ordering dictated by the connectives is respected. The predicate *pos* acts as a *guard* in the last two macro-action clauses, checking the agent's current location and directing the agent towards the next action. If all the goals are successfully planned for and the atomic actions are successfully executed the agent would have traversed a sequence of states the totality of which corresponds to a (minimum) model in which the reactive rule is true.

Example 3: The following is a reformulation in LPS of another example in [4], which involves buying a gift. According to the scenario in [4], first the agent checks what gifts are available in Harrods, and forms a plan to go to Harrods and purchase the gift. Then for some reason this plan does not succeed and a special plan revision rule changes the plan to purchasing that same gift from Dell. In LPS the beliefs required for this scenario can be formalized without plan revision rules, as follows:

Planning rule: *have(Gift) ← sells(harrods, Gift): buy(Gift).*
Macro-action definitions:
buy(Gift) ← goto(harrods); purchase(Gift, harrods).
buy(Gift)←online(Store):sells(Store,Gift):goOnline(Store);purchase(Gift, Store).
The database contains the fact: *online(dell).*

The LPS operational semantics is neutral with respect to the search strategy used to explore the search space, and the "conflict resolution strategy" used to select an action to execute. To obtain the behavior of the scenario described in [4], these strategies would need to try the macro-action rules in the order in which they are written, try the first action (*goto(harrods))*, and if it fails, either re-attempt the action later or execute the alternative action (*goOnline(dell)).*

Here is an alternative, more flexible formalization using only planning rules:
have(Gift)←is-Store(Store): sells(Gift, Store): goto(Store); purchase(Gift,Store).
have(Gift)←online(Store):sells(Gift,Store):goOnline(Store);
purchase(Gift,Store).

2 Background

2.1 Informal Comparison of Agent Languages and LPS

Practical agent languages can be regarded as an extension of production systems, in which condition-action rules are generalised to condition-action-and-goal rules. Both production systems and agent languages manipulate a database of facts or *beliefs*, which represents the current state of the environment. The database is updated destructively both by the agent's observations and by the agent's actions. The agent's goals are represented either as goal facts in the database, or in a separate stack of goals and actions, which represents the agent's intentions.

Like condition-action rules in production systems, condition-action-and-goal rules, called *plans* in agent languages, provide two main functions. Arguably their primary function is as *reactive rules*, to react to changes in the database, verifying that the condition holds and adding the corresponding goals and actions either to the database or the stack of goals. However, in practice they often function as *goal-reduction rules*, to match a current goal with one of the conditions of a plan, verify the other conditions of the plan, and add the corresponding goals and actions to the database or stack of intentions.

LPS borrows from production systems and agent languages their state-free syntax and their destructively changing database. It uses the database to represent the current state of the environment, and represents goals (or alternative candidate intentions) as a set of goal clauses, executing them as in SLD resolution. The search strategy and selection function can treat the set as a stack in the same way that Prolog implements a restricted version of SLD resolution. Alternatively, it can use the selection function more freely to interleave planning with plan execution.

The main difference between LPS and more conventional agent languages is that LPS interprets and represents reactive plans and goal-reduction plans differently, and this difference is exploited to provide LPS with a model-theoretic semantics. It interprets goal-reduction plans as *beliefs* and represents them as logic programs. It provides them with a backward reasoning operational semantics and a minimal model declarative semantics. It interprets reactive plans as *(maintenance) goals* (or *policies*) and represents them as integrity constraints (as in abductive logic programming). It provides them with a forward reasoning operational semantics and the model-theoretic semantics of integrity constraints.

Production systems and agent languages typically represent actions performed on the internal database as additions or deletions of facts in the database. LPS employs a more structured representation of actions in the tradition of the situation calculus and event calculus. Additions and deletions are not explicit actions, but are consequences of an action theory. It uses destructive change of state to deal with the computational aspects of the frame problem.

In production systems and agent languages, when the conditions of more than one condition-conclusion rule hold, a choice needs to be made between the different conclusions. In production systems, this is made by means of a conflict resolution strategy. In agent languages, it is made by selecting one of the conclusions as an intention, and possibly repairing the resulting plan if it fails. In ALP and LPS, when the rules are interpreted as beliefs represented as logic programming clauses, the choice is dealt with by the selection function and search strategy. When the rules are interpeted as maintenance goals represented by integrity constraints, ***all*** maintence goals must be made true, by making their conclusions true whenever their conditions are true.

However in LPS, an analogue of conflict resolution is performed when the agent decides which action to execute. In ALP agents, we have explored the use of Decision Theory for this purpose. However, in LPS we assume that the choice is made by the selection and search strategies, subject to the constraint that no action is selected if there are other actions that need to be executed earlier.

2.2 Abductive Logic Programming

LPS is based on abductive logic programming (ALP) [9] and abductive logic programming agents (ALP agents) [12]. ALP extends logic programming (LP) by allowing some predicates, ***Ab***, the *abducibles*, to be undefined, in the sense that they do not occur in the conclusions of clauses. Instead, they can be assumed, but are constrained directly or indirectly by a set ***IC*** of *integrity constraints*.

Thus an *ALP framework* <***L***, ***Ab***, ***IC***> consists of a logic program ***L***, a set of abducibles ***Ab***, and a set of integrity constraints ***IC***. The predicates in the conclusions of clauses in ***L*** are disjoint from the predicates in ***Ab***. An atom whose predicate is in ***Ab*** is called *abducible*. In LPS, the abducible atoms represent actions and events, and the integrity constraints represent reactive rules (or policies).

In LPS, we use integrity constraints for reactive rules and restrict them to the form *condition* $\rightarrow$ *conclusion,* where *condition* and *conclusion* are conjunctions of atoms, and all the variables occurring in *condition* are universally quantified over the implication, and all variables occurring only in the *conclusion* are existentially quantified over the *conclusion*. For simplicitly, we restrict logic programs to Horn clauses [11]. This restriction has the advantage that Horn clauses have a unique minimal model [5]. The restriction can be relaxed in various ways, as we will discuss later.

Definition 1. Given an ALP framework <***L***, ***Ab***, ***IC***> and a conjunction of atoms *C* (which can be the empty clause), a *solution* is a set of atomic sentences Δ in the predicates ***Ab***, such that both *C* and ***IC*** are true in the minimal model of $\boldsymbol{L} \cup \Delta$. □

This semantics is one of several that have been proposed for ALP and for integrity constraints more generally. It has the advantage that it provides a natural semantics for LPS. In LPS, the analogue of the minimal model of $\boldsymbol{L} \cup \Delta$ is the sequence of database states extended by the logic programming component of LPS. The analogue of *C* and ***IC*** being true in the minimal model is the truth of the initial goals and reactive rules.

The ALP agent model [12] embeds the IFF [8] proof procedure for ALP in an observation-thought-decision-action cycle, in which abducible atoms ***Ab*** represent an agent's observations and actions, logic programs ***L*** represent the agent's beliefs, and integrity constraints ***IC*** represent the agent's goals. Logic programs give the proactive behaviour of goal-reduction procedures, and integrity constraints give the reactive behaviour of condition-action-and-goal rules. However, goals and beliefs also have a declarative reading, inherited from the semantics of ALP. The ALP agent cycle generates a sequence of actions in the attempt to make an initial goal and the integrity constraints true in the agent's environment.

In ALP agents, the agent's environment is an external, destructively changing semantic structure. The set Δ, on the other hand, is the agent's internal representation of its interactions with the environment. This internal representation is monotonic in ALP, in the sense that observations and actions are time-stamped and state representations are derived by an action theory, such as the situation or event calculus. In contrast, in production systems, in many agent systems and in LPS, the environment is simulated by an internal, destructively changing database. In LPS, this database can be viewed as a Kripke-like model, transformed into a single situation-calculus-like model.

3 LPS Language – Informal Description

In this section we give an informal description of the LPS language, and in the next section we define the language and its internal, state-based representation.

3.1 The Database

The LPS semantics is defined in terms of a minimal model associated with a sequence of databases state transitions $\boldsymbol{W_0}$, Ob_0, a_0, …, $\boldsymbol{W_i}$, Ob_i, a_i…where the $\boldsymbol{W_i}$ represent the successive states of the database, the Ob_i represent a set of observations, and the a_i represent the agent's actions.

The databases $\boldsymbol{W_i}$ represent the agent's beliefs about the current state of the environment. These correspond to the extensional predicates of a deductive database, e.g. *customer(john-smith), spent-to-date(john-smith, 500).* Because the transition from $\boldsymbol{W_i}$ to $\boldsymbol{W_{i+1}}$ is implemented by destructive assignment, the facts in $\boldsymbol{W_i}$ are written without state arguments. This means that the facts that are not affected by the transformation persist without being copied explicitly from one state to the next.

In addition to extensional predicates, which represent database states explicitly, there are intentional predicates defined by clauses $\boldsymbol{L_{ram}}$. For example:

status(X, gold) ← spent-to-date (X, V): 500≤V.
status(X, new) ← spent-to-date (X, V): V <500.

Here *spent-to-date* is an extensional predicate, which changes directly as the result of actions, such as *take-payment*, and *status* is an intensional predicate, which changes as a ramification of changes to the predicate *spent-to-date*.

The state-independent predicates are defined by ordinary logic programming clauses in $\boldsymbol{L_{stateless}}$. For example: *similar(X, Y) ← cd(X) ∧ dvd(Y).*

3.2 The Action Theory

State transitions are defined by a set of *action clauses* $\boldsymbol{A}$. The clauses in $\boldsymbol{A}$ are divided into clauses $\boldsymbol{A_{pre}}$ defining the preconditions and $\boldsymbol{A_{post}}$ defining the post-conditions of atomic actions. These have the form:

initiates(a, p) ← init-conditions
terminates(a, p) ← term-conditions
precondition(a, q) ← pre-conditions

where *a* represents an atomic action, *p* represents an extensional predicate and *q* represents an intensional, extensional or state-independent predicate. The first two types of clauses are in $\boldsymbol{A_{post}}$, and the last type of clause is in $\boldsymbol{A_{pre}}$. The conditions *init-conditions and term-conditions* are qualifying conditions, and together with *pre-conditions* are formulas that are checked in the current state. For example:

initiates(take-payment(X, ID, Value), spent-to-date(X, New)) ←
spent-to- date(X, Old) ∧ New = Old + Value.
terminates(take-payment(X, ID, Value), spent-to-date(X, Old)) ←
spent-to-date(X, Old).

An action *a* can be executed in state $\boldsymbol{W_i}$ provided that all of its precondions hold. This is determined by using the action theory to identify all the predicates *q* that should hold, and then checking that all such *q* do indeed hold in the current state $\boldsymbol{W_i}$ extended by means of the intensional and stateless predicates, as determined by $\boldsymbol{L_{ram}}$ and $\boldsymbol{L_{stateless}}$. Not every action needs to initiate or terminate database facts. In particular, the LPS agent can execute external actions, which have no impact on the database.

For simplicity and uniformity, we treat observations as external events that initiate and terminate fluents. Their postconditions are included in $\boldsymbol{A_{post}}$. For example:

initiates(login(X), logged-on(X)) *terminates(logout(X), logged-on(X)).*

Because observations only happen if they can happen, there is no need to include their preconditions in $\boldsymbol{A_{pre}}$. It is important to note that action theories are **not** used for planning, but only to perform the state transitions associated with the agent's actions and external events. We use planning clauses for planning.

3.3 Goals

In addition to the changing state of the database, the LPS operational semantics maintains an associated changing set of goal clauses $\boldsymbol{G_i}$, each of which can be regarded as a *partial plan* for achieving the initial goals $\boldsymbol{G_0}$ and the additional goals generated by the LPS cycle. These additional goals come from the conclusions of reactive rules. Both the initial goals and the additional goals are reduced to sub-goals by the logic programs used to define intensional predicates, macro-actions, stateless predicates and planning rules. Goals coming from different reactive rules can be solved independently and concurrently.

The intended semantics of goals is that, for every $\boldsymbol{G_i}$, one of the goal clauses in $\boldsymbol{G_i}$ should be true in the model that is generated by the LPS cycle. $\boldsymbol{G_0}$ may contain only the empty clause, as is typical of production systems. Informally speaking, the cycle succeeds in state *n*, if $\boldsymbol{G_n}$ contains the empty clause. However, the cycle does not terminate when it succeeds, because future observations may trigger future goals.

Initial goal clauses can contain actions, fluents, stateless predicates, and any of the logical connectives in the language, but not events. For example the goal clause

promotional-offer(Item): discount(Item, 20%, NewPrice); advertise(Item, NewPrice)

requires that a promotional item is determined and discounted by *20%*, and then the item and its new price are advertised.

3.4 Reactive Rules

The set $\boldsymbol{P}$ of *reactive rules* has the same form *condition* $\rightarrow$ *conclusion* and the same implicit quantification as ALP integrity constraints, where *condition* is a conjunction of atoms and *conclusion* has the same form as a goal clause. Reactive rules are executed by checking whether the *condition* holds in the current state of the database W_i, and if it does, then the *conclusion* is added to every goal clause in $\boldsymbol{G_i}$. The *condition* can also include a single atom representing an atomic action executed in the last cycle and any number of atoms representing the last set of observations. Thus ***P*** can include the event-condition-action rules of active databases. For example:

take-payment(X, ID, Value) : *Value≥50 →issue-sport-voucher(X, ID).*

3.5 Macro-actions

It would be possible to write agent programs using reactive rules alone, restricting the conclusions of reactive rules to atomic actions, and to extensional and intensional predicates that are checked in the current state as implicit consequences of the agent's actions or as serendipitous consequences of external events. Such reactive rules would be sufficient for implementing purely reactive agents. However, macro-actions and planning rules in LPS make it possible to implement agents with more deliberative/proactive capabilities.

Macro-actions are complex actions defined in terms of simpler (atomic and macro-) actions and fluents. Macro-actions, defined by the set of clauses $\boldsymbol{L_{macro}}$, are like transactions in *TR* Logic and complex actions in Golog. Examples were given in section 1.1.

3.6 Planning Clauses

Agent programs written using only reactive rules and definitions of macro-actions achieve fluent goals only emergently and implicitly. Planning clauses allow programs to be written to achieve extensional fluent goals explicitly. To ensure that the agent's beliefs are true with respect to the action theory that maintains the database, we impose the restriction that the last condition in a planning clause is an atomic action that initiates the conclusion fluent, as determined by the action theory. $\boldsymbol{L_{plan}}$ represents such plans for achieving future states of the database. For example:

have(Gift)←is-Store(Store): sells(Gift,Store): goto(Store); purchase(Gift, Store).

Note that that the conclusions of plans represent the motivations of the agent's actions, in contrast with the action theory, which represents all the consequences of the agent's actions. For example, here the action theory may include clauses specifying other consequences of *purchase(Gift, Store)*, for example that the agent's financial resources will be reduced by the amount of the *Gift*.

Thus the planning clauses, together with the macro-action definitions implement planning from second principles, using pre-compiled plan schemata. However, planning clauses can also be used to implement planning from first principles, by including a planning clause of the form:

p ←init-conditions: pre-conditions$_1$*: q*$_1$*: ....: pre-conditions*$_n$*: q*$_n$*: a*
for every set of clauses
initiates(a, p) ← init-conditions
precondition(a, q$_1$*) ← pre-conditions*$_1$
....
precondition(a, q$_n$*) ← pre-conditions*$_n$

where the q_i are all the preconditions of *a*.

Whether the planning clauses are used for planning from first principles or planning from second principles, they share with classical planning the repeated reduction of fluent goals to fluent and action sub-goals. Because LPS is neutral with respect to search and action selection strategies, different strategies for interleaving planning and execution can be implemented. At one extreme, as in classical planning, plans can be fully generated before they are executed. At the other extreme, actions can be executed as soon as they are generated in a partial plan.

We now define the LPS language formally.

4 LPS Language – Formal Description

The vocabulary of LPS is divided into fluent, action, and auxiliary predicates. The *fluent* predicates consist of extensional and intentional predicates. The action predicates consist of atomic, macro-actions, and observations of external events, in the sets *A, M* and *Ob* respectively. The *auxiliary* predicates consist of "ordinary" stateless predicates and the predicates *initiates, terminates, precondition* in the action theory. All these sets of predicates are mutually exclusive.

The LPS framework presented in this paper employs a stateless *surface* syntax, which is syntactic sugar for an underlying *internal* syntax with explicit state arguments (which specify the semantics of the surface syntax). We use the internal syntax when describing the operational and the model-theoretic semantics later in the paper. Now we describe both the surface syntax and its semantics.

The surface syntax of all LPS components is defined in terms of *sequences* of predicates, where consecutive predicates are linked by : or ;. The *syntax of sequences* is defined recursively. We take the base case to be the empty sequence, which is also the empty clause, and we write it as *true*. If *P* is a predicate and *S* is a sequence, then *P:S* and *P;S* are sequences.

Below, where it is clear from the context, we use the terminology (fluent, stateless, atomic action, macro-action, event, extensional, intentional) predicate to mean an atom with such a predicate. The initial goal $\boldsymbol{G_0}$ is a set of goal clauses, each of which is a sequence with no events. Other goals $\boldsymbol{G_i}$, derived in the LPS cycle are sets of clauses expressed in the internal syntax with state arguments. They do not appear in the surface syntax.

In the internal syntax, goal clauses are conjunctions of atoms, and the goals $\boldsymbol{G_i}$ represent disjunctions of goal clauses. These goals have a search tree structure, which is not apparent in the set representation. As in normal logic programming, other representations, including search tree and and-or tree representations are possible. For simplicity, we do not explore these other representations in this paper.

$\boldsymbol{L_{stateless}}$ clauses have the form: $P \leftarrow P_1{:}P_2{:}\ldots{:}P_n,\ 0\leq n$, where *P* and each P_i are stateless predicates.

$\boldsymbol{L_{ram}}$ clauses have the form: $P \leftarrow P_1{:}P_2{:}\ldots{:}P_n,\ 1\leq n$, where *P* is an intensional predicate, each P_i is a fluent or stateless predicate and at least one P_i is a fluent.

$\boldsymbol{L_{macro}}$ clauses have the form: $M \leftarrow S$, where *M* is a macro-action predicate and *S* is a sequence containing at least one fluent or action predicate, and no event.

$\boldsymbol{L_{plan}}$ clauses have the form: $P \leftarrow S$ where *P* is an extensional predicate, and *S* is a sequence containing no event, and ending in an atomic action.

P reactive rules have the form: $[Evt_1 \wedge Evt_2 \wedge \ldots \wedge Evt_n \wedge A]: Q_1{:}Q_2{:}\ldots{:}Q_m \rightarrow S$ where S is a non-empty sequence, containing no event, and each Q_i is a fluent, or stateless predicate, A is an atomic action, and each Evt_i is an event. All Evt_i and A may be absent, in which case $1 \leq m$, otherwise $0 \leq m$.

A clauses have the forms : $initiates(a, p) \leftarrow P_1{:}P_2{:}\ldots{:}P_n$
$terminates(a, p) \leftarrow P_1{:}P_2{:}\ldots{:}P_n$
$precondition(a, q) \leftarrow P_1{:}P_2{:}\ldots{:}P_n$

where each P_i is a fluent or stateless predicate, and $0 \leq n$.

The semantics of each formula F of LPS, including predicates, goals, rules, clauses and sequences, is denoted by F^*. Either F is a stateless predicate, or F^* can be written in the form $F^*(T_1, T2)$, where T_1 and $T2$ are as explained below. The semantics of an atomic formula P is given by:

true is a stateless predicate, *true** is *true*.
If P is a stateless predicate, then P^* also written $P^*(T)$ is P.
If P is a fluent, then P^* also written $P^*(T, T)$ is $P(T)$.
If P is an atomic action or an event
then P^* also written $P^*(T, T+1)$ is $P(T, T+1)$.
If P is a macro-action, then P^* *also written* $P^*(T_1, T2)$ is $P(T_1, T2)$.

Sequences have a similar semantics to predicates, either as stateless sequences or with two state arguments, which can be identical. The semantics of sequences is defined recursively, with the empty sequence having the semantics *true*.

Let P be a predicate and S a sequence, with semantics P^* and S^* respectively.
Let F be $P{:}S$, where neither P nor S is stateless.
Then $F^*(T_1, S_2)$ is $P^*(T_1, T_2) \wedge S^*(S_1, S_2) \wedge T_2 = S_1$.
Let F be $P;S$ where neither P nor S is stateless.
Then $F^*(T_1, S_2)$ is $P^*(T_1, T_2) \wedge S^*(S_1, S_2) \wedge T_2 \leq S_1$.
Let F be $P{:}S$ or $P;S$. Then:
If both P and S are stateless, then F^* is $P^* \wedge S^*$ and stateless.
If P is stateless and S is not, then $F^*(T_1, T_2)$ is $P^* \wedge S^*(T_1, T_2)$.
If S is stateless and P is not, then $F^*(T_1, T_2)$ is $P^*(T_1, T_2) \wedge S^*$.

The semantics of the initial goal $\boldsymbol{G_0}$ is the semantics of its sequences. All the variables in $\boldsymbol{G_0}$ (and subsequent $\boldsymbol{G_i}$) are existentially quantified.

The semantics of $\boldsymbol{L_{ram}}$ clauses $P \leftarrow P_1{:}P_2{:}\ldots{:}P_n$ is $P(T) \leftarrow P_1(T) \wedge P_2(T) \wedge \ldots \wedge P_n(T)$.
The semantics of $\boldsymbol{L_{macro}}$ clauses $M \leftarrow S$ is $M(T_1, T_2) \leftarrow S^*(T_1, T_2)$.
The semantics of $\boldsymbol{L_{plan}}$ clauses $P \leftarrow S$ is $P(T_2) \leftarrow S^*(T_1, T_2)$.

Note that these clauses do not contain any analogue of the frame axiom(s) in the situation calculus. Persistence (or inertia), which is formalised by frame axioms, is obtained in LPS implicitly through the maintenance of the current state of the database, without the computational overheads of reasoning with frame axioms.

The semantics of reactive rules $[Evt_1 \wedge Ev_2 \wedge \ldots \wedge Evt_n \wedge A]: Q_1{:}Q_2{:}\ldots{:}Q_m \rightarrow S$ is

> $[Evt_1(T\text{-}1, T) \wedge \ldots \wedge Evt_n(T\text{-}1, T) \wedge A(T\text{-}1, T)] \wedge Q_1(T) \wedge Q_2(T) \wedge \ldots \wedge Q_m(T) \rightarrow S^*$ if S is stateless, and $[Evt_1(T\text{-}1, T) \wedge \ldots \wedge Evt_n(T\text{-}1, T) \wedge A(T\text{-}1, T)] \wedge Q_1(T) \wedge Q_2(T) \wedge \ldots \wedge Q_m(T) \rightarrow S^*(T_1, T_2) \wedge T \leq T_1$ otherwise.

The conditions of reactive rules do not contain macro-actions, because the sequence of states from T_1 to T_2 associated with the semantics $M(T_1, T_2)$ of a macro-action M is generally not accessible in the current state T of the database.

The semantics of $\boldsymbol{A}$ clauses:

$initiates(a, p) \leftarrow P_1{:}\ P_2{:}\ \ldots{:}\ P_n$	is $initiates(a, p, T) \leftarrow P_1(T) \wedge P_2(T) \wedge \ldots \wedge P_n(T)$
$terminates(a, p) \leftarrow P_1{:}\ P_2{:}\ \ldots{:}\ P_n$	is $terminates(a, p, T) \leftarrow P_1(T) \wedge P_2(T) \wedge \ldots \wedge P_n(T)$
$precondition(a, q) \leftarrow P_1{:}\ P_2{:}\ \ldots{:}\ P_n$	is $precondition(a, q, T) \leftarrow P_1(T) \wedge P_2(T) \wedge \ldots \wedge P_n(T)$

Finally if S is a set of formulas then S^* is the set of all F^* for F in S.

Note that these syntax and semantics impose the restriction that no two actions (whether from A or M) have the same pair of state arguments. This is because, for simplicity, the LPS operational semantics executes at most a single action in each cycle/state. Because the operational and model-theoretic semantics of LPS are both defined for the internal semantics, it is possible to define other surface syntaxes and to mix state-based and stateless syntaxes. The syntax chosen for this paper can be extended in several ways, but has the advantage of simplicity.

5 The Operational Semantics

The operational semantics manipulates the database by adding and deleting extensional predicates. However, the model-theoretic semantics interprets the facts in state $\boldsymbol{W_i}$ as containing the implicit state argument i. We use the notation $\boldsymbol{W^*_i}$ when we need to refer to facts containing explicit state arguments: $W^*_i = \{p(i) : p \in W_i\}$.

Actions and events update the database from one state to the next, as specified in the LPS cycle below. However, for the execution of an action a to be attempted all of its preconditions must hold in the current state of the database $\boldsymbol{W_i}$.

Definition 2. An action a is *executable* in state $\boldsymbol{W_i}$ if and only if for every *precondition(a, q, i)* that holds in $\boldsymbol{W^*_i} \cup \boldsymbol{A^*} \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{stateless}}$,
q holds in $\boldsymbol{W^*_i} \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{stateless}}$. □

In the LPS cycle, when an action is chosen for execution, all of its arguments (other than state arguments) need to be variable-free (a safety requirement). In addition, the selection function and search strategy need to be *timely*, as defined below.————-□

Definition 3. A selection function is *safe* if and only if, when it selects an action, the action is ground (except possibly for state variables). A selection function is *timely* if and only if, when it selects an action $a(t, t{+}1)$ in a goal clause C, then C contains no other atom which is earlier in the same sequence in C. A search strategy is *timely* if and only if, when it resolves an extensional atom in a goal clause C with the database, then C contains no other atom which is earlier in the same sequence in C. □

Note that the selection function is not restricted to selecting predicates in the sequence in which they are written. Predicates can be selected and resolved, so that planning and execution are interleaved. However, to ensure the existence of safe selection functions, LPS frameworks need to be *range-restricted.* We define range-restriction after the LPS cycle.

The operational semantics is a potentially non-terminating cycle in which the agent repeatedly observes events in the environment, updates the database to reflect the changes brought about by those events, performs a bounded number of inferences, and selects an action to execute. If there is no such action that can be executed within the bound or if the action is attempted and fails, then an empty action is generated. Similarly, if there is no observation available at the beginning of a cycle then the set of observations is empty.

The internal syntax of LPS clauses and rules includes inequalities (≤) between states. For the model-theoretic semantics we need a theory $\boldsymbol{L_{temp}}$ that defines the inequality relation. However, this theory is not needed for the operational semantics, because timeliness and range-restriction ensure that if all other goals in a goal clause succeed, then all the inequalities in the goal clause are also true. So for implementation purposes we can assume the inequalities are deleted from the clauses and rules. This is equivalent to resolving inequalities with clauses in $\boldsymbol{L_{temp}}$, which always succeeds.

Definition 4. LPS cycle: Let **Max** be a bound on the number of resolution steps to be performed in each iteration of cycle. Given a range-restricted LPS framework $<\boldsymbol{W_0}, \boldsymbol{G_0}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{L_{ram}}, \boldsymbol{L_{stateless}}, \boldsymbol{L_{macro}}, \boldsymbol{L_{plan}}>$, a safe and timely selection function $\boldsymbol{s}$, a timely search strategy $\sum$, and a sequence of sets of observations $Ob_0, Ob_1, \ldots$, the LPS cycle determines a sequence of state transitions $<\boldsymbol{W_0}, \boldsymbol{G_0}>$, (Ob_0, a_0), …, $<\boldsymbol{W_i}, \boldsymbol{G_i}>$, (Ob_i, a_i)…where for all i, $0 \leq i$, Ob_i, a_i and $<\boldsymbol{W_{i+1}}, \boldsymbol{G_{i+1}}>$ are obtained from Ob_{i-1}, a_{i-1} and $<\boldsymbol{W_i}, \boldsymbol{G_i}>$ by the following steps:

LPS0. Let Ob_i be the set of observations made in this round of cycle. $\boldsymbol{W_i}$ is updated to $\boldsymbol{WO_i}$ as follows: $\boldsymbol{WO_i} = (\boldsymbol{W_i} - \{p\text{: } a \in Ob_i$ and $terminates(a, p, i)$ holds in $\boldsymbol{W}^*_i \cup \boldsymbol{A}^* \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{stateless}}\}) \cup \{p\text{: } a \in Ob_i$ and $initiates(a, p, i)$ holds in $\boldsymbol{W}^*_i \cup \boldsymbol{A}^* \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{stateless}}\}$.

LPS1. For every instance $condition\ \sigma \rightarrow conclusion\ \sigma$ of a rule in $\boldsymbol{P}^*$ such that $condition\ \sigma$ holds in $\boldsymbol{WO_i}^* \cup \{a_{i-1}^*\} \cup Ob_{i-1}^* \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{stateless}}$, add $conclusion\ \sigma$ to every clause in $\boldsymbol{G_i}$. Let $\boldsymbol{GP_i}$ be the resulting set of goal clauses.

LPS2. Using the selection function $\boldsymbol{s}$ and search strategy $\sum$, let $\boldsymbol{GPL_i}$ be a set of goal clauses, starting from $\boldsymbol{GP_i}$, derivable by SLD-resolution using the clauses in $\boldsymbol{WO_i}^* \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{plan}}^* \cup \boldsymbol{L_{macro}}^* \cup \boldsymbol{L_{stateless}}$ such that one of the following holds:

LPS2.1 No goal clause containing an executable action is generated within the maximum number, **Max**, of resolution steps. This includes the case of an empty clause being generated. Then $\boldsymbol{G_{i+1}} = \boldsymbol{GPL_i}$, $\boldsymbol{W}_{i+1} = \boldsymbol{WO_i}$, and a_i is the *empty action* ϕ (an action that will always succeed, but has no effect on the database). Cycle will proceed into further rounds because further observations are possible. (An agent cycle must be perpetual; it never stops, because there can always be observations.)

LPS2.2 At least one goal clause whose selected literal is an executable action is generated within the maximum number, **Max**, of resolution steps.

LPS2.2.1 Then one such action $a(T,T+1)$ in a goal clause C in $\boldsymbol{GPL_i}$ is chosen for execution by the search strategy $\sum$. Note that $a(T, T+1)$ might have been generated and selected in an earlier cycle, but not have been executable before. Moreover, even if it was selected and executable before, the search strategy might have chosen some other action. Moreover, it might have been executed and failed. It might even have been executed before and succeeded, but might need to be executed again, because later goals, dependent upon it, have failed. Note T can be a constant = i or a variable.

LPS2.2.2 The action $a(T,T+1)$ is executed. If the action fails, then $\boldsymbol{G_{i+1}} = \boldsymbol{GPL_i}$, $\boldsymbol{W_{i+1}} = \boldsymbol{WO_i}$, and a_i is the *empty action* ϕ. If the action succeeds, then a_i is $a(i, i+1)$.
$\boldsymbol{G_{i+1}} = \boldsymbol{GPL_i} \cup \boldsymbol{C'}$, where $\boldsymbol{C'}$ is the resolvent of C with $a(i, i+1)$.
$\boldsymbol{W_{i+1}} = (\boldsymbol{WO_i} - delete(a)) \cup add(a)$ where
$delete(a) = \{p: terminates(a, p, i)$ holds in $\boldsymbol{WO}^*_i \cup \boldsymbol{A}^* \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{stateless}}\}$
$add(a) \quad = \{p: initiates(a, p, i)$ holds in $\boldsymbol{WO}^*_i \cup \boldsymbol{A}^* \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{stateless}}\}$. □

The LPS cycle is an operational semantics, not an efficient proof procedure. However, there are many refinements that would make it more efficient. These include the deletion of subsumed clauses (including all other goal clauses, once the empty goal clause has been generated), as well as the deletion of clauses containing fluents or actions whose state argument is instantiated to a state earlier than the current state.

Definition 5. The cycle *succeeds in state n* if and only if $\boldsymbol{G_n}$ contains an empty clause and $\boldsymbol{GP_n} = \boldsymbol{G_n}$. □

Definition 6. An LPS framework $<\boldsymbol{W_0}, \boldsymbol{G_0}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{L_{ram}}, \boldsymbol{L_{stateless}}, \boldsymbol{L_{macro}}, \boldsymbol{L_{plan}}>$ is *range-restricted* if and only if all rules in $\boldsymbol{P}$ and all clauses in $\boldsymbol{A}$, $\boldsymbol{L_{ram}}$, $\boldsymbol{L_{stateless}}$, $\boldsymbol{L_{macro}}$, $\boldsymbol{L_{plan}}$ and $\boldsymbol{G_0}$ are range-restricted, where:

A sequence S is range-restricted if and only if every variable in an atomic action in S occurs earlier in the sequence.

A clause *conclusion* ← *conditions* in $\boldsymbol{L_{ram}}$, $\boldsymbol{L_{stateless}}$, $\boldsymbol{L_{macro}}$, $\boldsymbol{L_{plan}}$ is range-restricted if and only if *conditions* is range-restricted and every variable in *conclusion* occurs in *conditions.*

A clause *conclusion* ← *conditions* in $\boldsymbol{A}$, where *conclusion* is *initiates(a, p)*, *terminates(a, p),* or *precondition(a, p),* is range-restricted if and only if every variable in *p* occurs either in *conditions* or in *a.*

A rule *condition* → *conclusion* in $\boldsymbol{P}$ is range-restricted if and only if every variable occurring in an atomic action *a* in *conclusion*, occurs either in the *condition* or in an atom earlier than *a* in the *conclusion.* □

6 Model-Theoretic Semantics

The model-theoretic semantics requires a Horn clause definition $\boldsymbol{L_{temp}}$ of the inequality relations. Any correct definition will serve the purpose including, for example:

$$0 \leq T \qquad\qquad S+1 \leq T+1 \leftarrow S \leq T.$$

Every set $\boldsymbol{S_n}$ of sentences $\boldsymbol{W_0}^* \cup \ldots \cup \boldsymbol{W_n}^* \cup \{a_0^*, \ldots, a_{n-1}^*\} \cup Ob_0^* \cup \ldots \cup Ob_{n-1}^* \cup \boldsymbol{L_{stateless}} \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{temp}} \cup \boldsymbol{L_{macro}}^*$ is a Horn clause logic program. Therefore, $\boldsymbol{S_n}$ has a unique minimal model $\boldsymbol{M_n}$. This model is like a Kripke structure of possible worlds $\boldsymbol{M^i} = \boldsymbol{W_i} \cup \boldsymbol{L_{stateless}} \cup \boldsymbol{L_{ram}}$ embedded in a single model $\boldsymbol{M_n}$, where the actions and observations $\{(Ob_0, a_0), \ldots, (Ob_{n-1}, a_{n-1})\}$ determine the transition relation from one possible world to another.

6.1 Soundness

To prove the soundness of the LPS cycle, $\boldsymbol{L_{plan}}$ needs to be compatible with the action theory $\boldsymbol{A}$. Compatibility ensures that the clauses in $\boldsymbol{L_{plan}}^*$ are *true* in all $\boldsymbol{M_n}$.

Definition 7. $\boldsymbol{L_{plan}}$ is *compatible* with $\boldsymbol{A}$ if for every clause in $\boldsymbol{L_{plan}}$ of the form $p \leftarrow S$ there exists an instance of a clause in $\boldsymbol{A}$ of the form $initiates(a, p) \leftarrow P_1{:}\ P_2{:}\ \ldots{:}\ P_n$ such that $S^* \cup \boldsymbol{L_{stateless}} \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{temp}}$ entails $(P_1{:}\ P_2{:}\ \ldots{:}\ P_n)^*$. □

It is easy to satisfy this condition, and all the examples in this paper, if done in full will have this property. Note that we can plan to achieve intentional atoms by combining such clauses in $\boldsymbol{L_{plan}}$ with clauses in $\boldsymbol{L_{ram}}$ and $\boldsymbol{L_{macro}}$.

Theorem. Given a range-restricted LPS framework $<\boldsymbol{W_0}, \boldsymbol{G_0}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{L_{ram}}, \boldsymbol{L_{stateless}}, \boldsymbol{L_{macro}}, \boldsymbol{L_{plan}}>$, a safe and timely selection function $\boldsymbol{s}$, a timely search strategy $\sum$, and a sequence of sets of observations $Ob_0, Ob_1, \ldots, Ob_{n-1}$, if $\boldsymbol{L_{plan}}$ is compatible with $\boldsymbol{A}$ and the cycle succeeds in state n, then some clause C_0 in $\boldsymbol{G_0}^*$ is true in $\boldsymbol{M_n}$ and all the rules in $\boldsymbol{P}^*$ are true in $\boldsymbol{M_n}$.

Sketch of proof: If the cycle succeeds in state n, then $\boldsymbol{G_n}$ contains the empty clause. The proof of this empty clause can be traced backwards to a sequence of clauses, starting with some C_0 in $\boldsymbol{G_{0^*}}$: $C_0, \ldots, C_i, \ldots, C_m = true$, where C_{i+1} is obtained from C_i in one of two ways:

1. In LPS1, C_{i+1} is C_i conjoined with *conclusion* σ for every instance *condition* $\sigma \rightarrow$ *conclusion* σ of a rule in $\boldsymbol{P}^*$ such that *condition* σ holds in $\boldsymbol{WO_i}^* \cup \{a_{i-1}^*\} \cup Ob_{i-1}^* \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{stateless}}$.
2. C_{i+1} is obtained by SLD-resolution between C_i and some clause C in $\boldsymbol{WO_i}^* \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{plan}}^* \cup \boldsymbol{L_{macro}}^* \cup \boldsymbol{L_{stateless}}$ in LPS2, by resolution with a_j^* in LPS2.2.2, or by implicit resolution of inequalities with clauses in $\boldsymbol{L_{temp}}$.

It suffices to prove the **lemma:** All the C_i are true in $\boldsymbol{M_n}$. The lemma implies that $\boldsymbol{C_0}$ is true in $\boldsymbol{M_n}$. Together with the condition $\boldsymbol{GP_n} = \boldsymbol{G_n}$, the lemma also implies that all the rules in $\boldsymbol{P}^*$ are true in $\boldsymbol{M_n}$.

Proof of lemma: The lemma follows by induction, by showing the base case C_m = *true* is true in $\boldsymbol{M_n}$ and the induction step if C_{i+1} is true in $\boldsymbol{M_n}$, then C_i is true in $\boldsymbol{M_n}$. The base case is trivial. For the induction step, there are two cases: In case 1 above, if C_{i+1} is true in $\boldsymbol{M_n}$, then C_i is true in $\boldsymbol{M_n}$, because if a conjunction is true then so are all of its conjuncts.

In case 2 above, the clauses C_{i+1} and C_i are actually the negations of clauses in ordinary resolution. So, according to the soundness of ordinary resolution, $\neg C_{i+1}$ is a logical consequence of $\neg C_i$ and C. Therefore, if both C and C_{i+1} are true in $\boldsymbol{M_n}$, then C_i is true in M. But any clause C in $\boldsymbol{WO_i}^* \cup \{a_{i-1}^*\} \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{macro}}^* \cup \boldsymbol{L_{stateless}} \cup \boldsymbol{L_{temp}}$ is true in $\boldsymbol{M_n}$ by the definition of $\boldsymbol{M_n}$. It suffices to show that all clauses in $\boldsymbol{L_{plan}}^*$ are also true in $\boldsymbol{M_n}$. But this follows from the compatibility of $\boldsymbol{L_{plan}}$ with $\boldsymbol{A}$.

This theorem is restrictive in two ways. First, it considers only the first n sets of observations. Second, it considers only the case in which the actions needed to solve all the goals in $\boldsymbol{G_0}$ and introduced by the reaction rules are successfully executed by state n. Both of these restrictions can be liberalised, mainly at the expense of complicating the statement of the theorem, but the proofs are similar. We omit the theorems and their proofs for lack of space. However, it is worth noting that to deal with potentially non-terminating sets of observations, we need minimal models $\boldsymbol{M_\infty}$ determined by the potentially infinite Horn clause program $\boldsymbol{W_0}^* \cup \ldots \cup \boldsymbol{W_n}^* \cup \ldots \{a_0^*, \ldots, a_n^*, \ldots\} \cup Ob_0^* \cup \ldots \cup Ob_n^* \cup \ldots \boldsymbol{L_{stateless}} \cup \boldsymbol{L_{ram}}^* \cup \boldsymbol{L_{temp}} \cup \boldsymbol{L_{macro}}^*$.

Note also that LPS can be extended to include negation in both the conditions and conclusions of reaction rules and in the conditions of clauses. The most obvious such extension is to the case of locally stratified programs with their perfect models.

6.2 Completeness

Because of the completeness result for the IFF proof procedure [8] for ALP, it might be expected that a similar completeness result would hold for LPS: Given a minimal model $\boldsymbol{M}$ of some clause C_0 in $\boldsymbol{G_0}$ and of all the rules in $\boldsymbol{P}$, it might be hoped that there would exist some search strategy $\sum$ that together with the LPS cycle could generate some related model $\boldsymbol{M}'$, possibly determined by a subsequence of the actions of $\boldsymbol{M}$. Unfortunately this is not always possible. The LPS cycle will not generate models that make rules true by making their conditions false. For example:

$\boldsymbol{P}$: $q \rightarrow a$ $\qquad$ $\boldsymbol{A}$: *terminates(b, q)* $\qquad$ $\boldsymbol{W_0}$: $\{q\}$

Here a and b are actions. There is a minimal model corresponding to the sequence of actions b, a, but the LPS cycle can only generate the non-terminating sequence $a, a, \ldots$

This problem can be dealt with in the manner of the IFF proof procedure, by replacing every reactive rule of the form $p : q \rightarrow a$ with rules of the form $p: q \rightarrow a \vee b \vee$ c, where b and c are atomic actions such that *terminates(b, q)* and *terminates(c, p)*. We do not consider completeness further here for lack of space.

6.3 Relationship with the Situation Calculus and Event Calculus

The minimal model $\boldsymbol{M}$ generated by LPS is both like a modal possible worlds semantic structure and like a minimal model of the situation calculus represented as a logic program. Ignoring observations and simplifying the situation calculus representation, the frame axioms have the form:

$$P(T+1) \leftarrow P(T) \wedge A(T, T+1) \wedge \neg\, terminates(A, P, T)$$

for every extensional predicate *P*. In LPS, these axioms are true in ***M***, but are not used to generate ***M***. Instead of reasoning explicitly that most fluents *P* that hold in state *T* continue to hold in *T+1*, destructive assignment is used to update only those fluents explicitly affected by *A*.

The use of destructive assignment, as in LPS, to implement the frame axiom, can be exploited for other applications, such as planning, provided only one state is explored at a time. In particular, for classical planning applications, the LPS approach can be generalised to store the complete history of actions and events leading up to a current database state. The database can be rolled back to reproduce previous states, and rolled forward to generate alternative databases states. However, these possibilities are topics of research for the future.

7 Related and Future Work

LPS provides an agent framework that combines a model-theoretic semantics with a state-free syntax and a database maintained by destructive assignment. To the best of our knowledge, this combination is novel. Most agent frameworks have an operational semantics, but no declarative semantics. Some logic-based frameworks like Golog, ALP agents and KGP [10] have a model-theoretic semantics, but represent the environment using time or state and manipulate the representation using the situation or event calculus. Metatem [6], on the other hand, is a logic-based agent language with a Kripke semantics for modal logic sentences resembling production rules. Because of the Kripke-like semantics of LPS, it would be interesting to explore a similar modal syntax for LPS.

Costantini and Tocchio [3] also employ a logic programming approach with a similar model-theoretic semantics, in which external and internal events transform an initial agent program into a sequence of agent programs. The semantics of this evolutionary sequence is given by the associated sequence of models of the sequence of programs. In LPS, this sequence is represented by a single model.

FLUX [15] is a logic programming agent language with several features similar to LPS, including the use of destructive assignment to update states. In FLUX, these states are not stored in a database as in LPS, but in a reified, list-like structure. FLUX employs a sensing and acting cycle, which it uses to plan and execute plans for achievement goals.

Thielscher [17] provides a declarative semantics for AgentSpeak by defining its cycle and procedures by means of a meta-interpreter represented as a logic program. Like LPS, the resulting agent language incorporates a formal action theory. However, unlike LPS, the language does not distinguish between different kinds of procedures, according to their different functionalities. LPS, in contrast, distinguishes between reactive rules, planning rules, macro-actions and ramifications, representing different kinds of AgentSpeak-like procedures in different ways. On the other hand, the agent architecture of Hayashi et al. [18] separates the representation of reactive rules and planning rules, as in LPS.

There is also related work, combining destructive assignment and model-theoretic semantics in other fields, not directly associated with agent programming languages. EVOLP [1], in particular, gives a model-theoretic semantics to evolving logic programs that change state destructively over the course of their execution. Several other authors, including [7, 16] obtain a model-theoretic semantics for event-condition-action rules in active database systems, by translating such rules into logic programs with their associated model theory.

Perhaps the system closest to LPS is Transaction Logic [2], which gives a Kripke-like semantics for transactions (which are similar to macro-actions), represented in a state-free syntax. *TR* Logic also gives a semantics to reactive rules, which involves translating them into transactions. In LPS, the Kripke-like semantics is transformed into a single situation-calculus-like model, in the spirit of Golog. This transformation makes it possible to apply the general-purpose semantics of ALP to the resulting minimal model. In contrast, the semantics of *TR* Logic and Golog are defined specifically for those languages.

Because LPS is based on the ALP agent model and the ALP model is more powerful than LPS, it would be interesting to extend LPS with some additional ALP agent features. These features include: partially ordered plans, more complex constraints on when actions should be performed and when fluent goals should be achieved, concurrent actions, conditionals in the conditions of clauses, active observations, a historical database of past actions and observations, abduction to explain observations that are fluents rather than events, and integrity constraints that prohibit actions rather than generate actions.

It would also be useful to study more closely the relationship between LPS and other agent models with a view to using the LPS approach to provide those languages with model-theoretic semantics. In addition, because the LPS cycle can be viewed as a model generator, which makes the reactive rules true, it would be valuable to explore the relationship with model checking and model generation in other branches of computing.

In this paper we have focused on the theoretical framework of LPS. However, the ultimate test of the framework is its value as a practical agent language. For this purpose, we are developing further enhancements and are experimenting with an implementation.

Acknowledgments. We are grateful to Ken Satoh, Luis Moniz Pereira, Harold Boley, Thomas Eiter and Keith Stenning for helpful discussions, and to the anonymous referees for helpful suggestions.

References

1. Alferes, J., Leite, J., Pereira, L.M., Przymusinska, H., Przymusinski, T.: Dynamic Updates of Non-Monotonic Knowledge Bases. J. of Logic Programming 45(1-3), 43–70 (2000)
2. Bonner, Kifer, M.: Transaction logic programming. In: Warren, D.S. (ed.) Logic Programming: Proc. of the 10th International Conf., pp. 257–279 (1993)
3. Costantini, S., Tocchio, A.: About Declarative Semantics of Logic-Based Agent Languages. In: Baldoni, M., Endriss, U., Omicini, A., Torroni, P. (eds.) DALT 2005. LNCS (LNAI), vol. 3904, pp. 106–123. Springer, Heidelberg (2006)
4. Dennis, L.A., Farwer, B., Bordini, R.H., Fisher, M., Wooldridge, M.: A Common Semantics Basis for BDI Languages. In: Dastani, M.M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS (LNAI), vol. 4908, pp. 124–139. Springer, Heidelberg (2008)

5. van Emden, M., Kowalski, R.: The Semantics of Predicate Logic as a Programming Language. JACM 23(4), 733–742 (1976)
6. Fisher, M.: A Survey of Concurrent METATEM - The Language and its Applications. LNCS, vol. 827. Springer, Heidelberg (1994)
7. Flesca, S., Greco, S.: Declarative Semantics for Active Rules. Theory and Practice of Logic Programming 1(1), 43–69 (2001)
8. Fung, T.H., Kowalski, R.: The IFF Proof Procedure for Abductive Logic Programming. J. of Logic Programming (1997)
9. Kakas, T., Kowalski, R., Toni, F.: The Role of Logic Programming in Abduction. In: Handbook of Logic in Artificial Intelligence and Programming, vol. 5, pp. 235–324. Oxford University Press, Oxford (1998)
10. Kakas, A., Mancarella, P., Sadri, F., Stathis, K., Toni, F.: Computational Logic Foundations of KGP Agents. Journal of Artificial Intelligence Research 33, 285–348 (2008)
11. Kowalski, R.: Predicate Logic as Programming Language. In: Proceedings IFIP Congress, Stockholm, pp. 569–574. North Holland Publishing Co., Amsterdam (1974)
12. Kowalski, R., Sadri, F.: From Logic Programming Towards Multi-agent Systems. Annals of Mathematics and Artificial Intelligence 25, 391–419 (1999)
13. Kowalski, R., Sadri, F.: Integrating Logic Programming and Production Systems in Abductive Logic Programming Agents. In: Proceedings of the Third International Conference on Web Reasoning and Rule Systems, Chantilly, Virginia, USA (2009)
14. Reiter, R.: Knowledge in Action. MIT Press, Cambridge (2001)
15. Thielscher, M.: FLUX: A Logic Programming Method for Reasoning Agents. Theory and Practice of Logic Programming 5(4-5), 533–565 (2005)
16. Zaniolo, C.: A Unified Semantics for Active and Deductive Databases. In: Procs. 1993 Workshop on Rules in Database Systems, RIDS 1993, pp. 271–287. Springer, Heidelberg (1993)
17. Thielscher, M.: Integrating Action Calculi and AgentSpeak. In: Lin, F., Sattler, U. (eds.) Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR), Toronto (2010)
18. Hayashi, H., Tokura, S., Ozaki, F., Doi, M.: Background Sensing Control for Planning Agents Working in the Real World. International Journal of Intelligent Information and Database Systems 3(4), 483–501 (2009)

Model Checking Agent Programs by Using the Program Interpreter

Sung-Shik T.Q. Jongmans, Koen V. Hindriks, and M. Birna van Riemsdijk

Delft University of Technology

Abstract. Model checking agent programs is a challenge and it is still a question which approaches can suitably be applied to effectively model check such programs. We present a new approach to explicit-state, on-the-fly model checking for agent programs. In this approach we use the agent program interpreter for generating the state space. A model checker is built on top of this interpreter by implementing efficient transformations of temporal properties to Büchi automata and an efficient bookkeeping mechanism that maintains track of states that have been visited. The proposed approach is generic and can be applied to different agent programming frameworks. We evaluate this approach to model checking by comparing it empirically with an approach based on the Maude model checker, and one based on the Agent Infrastructure Layer (AIL) intermediate language in combination with JPF. It turns out that although our approach does not use state-space reduction techniques, it shows significantly improved performance over these approaches. To the best of our knowledge, no such comparisons of approaches to model checking agent programs have been done before.

1 Introduction

Various approaches have been used for model checking agent systems (see, e.g., [1,2,3,4,5,6,7,8]). In this paper, we focus on *explicit-state on-the-fly model checking for agent programming languages.* Current state-of-the-art approaches for model checking agent programs are based on the use of *existing model checkers.* In particular, in [8] and [1] agent programs written in Mable and AgentSpeak(F), respectively, are translated to Promela and verified with SPIN [9]. In the Agent Infrastructure Layer (AIL) project [10] a Java-based framework to which various APLs can be translated is used, in combination with the Java model checker Java Path Finder (JPF) [11]; the model checker is called AJPF. The definition of this translation needs to be specified only once for each AIL interpreted language. Moreover, in [7], an implementation of (a simplified version of) 3APL is presented in the Maude term rewriting language [12]. This enables model checking of 3APL programs with the Maude model checker (MMC) [13]. A possible advantage of such approaches is that built-in optimizations and state space reduction techniques of the existing model checker may be reused for the verification of agent programs.

In this paper, we propose a new approach in which a model checker is built from scratch on top of the interpreter of an agent programming language. Although any model checker needs to rely on an implementation of the semantics of agent programs, our approach differs from others in the sense that it relies on an explicit but abstract interface to an agent program interpreter and evaluation of agent specific conditions are delegated to a standard interpreter for the language whereas temporal properties are handled by well-known techniques for LTL model checking (see also the architecture in Figure 1 below).[1] We have chosen the agent programming language Goal [14] as our target language. One reason for choosing Goal is that the operational semantics of Goal has been implented in Maude, enabling the use of the MMC for Goal, and a translation has been defined to AIL. This facilitates comparison between our approach and other approaches. To the best of our knowledge, no such comparisons of approaches to model checking agent programs have been done before. In this paper, we present an empirical evaluation of these approaches. It turns out that even though our approach does not use state-space reduction techniques, it shows significantly improved performance over these other approaches.

The contribution of this paper is thus twofold: we provide a new approach to model checking agent programs, and a comparison that provides insight into aspects that influence performance when using existing model checkers to model check agent programs.

The rest of the paper is organized as follows. Section 2 introduces some preliminaries. In Section 3, we introduce a new approach to model checking that is based on using the interpreter for an agent programming language itself during the verification of an agent program. We have implemented an interpreter-based model checker for the language Goal, and briefly discuss the associated language for specifying properties. Section 4 presents a number of experiments and the results of a comparison between available approaches for model checking Goal agents. Section 5 discusses our findings. The paper is concluded in Section 6.

2 Preliminaries

In this section, we briefly explain model checking, the Goal language, and its property specification language.

Given a model of a system, model checking tests automatically whether this model satisfies a given property (see, for example, [15]). Properties are often specified in a temporal logic such as LTL [16], as we do in this work. A model of a program (consisting of all its possible computations) satisfies an LTL property φ if all computations satisfy the property. The model checking algorithm

[1] The model checker JPF used in an alternative model checking approach discussed in this paper is not built on top of the standard JVM in this sense but relies on a dedicated JVM developed for JPF. Most other model checking approaches require a translation to a specific language supported by the model checker (such as Maude). Our approach does not require such a translation.

searches for a counterexample, i.e. a computation on which $\neg\varphi$ is true; if such a computation cannot be found, the model satisfies the property φ.

A GOAL agent decides which action to perform next based on its beliefs and goals. The beliefs (collectively called the *belief base*) typically represent the current state of the agent's environment. The GOAL interpreter also offers the possibility to specify knowledge, which is general knowledge about the environment that is typically static. In the interpreter for GOAL, both knowledge base and belief base are Prolog programs. A decision to act will usually also depend on the goals of the agent. Goals of an agent are stored in a *goal base*. The goal base consists of conjunctions of Prolog atoms. Together, the beliefs and goals make up an agent's *mental state*. To make decisions, a GOAL agent uses so-called *action rules* which consist of a mental state condition used to inspect the agent's beliefs and goals, and an action that may be executed if the mental state condition holds. Actions include constructs for changing the agents' beliefs as well as goals; as we focus here on single agents communication primitives are not allowed. Action rules can be combined into so-called modules to provide additional structure to an agent program. To be precise, mental state conditions ϕ are built from *mental atoms* $\mathbf{B}\psi$ and $\mathbf{G}\psi$ as follows:

$$\begin{aligned}
\chi &::= \text{first-order atoms} \\
\psi &::= \chi \mid \neg\chi \mid \chi \wedge \chi \\
\phi &::= \mathbf{B}\psi \mid \mathbf{G}\psi \mid \neg\phi \mid \phi \wedge \phi
\end{aligned}$$

Informally, $\mathbf{B}\psi$ is true if ψ follows from the belief base of the agent, and $\mathbf{G}\psi$ is true if ψ follows from the goal base. For example, we have $\mathbf{G}(p)$ in a mental state with goal base $\Gamma = \{p \wedge q\}$; due to space limitations, we cannot provide all the details here but refer the reader to [17] and remark that $\mathbf{G}$ refers to the primitive goal operator discussed in detail in [17]. A GOAL computation $t = m_0, a_0, m_1, a_1, \cdots$ is an infinite sequence of mental states m_i and actions a_i such that execution of a_i in m_i brings about m_{i+1}, and m_0 is the initial mental state. The meaning of a GOAL program is defined as the set of all its possible computations. More details about the program constructs the GOAL language supports and are supported by the model checker as well can be found in [17].

Although the model checking approach presented in this paper is able to handle programs with all features mentioned above, not all of these features could be used in the comparative experiments presented in this paper since various of these features are not supported by the translation of GOAL to AIL, nor by the implementation of the GOAL semantics in Maude. We recognize that in principle it is possible to extend the model checking approaches based on AIL and Maude to include these features. In this paper, however, we focus in particular on the performance of these approaches for the core subset of GOAL that is supported by all three of approaches discussed here. The basic assumption here is that if for a subset of the GOAL language we can already show significant performance differences, then it is unlikely that extensions to the full GOAL language will perform much better.

A simple GOAL program for solving a blocks world [18] tower building problem is given in Table 1. Blocks are represented using the predicate `block(X)`, the fact that a block X is stacked on another block Y is represented as on(X,Y), and the fact that there is no block on top of a block X is represented as clear(X). This program assumes a single tower of blocks labelled as aa,ab,ac etc. and stacked in this order where aa is the bottom block. The agent moves all blocks to the table, i.e., when block ab is moved to the table, the tower has been unstacked. For this reason, the agent has the goal of having ab on the table.[2]

```
main: agent {
  beliefs {
      block(aa). block(ab). block(ac). block(ad). block(ae).
      on(aa,table). on(ab,aa). on(ac,ab). on(ad,ac). clear(ad).
      on(ae,table). clear(ae).
  }
  goals { on(ab,table). }
  program {
      if goal(on(ab,table)), bel(on(X,Y)), bel(clear(X))
      then moveXfromYtoTable(X,Y) .
  }
  action-spec {
      moveXfromYtoTable(X,Y) {
         pre { block(X) }
         post { not(on(X,Y)), on(X,table), clear(Y) }
      }
  }
}
```

Table 1. Simple GOAL program for the blocks world

The language used here to specify *properties of a* GOAL *program* is LTL, where the propositional atoms are mental atoms defined above.

3 An Interpreter-Based Model Checker

Figure 1 provides a graphical representation of an architecture for an interpreter-based model checker (IMC). The IMC architecture consists of two main components. The first component translates the negation of an LTL formula from the property language to a Büchi automaton, thus representing the property state space. (Recall that mental atoms are treated as propositional atoms in this component.) The formula translator that has been implemented is based on the LTL2AUT algorithm of [19]. The second component evaluates the property by means of a search of the product state space. The product state space is the product of the property state space and the program state space. The component implements the generalized nested depth-first search algorithm of [20], an *on-the-fly* exploration algorithm. This means that only parts of the product state space that are needed are actually generated. The program state space is obtained by means of the agent program interpreter which explains why we have labeled our approach interpreter-based.

[2] In GOAL, a conjunctive goal may be used to express that a set of blocks should be on the table, but the AIL translation does not allow the use of conjunctive goals.

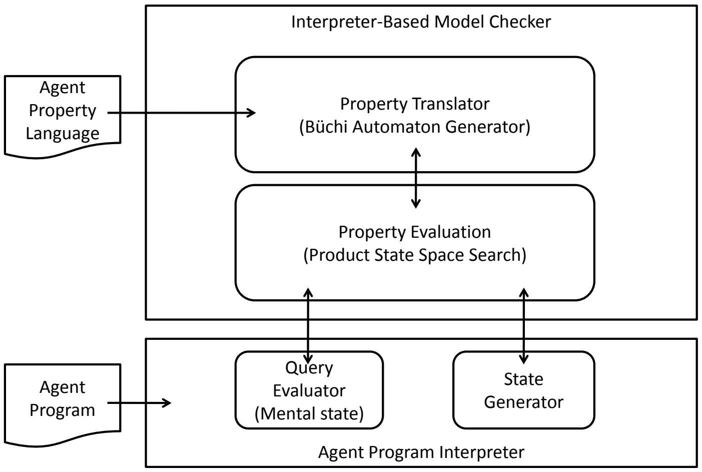

Interpreter-Based Model Checker
Agent Property Language
Property Translator (Büchi Automaton Generator)
Property Evaluation (Product State Space Search)
Agent Program
Query Evaluator (Mental state)
State Generator
Agent Program Interpreter

Fig. 1. Architecture of an Interpreter-Based Model Checker

We have created a model checker for the agent language GOAL by plugging in the interpreter of GOAL and using the mental atoms of the GOAL language as atoms in the property language. This means that the mental state condition evaluation is delegated to the interpreter for the agent language itself and handled by the query evaluator that is part of the agent interpreter whereas temporal properties are represented by a Büchi automaton (see also Figure 1). In a similar way IMCs for other agent languages can be obtained by plugging in an interpreter for those languages and instantiating the atoms of the property language accordingly. For example, as Jason uses the construct ? for inspecting an agent's beliefs and ! for inspecting an agent's event base, these operators could be used instead of the **B** and **G** operators that are part of GOAL (or similar mappings as those proposed in [1] as long as the interpreter provides support to evaluate such conditions). So, even though the main IMC components have been built from scratch, this effort is not dedicated to a single agent programming language.

To be able to support various agent languages a well-defined interface is needed from the IMC to the interpreter of a specific agent language. The interface for any IMC needs to provide support for two types of requests from the exploration component: requests for supplying successors states (given the current state and the agent program that is being checked), and requests for evaluating mental atoms in the mental state that is currently being examined. Both requests are handled by invoking existing methods of the interpreter. One advantage of the IMC approach is that the full support offered by the interpreter can be reused. Existing model checkers for APLs sometimes limit the expressiveness of the property language: the logic of [3], for example, only allows ground atomic formulas inside mental literals. In contrast, because the query mechanism of the GOAL interpreter is used, *any* mental atoms that this interpreter can handle can be used as "atoms" in the property language, including conditions that contain free variables, the use of conjunctions and negations in mental atoms, and the use of knowledge rules. Another benefit of using an interface as described is that it only requires support for generating successor states and handling queries, and the approach abstracts from more specific differences between agent languages related to the precise set of built-in actions that is supported. The point is that although specific and concrete actions are needed in agent languages to compute successor states, the model checking approach presented here only needs to know the successor states but not the manner in which these were generated. The latter is delegated to the agent interpreter itself.

In addition, it is always possible to refine the IMC interpreter and add specific optimizations to make the interpreter more efficient. In our current implementation for GOAL we have implemented a translation of mental states to binary representations in order to more efficiently use memory resources. This representation of GOAL mental states increases the performance of the model checker. It is more costly to perform frequently used operations such as checking for equality of states and computing hash codes for states when the mental state representation that the interpreter manipulates is used instead of the binary representation. These operations need to be performed to check whether states have already been visited and to check for cycles in the search. We briefly explain the details of this representation for GOAL. Informally, every bit (having a unique index) in the binary representation of a mental state corresponds to a belief or a goal: 1-bits indicate that the corresponding belief or goal is part of the mental state, whereas 0-bits indicate the opposite. To translate binary representations of mental states back and forth, we need a bookkeeping mechanism that associates each indexed bit with a unique belief or goal. Since it is infeasible to compute beforehand which beliefs and goals an agent might have in a computation, assigning indices to beliefs and goals is done during model checking. This is achieved by *dynamically* identifying beliefs and goals that have not occurred so far, and assigning indices incrementally, starting from 0. This means that *at runtime* any beliefs or goals that have not occurred so far during execution are added to a list of beliefs and goals that have occurred so far and associated with an integer index (the position in the bit string); there thus is no need to know beforehand which beliefs or goals will occur. The benefit is that instead of using the explicit representation of beliefs and goals we can now use bit strings instead to represent mental states in the state space that needs to be searched. Using bit strings provides two benefits: it reduces memory (space) requirements but most importantly it reduces time needed to compare states (in order to check whether a state has already been visited).[3]

Due to space limitations we are not able to provide all the details here. In essence, the bit string representation of a mental state is a conversion of a mental state to a "canonical representation" of that state. The main point here is that

[3] Note that as knowledge (rules) are assumed to be static we do not need to represent these as part of the state space because they never change. Most, if not all, agent programming languages similar to GOAL assume rules to be static.

if e.g. a belief or goal formula occurs in a mental state the presence of that formula can be checked at a unique index of the bit string. This means that whenever the state space exploration algorithm needs to verify that a mental state has already been visited it does not need to transform visited states into such a canonical representation first (e.g. by sorting the formulas in that state) in order to compare it with the new state to be checked. This involves a significant reduction of time needed to check the visited lists during state space exploration (which yields a time reduction approximately in the order of $N \times |m| \times log(|m|)$ where N denotes the length of the visited list and $|m|$ the average size of a mental state for each time that a visited check needs to be performed). To give an indication of the space reduction, we briefly discuss the more specific issue related to storing ground belief atoms only. Suppose that for explicit representation of such atoms we would use a string representation. A reasonable measure of size needed in that case would be the length of that representation, i.e. string length (in terms of bytes). For the sake of argument, let us assume that we can approximate space requirements by the average length of such strings and on average we would need L bytes to store a belief (this is an underestimate of what is really needed). Moreover, suppose we have N different belief atoms which may or may not occur in a belief base. This yields 2^N possible belief bases. The average number of beliefs in these belief bases is $\Sigma_0^N k \times \binom{N}{k}$ divided by 2^N, which equals $N/2$. Explicit state space representation (using only belief bases) would thus require $N \times 2^{N-1} \times L$ bytes, or $N \times 2^N \times 4 \times L$ bits. Instead the bit string representation would require $N \times L$ bytes to represent the list of belief atoms and $N \times 2^N$ bits to represent the state space. This is a conservative estimate of the space reduction, which shows that minimally space requirements are reduced with a factor $4 \times L$.

4 Experiments and Results

As we view our approach as only one alternative to others, it is important to evaluate our approach and compare it with existing alternatives. The key measure for comparison is performance as model checking of reasonably complex agent programs is only possible if performance is adequate. Our implementation of an IMC for Goal enables a comparison between three approaches. The IMC for Goal can be compared with AJPF (the AIL project model checker) and the MMC (based on Maude) that both provide a tool to model check Goal agents. In order to gain a better understanding of the differences between these approaches we present a number of experiments and corresponding results.

4.1 Experimental Evaluation

Before introducing the experiments that we performed, we first discuss a number of issues related to evaluating the results of our experiments. One issue concerns the input that is provided to the model checkers to obtain a fair comparison. A second issue concerns what it means to say that a model checker *scales well.* We use regression analysis to obtain resource consumption functions (time, space) that fit the data. A third issue concerns the interpretation of the data obtained from the experiments.

Semantic Equality of Agents. The experiments are designed to enable a fair comparison as much as possible. The most important condition for a fair comparison is that semantically equivalent GOAL programs are provided as input to the three different model checkers. Unfortunately, as AJPF and MMC implement different subsets of GOAL, it turned out to be rather complex task to design experiments. For example, in AJPF, goals must be (negations of) single terms. As a result, the experimental agents used in the experiments are simple and are artificial in various respects. IMC is based on GOAL's interpreter and as such does not pose any restrictions, which illustrates another advantage of the interpreter-based approach introduced here; although there is no principled reason to assume other approaches cannot be extended to cover additional language features, the approach presented here provides an alternative that is able to support checking agent programs that use almost any feature supported by the agent interpreter (see also the discussion of the IMC interface above).

Apart from language support considerations there is a second reason for keeping the experiments simple. We first need to evaluate the performance of the model checkers for simple cases before providing more complex programs to check. This has also motivated us to use deterministic agents, i.e. agents that have only one possible computation, in our experiments. The relevant literature provides indications that it is already hard to model check simple programs due to performance reasons and these findings are confirmed by our experiments. We found that even simple non-deterministic agents were beyond the capabilities of MMC, and to a lesser extent this also is the case for AJPF. Another reason for our choices in this regard are that by only considering deterministic agents, it is easier to draw conclusions about resource consumption during the model checking of agents.

Scalability. In order to be able to model check reasonably sized agent programs a model checker needs to be scalable. To avoid confusion, it is important to more precisely define when a model checker is said to scale well. Though improving scalability has been an important factor in research on software verification, to the best of our knowledge, no standards or metrics regarding scalability have been proposed for model checkers. In order to clarify this notion we introduce the following definition.

Definition 1 (Scalability and Unscalability). *A model checker is said to be* scalable *or* scale well *with regard to certain conditions, if the relation between those conditions and resource consumption of the model checker can be described by one of the following functions:*

$$\begin{array}{ll} \text{Logarithmic} & : y = b \cdot \log(x) + a \\ \text{Polynomial } (d < 1) & : y = b \cdot x^d \\ \text{Linear} & : y = b \cdot x + a \end{array}$$

A model checker is said to be not scalable *or* scale poor *with regard to certain conditions, if the relation between those conditions and resource consumption of the model checker can be described by one of the following functions:*

$$\begin{array}{ll} \text{Polynomial } (d > 1) & : y = b \cdot x^d \\ \text{Exponential} & : y = b \cdot r^x \end{array}$$

In these functions, x and y represents, respectively, the conditions and the resource consumption, a is called the intercept*, b is called the* coefficient*, d is called the* degree *of the polynomial, and r is called the* radix..

The intuition behind Definition 1 is that a model checker scales well if the relation between conditions and consumption is at most linear. The reason for regarding polynomial relations in degree $d > 1$ as not scalable is that x is, in general, very large for real-world model checking problems. In such cases, a quadratic relation versus a cubic relation can already make the difference between (in)tractability of a problem. Note that our definition differs from complexity theory, where all problems that can be solved in polynomial time are deemed tractable.

Regression analysis. To determine the type of relation between experimental conditions and resource consumption, we will apply regression analysis to analyze the experimental data. That is, we will fit the functions given in Definition 1 to the measurements using least-squares regression, and assess how good the fits found are by comparing their R^2 values. The fitted function that yields the highest (i.e. closest to 1) R^2 is deemed the relation between conditions and consumption. The resource consumption of the model checkers is obtained by measuring two *dependent variables*: verification time and memory consumption.

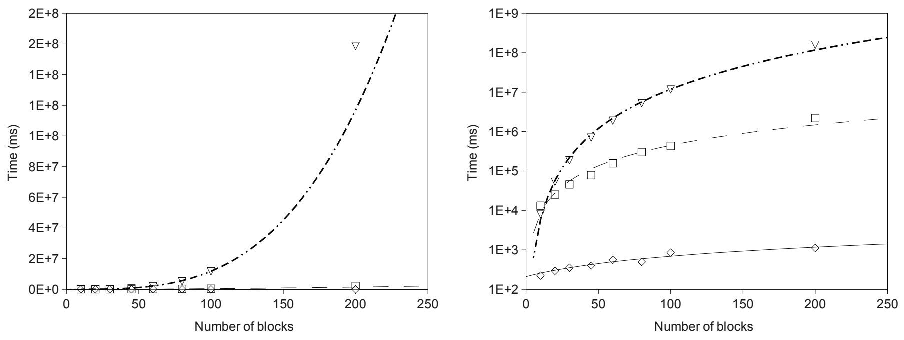
The conditions under investigation, in statistics called *independent variables*, are the size of the belief base and the size of the state space. The size of the state space is defined as the total number of mental states that can be encountered on all computations of the Goal agent. The size of a belief base is defined as the number of elements it contains (i.e. ground atoms). In the experiments reported on below, we simply used the model checkers to report the number of states visited.

To study the different effects of both independent variables, the experiments are organized as follows. In the first experiment, the size of the belief base is varied, while the size of the state space is kept constant. In the second experiment, the size of the belief base is kept constant, while the size of the state space is varied. Finally, in the third experiment, both the size of the belief base and state space are varied.

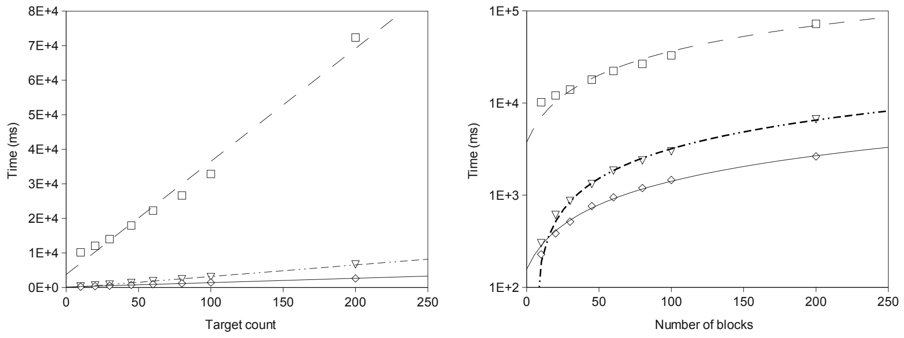
4.2 Experiments

We now present the three experiments that we have performed (together with the experimental results).

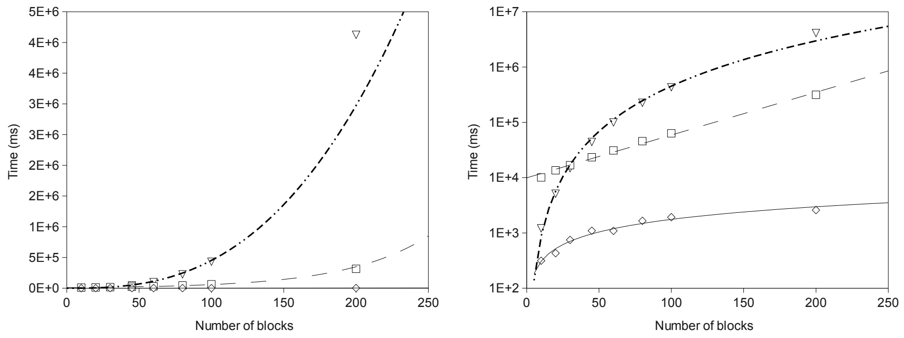
Experiment 1 (Size of Belief Base). In this first experiment, we investigate the scalability of the model checkers in the size of the belief base; the size of the state space is kept constant.



(a) Experiment 1.



(b) Experiment 2.



(c) Experiment 3.

Fig. 2. Verification times of IMC (continuous line, measurements as ◇), AJPF (dashed line, measurements as □), and MMC (dashed-dotted line, measurements as ▽) in Experiments 1, 2, and 3. Plotted lines are best-fit regression lines. Left figures have a linear scale on the y-axis, whereas the scale on the y-axis of right figures is logarithmic.

For this experiment, we have used variants of the agent program of Table 1 with $n \in \{10, 20, 30, 45, 60, 80, 100, 200\}$ blocks, rather than $n = 5$ as used in the agent program. Four of the n blocks are initially stacked on each other, whereas the remaining $n - 4$ blocks are on the table. For all n, the stacked blocks are aa, ab, ac, and ad: aa is on the table, ab is on aa, ac is on ab, and ad is on ac. In the target configuration, all the blocks are on the table. The property φ under investigation is whether the agent eventually brings about the target configuration. For all values of n, the program state space contains only four mental states. In contrast, the belief base grows as n increases: it becomes filled with *redundant* beliefs. That is, removing these beliefs would not affect the behaviour of the agent.

The verification times are displayed in Fig. 2a, and the calculated relations are given in the first column of Table 2a. These results suggest that IMC scales well to larger belief bases, in contrast to AJPF and MMC. Though the verification times of the latter two both grow polynomially in the size of the belief base, the degrees of the fitted functions (shown in Table 2a between brackets) show that the increase in verification time of MMC is more than cubic, whereas AJPF remains under quadratic. This difference can also be observed in Fig. 2a, and supports our decision to classify polynomial relations in degree $d > 1$ as not scalable. For $n = 200$, the absolute difference in performance is the largest: IMC took 1 second, AJPF took 40 minutes, while MMC took 44 hours.

The memory consumption is displayed in Fig. 3a, and the calculated relations are given in the first column of Table 2b. One might notice that although the degrees of the fitted polynomial functions for IMC and AJPF are the same, the memory requirements of the former are much lower. The reason for this is that the coefficient b (see Definition 1) for IMC is roughly 2, whereas for AJPF it is roughly 22. According to Definition 1, all three model checkers scale well to larger belief bases with respect to memory consumption. Nevertheless, AJPF is substantially more memory demanding than IMC and MMC.

Experiment 2 (Size of State Space). In this second experiment, we investigate the scalability of the model checkers in the size of the state space; the size of the belief base is kept constant. To reduce the effect of the size of the belief base on the experimental results (the previous experiment showed that such an effect is definitely present), it should be as small as possible. As these requirements (small belief base, growing state space) are not easily satisfied by a Blocks World scenario, we use the following setting.

In this experiment, the agent is a simple counter: it starts at 0, and counts until infinity. There are various ways to implement this behaviour, and we chose an implementation in which: the belief base is used as little as possible (for reasons outlined above), and goals are issued very frequently. The reason for issuing many goals is that goal creation is a relatively slow operation in the current GOAL interpreter. Thus, by issuing many goals, not only do we challenge the interpreter, but the interpreter-based model checker as well. Note that because the agent can count until infinity, the state space of the agent is not finite. To ensure that the model checking procedure is decidable, the property φ must be verifiable in a finite number of interpretation cycles. One such property is that the agent eventually believes that its current number is some natural number n. Hence, the size of the state space is controlled by the value of n in φ; in this experiment, we chose $n \in \{10, 20, 30, 45, 60, 80, 100, 200\}$.

The verification times are displayed in Fig. 2b, and the calculated relations are given in the second column of Table 2a. Though we expected MMC to be the slowest of the three (based on its performance in Experiment 1), AJPF is in fact ten times slower: the slope of the linear function fitted on the AJPF measurements is roughly 330, whereas the slope of MMC's linear fit is only 33. For $n = 200$, the difference is the largest: IMC took 3 seconds, AJPF took 72 seconds, and MMC took 7 seconds.

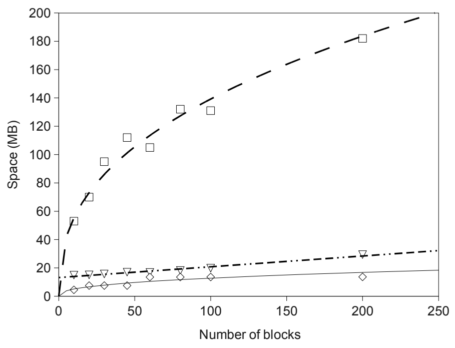
The memory consumption is displayed in Fig. 3b, and the calculated relations are given in the second column of Table 2b. Similar to Experiment 1, all reported relations are scalable according to Definition 1, but again, AJPF demands substantially more memory. Also, the relations for IMC and MMC imply that MMC's memory consumption grows faster in the size of the state space than that of IMC. Hence, it is to be expected that for some $n > 200$, IMC will consume less memory than MMC. This is not obvious from Fig. 3b.

Experiment 3 (Size of State Space and Belief Base). In the third experiment, we investigate the scalability of the model checkers with respect to both the size of the belief base and the state space.

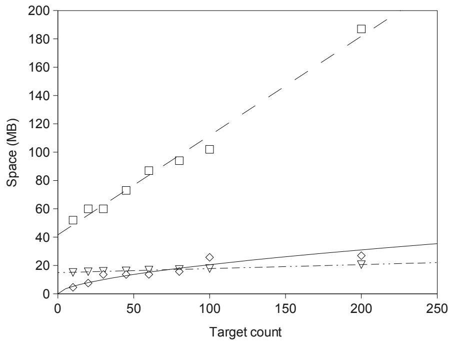
To satisfy the desired experimental conditions (growing belief base and state-space), we adapt the agent of Experiment 2 in such a way that it remembers all counted numbers. As a consequence, the size of the belief base will increase linearly in the size of the state space. These beliefs are, like the superfluous blocks in Experiment 1, redundant. The property under investigation is the same as in Experiment 2 for the same values of n such that all three model checkers terminate eventually.

The verification times are displayed in Fig. 2c, and the calculated relations are given in the third column of Table 2a. IMC is again the fastest of the three model checkers, and still shows to scale well. In contrast, scalability of AJPF and MMC drops from linear to exponential and polynomial in degree 2.7, respectively. For $n = 200$, the absolute difference in performance is the largest: IMC took 3 seconds, AJPF took 5 minutes, while MMC took over an hours.

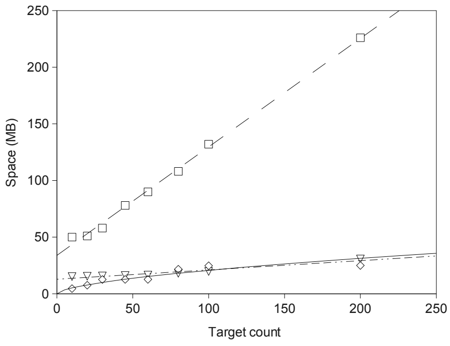
The memory consumption is displayed in Fig. 3c, and the calculated relations are given in the third column of Table 2b. The memory demands are similar to those in Experiments 2: AJPF performs, though depending linearly on the experimental conditions (thus, scalable in terms of Definition 1), the least well of the three model checkers, whereas IMC and MMC perform roughly equal. It is interesting to see that the intersection point of IMC and MMC for some $n > 200$, mentioned when treating the memory consumption of the model checkers in Experiment 2, is almost within the range of the values of n in Experiment 3.



(a) Experiment 1.



(b) Experiment 2.



(c) Experiment 3.

Fig. 3. Memory consumption of IMC (continuous line, measurements as $\diamond$), AJPF (dashed line, measurements as $\square$), and MMC (dashed-dotted line, measurements as $\triangledown$) in Experiments 1, 2, and 3. Plotted lines are best-fit regression lines.

Summary. Table 2a summarizes the results with respect to verification time. IMC is the only model checker that scaled well in all three experiments. The other two model checkers clearly have problems when the size of the belief base increases. Table 2b summarizes the results with respect to memory consumption. With only polynomial relations in degree $d < 1$, IMC has performed best with regard to memory consumption. Though the measurement for AJPF and MMC imply scalability as well, Fig. 3 shows that there still are clear differences between IMC and MMC on the one hand, and AJPF on the other.

Comparison with Normal Execution. In order to distinguish between overhead caused by the model checker and possible inefficiency caused by the underlying execution mechanism, we have measured resource consumption during execution (rather than verification) of the agents by the three underlying platforms as well. This can be done because the programs are deterministic, and

Table 2. Relations between resource consumption and experimental conditions

(a) Verification time. Boldface shows scalability.

	Experiment 1		Experiment 2		Experiment 3	
	Relation	R^2	Relation	R^2	Relation	R^2
IMC	**Linear**	0.9329	**Linear**	0.9974	**Poly** (0.75)	0.9722
AJPF	Poly (1.7)	0.9706	**Linear**	0.9829	Exponential	0.9925
MMC	Poly (3.3)	0.9958	**Linear**	0.9985	Poly (2.7)	0.9936

(b) Memory consumption. Boldface shows scalability.

	Experiment 1		Experiment 2		Experiment 3	
	Relation	R^2	Relation	R^2	Relation	R^2
IMC	**Poly** (0.40)	0.8266	**Poly** (0.59)	0.9109	**Poly** (0.60)	0.9113
AJPF	**Poly** (0.40)	0.9663	**Linear**	0.9860	**Linear**	0.9966
MMC	**Linear**	0.9664	**Linear**	0.9982	**Linear**	0.9491

therefore the trace that is model checked coincides with the trace generated by executing the programs.

In case of the program of Experiment 1, we observed that both AJPF and MMC introduce substantial overhead with respect to run-time: the largest difference between execution and verification times, measured for $n = 200$, amounted to roughly 37 minutes for AJPF, and over 41 hours for MMC. In contrast, the difference measured for IMC was only 50 milliseconds. With respect to memory consumption, less extreme differences were measured: for $n = 200$, IMC, AJPF and MMC required, respectively, 10 MB, 125 MB, and 2 MB less than when model checking.

In case of the program of Experiment 2, we observed that the overhead of MMC during verification is a lot smaller than in Experiment 1: the largest difference, measured for $n = 200$, is only 2 seconds (compared to 41 hours in Experiment 1). For AJPF, the difference is again large: roughly 70 minutes. For IMC, the largest difference is still negligible: approximately half a second. With respect to memory consumption, larger differences were measured than for Experiment 1: for $n = 200$, executing the agent took IMC, AJPF, and MMC, respectively, 24MB, 172MB, and 6MB less than when model checking the agent.

In case of the program of Experiment 3, we observed that the overhead of AJPF with respect to run-time is, as in the previous experiments, substantial. For MMC, in contrast to Experiment 2, the overhead is significant as well. For $n = 200$, the difference between execution and verification times for IMC, AJPF, and MMC are 1 second, over 5 minutes, and 24 minutes, respectively. With respect to memory consumption, again larger differences were measured: for $n = 200$, executing the agent took IMC, AJPF, and MMC, respectively, 22 MB, 203 MB, and 16 MB less than when model checking the agent.

4.3 Wumpus Scenario

In order to illustrate that IMC is able to model check larger agent programs which give rise to much larger state spaces, we present results about model checking an agent for the well-known Wumpus World scenario [21]. The primary motivation for presenting this domain is to show IMC is able to handle more realistic scenarios. We are not able to present results for this domain for AJPF or MMC. The main reason is that even for small instances of this domain the state space already is significantly bigger than those used in the previous experiments, and by extrapolating the results obtained above it is unlikely to obtain results for either AJPF or Maude within reasonable time.

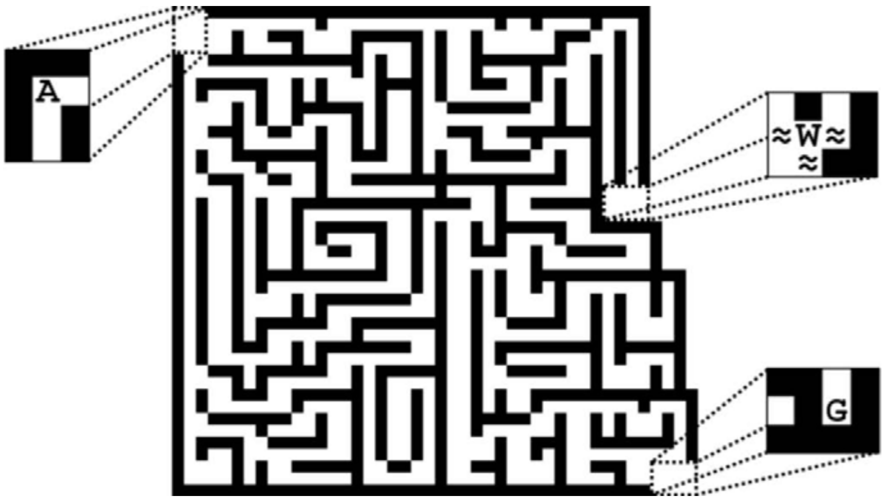


Fig. 4. Wumpus World

In the Wumpus World, a single agent is located in a cave that contains pits which need to be avoided, walls that prevent movement, and a beast called the Wumpus that as pits will kill the agent if it steps onto it. The cave is a grid world and locations can be identified by x and y coordinates. Figure 4 illustrates the environment; A denotes the agent, G represents gold, and W represents the Wumpus (the grids around the Wumpus are marked so the agent can smell it is next to the Wumpus; similarly pits are marked by 'breezes'). The goal of the agent is to locate gold that resides somewhere (at a position initially unknown to the agent, the environment is partially observable) and after getting the gold leaving the cave. We can model check this environment as it is a single agent and deterministic environment which ensures we can always determine a unique set of successor states for each action that is performed by the agent; for more details see [21].

The first agent that we verified moves through the cave completely non-deterministically. It bumps into a wall if a wall blocks its way, and returns to a previously visited position if it encounters a stench (indicating that the Wumpus is close by). We checked a property that specifies that, given this Wumpus-avoidance-policy of the agent, it can never be at the same position as the Wumpus: $\Box\neg$**B**`position(33,-11)`. As the position of the Wumpus is fixed (at `(33,-11)`), satisfaction of this property means that at least the agent will never die. The model checker reports after exploring 57355 states in 6:30 minutes and using 97 MB of memory that the property is true. Another property that we would like the agent to satisfy is that it eventually obtains the gold: $\Diamond$**B**`has(gold)`. Unfortunately, the model checker reports a (non-minimal) counterexample after exploring 89 states in 2 seconds and using 42 MB of memory: at some point, the agent enters a loop of turning left, moving forward three steps, turning left twice, moving forward three steps again, and turning left again. We can, however, establish that there exists at least one computation on which the agent does obtain the gold by verifying the property $\Box\neg$**B**`has(gold)`. The model checker reports a counterexample in 2:19 minutes and uses 36 MB of memory; this counterexample corresponds to the computation on which the agent obtains the gold.

The second agent that we verified is a lot smarter than the first: it maintains a mental map of the cave by remembering the positions it visited, including a “trail of breadcrumbs” to find its way outside efficiently, and systematically explores unknown grounds. This implementation removes the non-determinism from the agent (making the state space smaller), but because a lot of information must be stored, the belief base is much larger: it grows linearly as the agent explores the cave, which has over 1600 different positions. The Wumpus-avoidance-policy of this agent is the same as that of the first, and the model checker re-confirms its effectiveness after exploration of 8225 states in 55 seconds using 37 MB of memory. As a result the agent satisfies $\Diamond$**B**`has(gold)`: verification required exploration of 3877 states, took 48 seconds and required 30 MB of memory.

5 Discussion

The experimental results clearly show that IMC outperforms the other two model checkers. Also, the results show that especially MMC is unable to deal with the simple toy examples that were under investigation, particularly with regard to verification time; to a lesser extent, this is also true for AJPF. With regard to AJPF, we believe that for a large part, the overhead of JPF is responsible for the slow verification (as well as for higher memory consumption), because executing (rather than verifying) the agent is substantially faster: the agent of Experiment 3 easily counts to 200 within a few seconds, whereas verifying with AJPF whether this agent actually can count to 200 takes over 5 minutes. Similar differences were observed in all three experiments. With regard to MMC, the slow verification is partly ascribed to the rate at which the Maude Goal interpreter can generate the state space: the agent of Experiment 3 already takes 45 minutes to count to 200 (when executed). Note, however, that the overhead of Maude’s built-in model checker can be substantial as well: verifying whether the agent can actually count to 200 takes 24 more minutes.

Earlier, we mentioned that model checking non-deterministic agents is infeasible with AJPF and MMC: we carried out an additional experiment with IMC featuring a non-deterministic agent to illustrate this. Consider a Blocks World agent as in Experiment 1 with an initial belief base containing 200 blocks divided over 2 towers of 100 blocks each, and a property specifying that the target configuration (all blocks on the table) is reached. This non-deterministic agent has a state space of size 10,000. Nevertheless, verification with IMC takes only 2150 seconds, i.e. 35 minutes. In contrast, AJPF and MMC required already more time (40 minutes and 44 hours, respectively) to complete verification for $n = 200$ in Experiment 1: a comparable setting with only 4 states instead of 10,000. Given that the size of the belief base is constant, and assuming that AJPF and MMC scale linearly in the size of the state space (as suggested by the results of Experiment 2), it would take AJPF 100,000 minutes, i.e. 70 days, and MMC 110,000 hours, i.e. 12.5 *years*, to terminate.

Another observation concerns the size of the belief base: it turns out this has a large impact on the verification times associated with AJPF and MMC. It follows that the performance of these model checkers is not only dependent on the size of the state space, but also on the way that beliefs are dealt with. We speculate that two important aspects need to be optimized to increase performance. First, the mechanism for querying the belief base should be implemented as efficiently as possible. For MMC, this seems a problem as normal execution of the agents already took a long time in Experiments 1 and 3 (in Experiment 2, belief base queries were only performed on a belief base with at most two beliefs). Second, the model checker should not introduce overhead on this querying, which seems to be the case for AJPF.

6 Conclusion

We have distinguished three different approaches to model checking agent programs: two approaches that reuse existing model checkers in quite different ways, including for example [1] and the AIL approach [10], and the interpreter-based approach that we introduced in this paper. An architecture for this new approach has been introduced and we discussed how the approach can be applied to various agent programming languages. An implementation for the GOAL agent language has also been provided. One advantage of the interpreter-based approach is that in practice it supports a more expressive property language than that supported by currently existing alternatives.

In order to compare these approaches we performed various experiments to gain insight into the performance of these approaches. As far as we know, such performance comparisons have not been made before and providing such results is one of the contributions of this paper. Our main finding is that the interpreter-based approach outperforms the other two approaches. In addition, the computed relations with respect to resource consumption show that model checking non-deterministic agents (or, in general, any agent with a state space that is orders of magnitudes larger than, for example, the simple Blocks World examples we used) is currently beyond the capabilities of AJPF and MMC. In contrast, IMC handles those state spaces with relative ease.

We plan on further developing the interpreter-based Goal model checker, in particular by extending it with state space reduction techniques. As we have full control over de code of the model checker, we expect that implementing such techniques is, from a programming point of view, less complex than when such language-specific optimizations would need be incorporated in an existing model checker.

References

1. Bordini, R.H., Fisher, M., Pardavila, C., Wooldridge, M.: Model checking agentspeak. In: Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 409–416. ACM, New York (2003)
2. Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: Verifiable multi-agent programs. In: Dastani, M.M., Dix, J., El Fallah-Seghrouchni, A. (eds.) PROMAS 2003. LNCS (LNAI), vol. 3067, pp. 72–89. Springer, Heidelberg (2004)
3. Bordini, R.H., Dennis, L.A., Farwer, B., Fisher, M.: Automated verificiation of multi-agent programs. In: 23rd IEEE/ACM International Conference on Automated Software Engineering, pp. 69–78. IEEE Computer Society, Los Alamitos (2008)
4. Dennis, L.A., Fisher, M.: Programming verifiable heterogeneous agent systems. In: Hindriks, K.V., Pokahr, A., Sardina, S. (eds.) ProMAS 2008. LNCS, vol. 5442, pp. 40–55. Springer, Heidelberg (2009)
5. Kacprzak, M., Nabialek, W., Niewiadomski, A., Penczek, W., Pólrola, A., Szreter, M., Wozna, B., Zbrzezny, A.: Verics 2007 - a model checker for knowledge and real-time. Fundamenta Informaticae 85, 313–328 (2008)
6. Lomuscio, A., Raimondi, F.: Mcmas: a model checker for multi-agent systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 450–454. Springer, Heidelberg (2006)
7. van Riemsdijk, M.B., de Boer, F.S., Dastani, M., Meyer, J.J.C.: Prototyping 3apl in the maude term rewriting language. In: Inoue, K., Satoh, K., Toni, F. (eds.) CLIMA 2006. LNCS (LNAI), vol. 4371, pp. 95–114. Springer, Heidelberg (2007)
8. Wooldridge, M., Fisher, M., Huget, M.P., Parsons, S.: Model checking multi-agent systems with mable. In: Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 952–959. ACM, New York (2002)
9. Holzmann, G.J.: The SPIN model checker. Addison-Wesley, Reading (2003)
10. Dennis, L.A., Farwer, B., Bordini, R.H., Fisher, M., Wooldridge, M.: A common semantic basis for bdi languages. In: Dastani, M.M., El Fallah Seghrouchni, A., Ricci, A., Winikoff, M. (eds.) ProMAS 2007. LNCS (LNAI), vol. 4908, pp. 124–139. Springer, Heidelberg (2008)
11. Visser, W., Havelund, K., Brat, G., Park, S., Lerda, F.: Model checking programs. Automated Software Engineering 10, 203–232 (2004)
12. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: specification and programming in rewriting logic. Theoretical Computer Science 285, 187–243 (2002)
13. Eker, S., Meseguer, J., Sridharanarayanan, A.: The maude ltl model checker. In: Proceedings of the 4th International Workshop on Rewriting Logic and its Applications, pp. 162–187. Elsevier Science, Amsterdam (2002)

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 (LNAI), vol. 1986, pp. 248–257. Springer, Heidelberg (2001)
15. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. The MIT Press, Cambridge (1999)
16. Emerson, E.: 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)
17. Hindriks, K.V.: Programming rational agents in goal. Multi-Agent Programming, 119–157 (2009)
18. Slaney, J., Thiébaux, S.: Blocks world revisited. Artificial Intelligence 125, 119–153 (2001)
19. Daniele, M., Giunchiglia, F., Vardi, M.Y.: Improved automata generation for linear temporal logic. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 249–260. Springer, Heidelberg (1999)
20. Tauriainen, H.: Nested emptiness search for generalized buchi automata. In: Proceedings of the 4th International Conference on Application of Concurrency to System Design, pp. 165–174 (2004)
21. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, Englewood Cliffs (2003)

A Dialogue Games Framework for the Operational Semantics of Logic Agent-Oriented Languages

Stefania Costantini and Arianna Tocchio

Università degli Studi di L'Aquila
Dipartimento di Informatica
Via Vetoio, Loc. Coppito, I-67010 L'Aquila - Italy
stefania.costantini@univaq.it, arianna.tocchio@gmail.com

Abstract. We propose an approach to the operational semantics aimed at describing the behavior of an interpreter (or, anyway, of an implementation) of agent-oriented logic programming languages. We define a formal dialogue game framework that focuses on rules of dialogue that the players of the game, i.e., the components of the language interpreter, can be assumed to adopt in playing an "innocent" game that finalized to creating the interpreter behavior. The approach leads to a natural, uniform and modular way of modeling all the components of an interpreter, including the communication component and the communication protocol, and allows several properties to be formally proved.

1 Introduction

Logic Programming languages have been since the very beginning characterized by a rigorous definition of their declarative and procedural semantics [25] [2], formally linked to the operational semantics [14], [6]. As agents and multi-agent systems have become an emerging paradigm for programming distributed applications, several agent-oriented languages, formalisms and frameworks based on logic programming and computational logic have been proposed, among which [3], [21], [10], [34], [24], [11]. All of them have been developed with attention to the formal semantics, and to the compliance of the implementation w.r.t. the semantics.

Nevertheless, D'Inverno and Luck in [17] observe that *implementations have typically involved simplifying assumptions that have resulted in the loss of a strong theoretical foundation for them, while logics have had small relation to practical problems. Though this fragmentation into theoretical and practical aspects has been noted, and several efforts made in attempting to address this fragmentation in related areas of agent-oriented systems (for example, [16], [19], [26], [41]), there remains much to be done in bringing together the two strands of work.* An effort in this direction has been made via the formalization of operational semantics in an agent language. This has resulted for instance in the definition of AgentSpeak(L), which can be viewed as an abstraction of the implemented Procedural Reasoning System (PRS) [18] and the Distributed Multi-Agent Reasoning System(dMARS) [15], and which allows agent programs to be defined and interpreted [34].

The effort of providing an abstract formalization of agent behavior is aimed at the application of formal methods in a rigorous definition and analysis of agents functionalities.

This may allow one to demonstrate interesting properties related to system correctness. Bracciali et al. in [9] synthesize the need of being able to prove agent behavior correctness as follows: *the ever-growing use of agents and multi-agent systems in practical applications poses the problem of formally verifying their properties; the idea being that by verifying properties of the overall system we can make informed judgements about the suitability of agents and multi-agent systems in solving problems posed within application domains..*

We can find a similar consideration in [7], where the authors explain the motivations that have persuaded them to describe the AgentSpeak(L) behavior in view of applying Model-Checking techniques: *...tools should be usable by a general computing audience, there should also be strong theoretical foundations for such tools, so that formal methods can be used in the design and implementation processes. In particular, the verification of multi-agent systems showing that a system is correct with respect to its stated requirements is an increasingly important issue, especially as agent.*

Formal properties of agent systems should refer in general to multi-agent contexts, where the correctness of agent interactions assumes a relevant role. In [5] and [4], some interesting properties are demonstrated via AUML. Wooldridge and Lomuscio in [42] present the *VSK* logic, a family of multi-modal logics for reasoning about the information properties of computational agents situated in some environment. Viroli and Omicini in [40] study the impact of thinking about agents and multi-agent systems in terms of their observable behavior. Bracciali et al. in [9] define the semantics of a multi-agent system via a definition of stability on the set of all actions performed by all agents in the system and specify properties of individual success of agents, overall success of a multi-agent system, robustness and world-dependence of a multi-agent system, as well as a number of properties of agents within systems. The work by Bordini and Moreira reported in [8] exploits an operational semantics in order to investigate properties of AgentSpeak(L) agents. The approaches of Alechina et al. [1] and Hindriks at al. [20] propose the adoption of *transition systems* in order to define the operational semantics of Agent-Oriented programming Languages (in the following for short AOL's).

McBurney, Parsons, et al., in [28] [30] [32] [31], study argumentation-based dialogues between agents. They propose dialogue games as a formal framework for dialogues between autonomous agents, The authors discuss how to prove some properties of dialogues under a given protocol, in particular termination, dialogue outcomes, and complexity. Dialogue games are formal interactions between two or more participants, in which participants "move" by uttering statements according to pre-defined rules.

In this paper, we propose dialogue games as the foundation of a general methodology for the definition of the operational semantics of agent-oriented logic programming languages. However, our aim is not that of providing the operational semantics of an AOL, where this operational semantics will have to be implemented by means of some kind of computational device (interpreter, abstract machine, etc.). Instead, given an AOL, we intend to provide the operational definition of its implementation. This definition is a formal specification of the implementation, and allows one to prove properties, first of all the correctness of the implementation w.r.t. the programming language semantics. Therefore, our approach is not in contrast with previous ones, but rather complementary and potentially synergic.

We have applied the approach to the DALI language [11] [12]. In particular, in [39] the behavior of the DALI interpreter has been fully defined as a dialogue game whose players are the various modules composing the interpreter itself. I.e., the players are the components of the interpreter, considered as black boxes, that play an innocent game one towards the other.

A first advantage of the approach is that of being able to describe in a uniform way all aspects involved in processing an agent-oriented language, including communication. A second advantage is modularity: as a language interpreter is seen as composed of modules which are the players of a game, these modules can be composed, added, removed, replaced in flexible ways provided that they respect the "rules" of the game. A further advantage is that one can take profit of game theory for proving properties of the interpreter, among which various properties of correctness. In [39], correctness of the DALI interpreter w.r.t. the declarative and procedural definition of DALI extended resolution is formally proved.

The proposed approach is elaboration-tolerant w.r.t. the adopted communication protocol, that can be specified via a separate set of definitions. To illustrate this point, we show how to define compliance of the DALI interpreter w.r.t. a protocol and in particular we demonstrate compliance w.r.t. a simple protocol introduced by Sadri et al. [36]. Therefore, we claim that our proposal can be the basis of a general methodology for providing an account of the procedural behavior of AOL's, that enables interesting properties to be proved, also in synergy with other semantic definitions.

This paper is organized as follows. In Section 2 we introduce dialogue games. In Section 3 we propose formal dialogue games as a tool for defining the operational semantics of agent-oriented logic programming languages, and introduce to this aim a specific dialogue game framework. In Section 4 we illustrate some examples of laws and rules of the proposed dialogue game, taken from the operational description of the DALI language but general enough to be quite directly applicable to other languages/formalisms. In Section 5 we show how we can abstract aspects of the operational behavior from given laws and rules, so as to define and prove useful properties. Finally, we conclude in Section 6.

2 Dialogue Games as a Formal Framework for Dialogues between Autonomous Agents

Formal dialogue games have been studied in philosophy since at least the time of Aristotle. For a review of their history and applications the reader may refer to [28] [29] and to the references therein. Recently, they have been applied in various contexts in computer science and artificial intelligence, particularly as the basis for interaction between autonomous software agents. Dialogue game protocols have been proposed for agent team formation, persuasion, negotiation over scarce resources, consumer purchase interactions and joint deliberation over a course of action is some situation ([23],[27],[37],[38]).

In particular, formal dialogue games are interactions between two or more players, where each player "moves" by making utterances, according to a defined set of rules.

In this section, we present a model of a generic formal dialogue game as reported in [28]. We assume that the topics of discussion between the players can be represented in some logical language. A dialogue game specification then consists of the following elements:

- **Commencement Rules:** Rules which define the circumstances under which a dialogue commences.
- **Locutions:** Rules which indicate what utterances are permitted.
- **Combination Rules:** Rules which define the dialogical contexts under which particular locutions are permitted or not, or obligatory or not.
- **Commitments:** Rules which define the circumstances under which participants express commitment to a proposition.
- **Termination Rules:** Rules that define the circumstances under which a dialogue ends.

For locutions, we will basically adopt (and illustrate in the following sections) the form proposed in [31]. The definition of a dialogue game is completed by the specification of a *state transition system* composed of transition rules that define how the state of the system (composed of the set of players) changes as a result of a certain locution having been uttered (or, also, as a result of a certain locution having *not* being uttered). Transition rules provide a formal linkage between locutions and their possible usages and effect, and make automated dialogues possible.

Dialogue games differ from the games of economic game theory in that payoffs for winning or losing a game are not considered, and, indeed, the notions of winning and losing are not always applicable to dialogue games.

3 Operational Semantics as a Dialogue Game

We propose formal dialogue games and related transition systems as a tool for defining the operational semantics of agent-oriented logic programming languages.

A state transition system is an abstract machine used in the study of computation. The machine consists of a set of states (or "configurations") and transitions between states, which may be labeled with labels chosen from a set, where the same label may appear on more than one transition. In the operational semantics of programming languages [33], a transition system is a deductive system which allows one to derive the transitions of a program. This kind of transition system consists of a set of transition rules that specify the meaning of each programming construct in the language. Transition rules transform configurations.

In the agent framework, transition systems have been adopted, e.g., in [1] and [20], to provide the operational semantics of Agent-Oriented Languages (AOL's) with particular attention to those based upon the BDI (Belief, Desires, Intention) paradigm, where agents are supposed to have *mental states* that evolve according to their interaction with the environment. In their setting, a configuration is the encoding of a "mental state". Thus, a transition rule maprepresents the evolution of a mental state into another one, by, e.g., executing a goal or matching the head of a practical reasoning rule with a goal. A single transition models the execution of a simple "instruction" (where an instruction can be, e.g., assignment in the case of imperative programming, basic actions and tests in the agent case). The resulting operational semantics allows one to prove properties of an agent program at hand, or more generally properties of the programming language.

Our approach is different from those mentioned above. In fact, our aim is not that of providing an operational semantics for the programming language at hand, that will have to be implemented by some kind of computational device (interpreter, abstract machine, etc.). Instead, given an AOL, we intend to provide the operational definition of such a computational device. This means, we intend to describe what such a device, (in the following, "the interpreter"), does when elaborating an agent program. The resulting operational semantics allows one to prove properties of the interpreter, first of all its correctness w.r.t. the programming language semantics. Other properties that can be proved, as we will show in the following, are for instance that the interpreter correctly implements a certain communication protocol when adopted by the agent. One can also prove properties of a certain program (including termination) when running on the interpreter that has been described. The proposed approach is therefore not in contrast to those, like the above-mentioned ones, aimed at describing the operational semantics of agent-oriented languages: rather, it is complementary and potentially synergic as it allows correctness of the implementation to be formally proved.

Then, in our setting we have a transition system associated to the dialogue game describing the operation of the interpreter of a given AOL. The dialogue game will consist of a set of *Laws*, each one describing an item of the activity of the interpreter. The description is in term of a *locution* (or *utterance*) stating the fragment of an agent activity (in terms of the given AOL) that the interpreter is going to process. Locutions will depend upon both the AOL at hand and the structure of the interpreter. A possible locution can be, e.g., *receive_message*. The law also prescribes the preconditions for that item to be processed, and the post-conditions, i.e., how subsequent interpreter activities are affected. Configurations of this transition system will be states of the interpreter. Labels in this case are laws of the dialogue game. With utterances, the interpreter "declares" to itself what it is going to do. The pre- and post-conditions are related to the semantics and pragmatics of the programming language at hand. Then, each transition rule will describe a change that occurs in the interpreter state when processing a sequence of laws, or, symmetrically, a change that occurs when the interpreter is not able to process a certain law because its preconditions are not fulfilled.

We propose to describe the interpreter behavior not simply in terms of a dialogue game that the interpreter plays with itself, but rather as a dialogue game that the interpreter *components* play with each other. The motivation of this proposal relies in the nature of AOL's, that in affects in general the structure of the interpreter, which can be seen as composed of components (or modules) representing the AOL features. Despite the differences, all agent-oriented languages have (at least) the following basic features:

– A logical "core", that for instance in both KGP and DALI is resolution-based.
– Reactivity, i.e., the capability of managing external stimuli.
– Proactivity, i.e., the capability of managing internal "initiatives".
– The capability of performing actions.

- The capability of recording what has happened and has been done in the past.
- The capability of managing communication with other agents. This can be seen as a composition of sub-capabilities: managing both out-coming and incoming messages according to a given protocol, and possibly applying ontologies to understand message contents.
- A basic cycle that interleaves the application of formerly specified capabilities. E.g., in DALI the basic cycle is integrated with the logical core into an extended resolution, while in KGP the basic cycle has a meta-level definition and thus can be varied.

All these components can be seen as "players" that play together and make "moves", so as to coordinate themselves to generate the overall interpreter behavior. This means, our players participate in an "innocent" game where their objective is not to win, but rather is that of respecting the rules of the game itself. Thus, we expect each player to faithfully follow the laws and rules so as to produce a set of admissible moves. These moves will influence the other players and will determine the global game.

Advantages of the approach are the following:

- Each component can be seen as a "black-box". I.e., the corresponding locutions and transition rules can be defined independently from the rest. Then, capabilities can be easily added/removed, at the expense of modifying only the description of the basic cycle, which should be however by its very nature quite modular.
- There need to be no different formalism for the communication capability: one can give the full description of the language interpreter in one and the same formalism. Thus, properties that are not just related to communications, but rather involve communication in combination to other capabilities can be more easily proved.

We have experimented the proposed approach for defining the *full* operational semantics of the interpreter of the DALI language [39]. DALI [11] [12] is a Horn-clause agent oriented language, with a declarative and procedural semantics that have been obtained as an extension of the standard one. The operational semantics that we have defined has allowed us to prove some relevant properties. In particular, we have proved correctness of the interpreter w.r.t. DALI extended resolution, and compliance w.r.t. some communication protocols.

Motivated by this successful experiment, we here propose the adoption of dialogue games (with the associated transition systems) as a *general methodology* for describing the interpreters of AOL's, where an interpreter is seen as divided into "players" corresponding to the agent capabilities it implements.

Below we propose how to define such a dialogue game. Notice that many aspects of our formalization are meant to be pretty general so that they can be "customized" to the various approaches and languages to which the methodology can be applied.

Below we define in a very basic way the *state* of an agent.

Definition 1 (State of an agent). *Let Ag_x be the name of the agent. We define the internal state IS_{Ag_x} as the tuple $< E, A, G >$ composed by the current sets of events perceived, actions to be performed and goals to be achieved.*

Then, we define what is a law for the particular kind of dialogue game that we are introducing.

Definition 2 (Law). *We define a law L_x as a tuple composed of the following elements:*

- ***Name:*** *the name of law.*
- ***Locution:*** *The utterance, that expresses the purpose of the application of the law.*
- ***Preconditions:*** *The preconditions for the application the law.*
- ***Meaning:*** *The informal meaning of the law. This element is aimed at documentation.*
- ***Response (Post-conditions):*** *The effects of the application of the law, i.e., its post-conditions, that may involve expected utterances from other parties.*

Consider that, being the game innocent, the other players (and, ultimately, the overall interpreter) are expected to behave as specified by a law whenever it is applied. In particular, a law is applied with respect to the current state of the interpreter, which is defined as follows.

Definition 3. *A state of the interpreter is a pair $< Ag_x, S_{Ag_x} >$ where Ag_x is the name of the agent and the* operational state *S_{Ag_x} is a triple $< P_{Ag_x}, IS_{Ag_x}, Mode_{Ag_x} >$. The first element is the logic program defining the agent, the second one is the agent current internal state, the third one is a particular attribute, that we call* modality, *which describes what the interpreter is doing.*

The application of laws is regulated by transition rules that specify how the state of the interpreter changes when the laws are applied.

Definition 4 (Transition rule). *A transition rule has the following form:*
$< Ag_x, < P, IS, Mode >> \overset{L_i,\ldots,L_j}{\longrightarrow} < Ag_x, < NewP, NewIS, NewMode >>$
where $L_i, ..., L_j$ are the laws which are applied in the given state of the interpreter thus obtaining a new state where some elements have possibly changed. Namely, $NewP$, $NewIS$ and $NewMode$ indicate, respectively, P, IS and $Mode$ updated after applying the laws.

A transition rule can also describe how an agent can influence an other one. In this case, we will have:

Definition 5 (Inter-Agent Transition rule). *A transition rule involving two agents has the following form:*
$< Ag_x, < P_{Ag_x}, IS_{Ag_x}, Mode_{Ag_x} >> \overset{L_i,\ldots,L_j}{\longrightarrow} < Ag_y, < P_{Ag_y}, IS_{Ag_y}, Mode_{Ag_y} >>$
$where\, x \neq y$

About the general features of the game, we may notice that:

- **Commencement Rules** here define the activation of an agent, and imply as preconditions the acquisition of a syntactically correct logic program and of the possible initialization parameters. The response consists in the creation of the initial state and in the activation of the basic interpreter cycle.

- **Combination Rules** define in which order the transition rules should be applied if more than one is applicable (i.e., the preconditions of the corresponding laws are verified). In our case, the only rule is that the component corresponding to the basic interpreter cycle must regain the control after a predefined quantum of time. The basic cycle will then utter locutions that activate the other components according to its policy.
- **Commitments** are taken for granted, i.e., the players (components of the interpreter) always accept to give the needed responses and thus make the expected effects of transitions actual.
- **Termination Rules** should define under which circumstances an agent stops its activity. They may be missing if the agent is supposed to stay alive forever (or, in practice, as long as possible).

4 Examples of Laws and Transition Rules

In this Section we show some laws and rules that are taken from the operational semantics of DALI, but that are general enough to be easily adapted to other languages and formalisms. It is may be interesting to notice that the full DALI interpreter is described by 90 laws and 141 transition rules (fully reported in [39]).

4.1 Message Reception Player

We now present one of the players that belong to the DALI interpreter. Its function is that of receiving the incoming messages, and thus we may say that it is fairly general rather than specific of DALI. The only specific feature is that a DALI logic program includes a set of meta-rules that define the distinguished predicate *told* in order to specify constraints on the incoming messages. A message consists in general (at least) of: a sender, a primitive, a content, the language in which it is expressed and the adopted communication protocol. In DALI, if the message elements satisfy the constraints specified in the *told* rules, then the message is accepted; otherwise, it is discarded. The component of the DALI interpreter which implements message reception including the *told* filter on incoming messages is called TOLD player. Often, an agent may need some reasoning (e.g., ontological reasoning) in order to understand message contents: the DALI communication architecture [13] includes to this aim the distinguished predicate *meta* which has a predefined part, and can be also be customized by the programmer. The processing of this stage by the DALI interpreter corresponds to the META player.

Below are the laws and rules for the TOLD player. Laws and rules are reported with the numbers occurring in [39]. The first law connects the agent to the input message space. Each agent checks, from time to time, if in the space of incoming message (that, in the case of DALI, is the Linda tuple space) there a message for it. In this case, it takes the message and starts to inspect it.

The first law describes this process and extracts from the whole message the parameters which are relevant for TOLD rules: the sender agent and the content of the message.

L12: The **L12 receive_message(.)** law:
Locution: $receive\_message(Ag_x, Ag_y, Protocol, Ontology, Language, Primitive)$
Preconditions: this law can be applied when agent Ag_x finds a new message in input message space.
Meaning: the agent Ag_x receives a message from Ag_y (environment, other agents,...). For the sake of simplicity, we consider the environment as an agent.
Response: The player considers the information about language and ontology and extracts the name of sender agent and the primitive contained in the message.

Law L_{13} verifies protocol compatibility. If the sender agent protocol is different from that of the receiver agent, then the message is discarded "a priori".

L13: The **L13 receive_message(.)** law:
Locution: $receive\_message(Ag_x, Ag_y, Protocol, Ontology, Language, Primitive)$
Preconditions: The Protocol is compatible with the one of the receiver agent.
Meaning: This law discards all messages that the agent could not understand correctly considered the different protocols adopted. If the protocols coincide, then the message goes through.
Response: Accepted messages enters into the TOLD level.

L_{14} verifies whether the *told* constraints for the received message are all satisfied and, if so, it allows the message to move forward.

L14: The **L14 TOLD_check_true(.)** law:
Locution: $TOLD\_check\_true(Ag_y, Protocol, Primitive)$
Preconditions: The constraints of TOLD rules applicable to the name of the sender agent Ag_y and to the primitive and content must be true.
Meaning: The communication primitive is submitted to the check represented by TOLD rules.
Response: The primitive can be processed by the next step.

The TOLD player plays the game not only with itself and with other internal players but also with the environment. With itself, because it has an "its own opinion" (derived by means of the *told* rules) about the sender, primitive and content of the message (is the sender reliable? is the content compatible with its role and its knowledge base?) With the others, because it interplays moves with the META and interpreter (basic cycle) players, where the META player will apply ontologies for "understanding" the message contents. Finally, with the environment because the message contains information influencing its reception or elimination. For example, a non correct syntax of the message can determine its elimination independently of the "opinion" of the component.

The transition rules below specify how the laws affect the state of the interpreter. Of particular importance here is the third element of the configuration, i.e., the modality, which specifies the stage where the player (which is a component of the overall interpreter) is currently situated. For instance, the transition rule below specifies that if the TOLD player is in modality *manage_perceptions* and finds an incoming message, it will go into the $received\_message$ modality. The transition may occur if law L_{12} can be applied. This law incorporates the precondition that an agent can receive a message only if a physical communication act has taken place in the server. We can see that the state of the interpreter does not change. This because law L_{12} extracts the parameters from the message but does not involve internal processes of the agent.

$R45:$

$< Ag_1, < P, IS, manage\_incoming\_messages_{Ag_1} >> \xrightarrow{L_{12}}$
$< Ag_1, < P, IS, received\_message(Message)_{Ag_1} >>$

The R46 transition rule verifies the message protocol compatibility and only in the positive case allows the communicative act to go through. This basic filter is relevant because protocol incompatibility can generate serious damages in the receiver agent. In fact, also if the primitive names are the same, the arguments can be different and this may result in a misleading interpretation.

$R46$

$< Ag_1, < P, IS, received\_message(Message)_{Ag_1} >> \xrightarrow{L_{13}}$
$< Ag_1, < P, IS, protocol\_compatible(Protocol, Message)_{Ag_1} >>$

Rule R47 forces the agent to eliminate a message with an incompatible protocol. The specification $not(L_{13})$ means that this transition takes place only in case law L_{13} cannot be applied.

$R47$

$< Ag_1, < P, IS, received\_message(Message)_{Ag_1} >> \xrightarrow{not(L_{13})}$
$< Ag_1, < P, IS, end\_manage\_incoming\_messages_{Ag_1} >>$

The R48 transition rule specifies that, whenever parameters $Sender$ and $Content$ have been extracted and the protocol control has been successfully performed, the player must invoke the corresponding TOLD rules. L_{14} is in fact applicable in the situation where all constraints are verified and the message can be accepted by the agent. This step does not involve the logic program or the events queues, thus only the modality is affected.

$R48:$

$< Ag_1, < P, IS, protocol\_compatible(Message)_{Ag_1} >> \xrightarrow{L_{14}}$
$< Ag_1, < P, IS, TOLD(Sender, Content)_{Ag_1} >>$

If instead at least one constraint is not fulfilled, the precondition of L_{14} law becomes false and the above transition cannot be applied. This means that the message contains some item of information which is considered by the receiver agent to be either uninteresting or harmful.

The move of the TOLD player that influences the behavior of the META player is represented by the modality $TOLD(Sender, Content)$. In the DALI actual architecture, this move corresponds to the message overcoming of the TOLD filter level. Only in this case the message reaches the META component where the agent will try to "understand" the contents by possibly resorting to an ontology.

4.2 Communication Protocols

In [39] we have defined by means of suitable laws and transition rules two communication protocols: the first one is the FIPA/DALI protocol. The second one, called the

"Four-acts protocol", is inspired by the work Sadri, Toni and Torroni reported in [36]. We have described this protocol and we discuss its description here because its simplicity allows us to show all the related laws and rules. We notice however that the proposed semantics framework is in principle able to accommodate any other protocol.

In the Four-act protocol we have two agents, say Ag_1 and Ag_2, each of which defined by a set of rules that determine its role in the dialogue. The dialogue subject is the request of a resource R by Ag_1 to Ag_2. Agent Ag_1 has the following rules, stating that as soon as it decides (by performing the action $init(Ag_1)$ to initiate a conversation with agent Ag_2 (which is not explicitly named, as Ag_1 and Ag_2 are the only agents which are supposed to exist in this setting) it starts asking for a certain $Resource$. Either Ag_2 accepts to give it, and then Ag_1 goes into an *ok* state, or Ag_2 refuses and then Ag_1 asks again.

Ag_1 protocol rules
$ask(give(Resource), Ag_1) \Leftarrow init(Ag_1)$ (p_1)
$ok(give(Resource)Ag_1) \Leftarrow accept(give(Resource), Ag_2)$ (p_2)
$ask(give(Resource), Ag_1) \Leftarrow refuse(give(Resource), Ag_2)$ (p_3)

Ag_2 is the agent that receives the request and must decide whether to give the required resource. The choice depends upon its internal condition C, according to the following rules:

Ag_2 protocol rules
$accept(give(Resource), Ag_2) \Leftarrow ask(give(Resource), Ag_1) \wedge C$ (p_4)
$refuse(give(Resource), Ag_2) \Leftarrow ask(give(Resource), Ag_1) \wedge \neg C$ (p_5)

The requesting process goes on indefinitely if Ag_1 does not obtain the resource: therefore, the above definition makes sense as an example, as in "real world" the agents would presumably somehow put an end to the interaction. We now present the laws and transition rules that operationally formalize this protocol. These laws and rules rely upon general laws and rules managing message exchange (reported [39]), and this is why in transition rules below one may notice the occurrence of laws not shown here.

The first law makes the interpreter able to adopt this protocol (among the available ones), while the subsequent laws introduce the corresponding communication acts.

L1: adopt_four-acts_protocol(.) law:
Locution: $adopt\_four\_act\_protocol()$
Preconditions: No precondition.
Meaning: This law allows an agent to adopt Four-act protocol.
Response: Subsequent communication acts will be processed according to the Four-acts protocol.

The following transition rule leads the interpreter of any agent Ag to understand the communication primitives of forthcoming messages according to the *Four-acts protocol*:

$$\begin{array}{l} R1: \\ < Ag, < P, IS, choose\_protocol >> \xrightarrow{L_1} \\ < Ag, < P, IS, process\_communication(Four-act) >> \end{array}$$

The following laws and rules formalize both roles, the one of the former agent Ag_1, here called the $Sender$, and the one of the latter agent Ag_2. Law $L2$ processes the initiative that the former agent, here called $Sender$, to start the dialogue with the latter agent, here called "opponent", which is supposed to be active in the environment. $L3$ copes with the dispatch of a request. The subsequent laws pertain to the opponent agent. $L4$ copes with reception of a request, $L7$ checks whether the agent is able to give the resource or not, while $L5$ and $L6$ are respectively responsible for acceptance and refusal. Finally, in the $Sender$ agent $L8$ terminates the dialogue in case it has obtained the resource.

L2: process(Four-acts, init(.)) law:
Locution: $process(Four - Acts, init(Sender))$
Preconditions: No precondition.
Meaning: This law sets the interpreter of the *Sender* agent to a state where the agent has initiated a dialogue.
Response: A dialogue is started.

L3: process(Four-acts, out_ask(.)) law:
Locution: $process(Four - Acts, send(ask(Resource, Sender)))$
Preconditions: The dialogue has been started through the *init* primitive or a refusal has been received.
Meaning: This law sets the interpreter to a state where a request has been sent.
Response: The request is delivered to the opponent agent.

L4: process(Four-acts, in_ask(.)) law:
Locution: $process(Four - acts, receive(ask(Resource, Sender)))$
Preconditions: The dialogue has been started through the *init* primitive and a request has been received.
Meaning: This law sets the interpreter of the opponent agent to a state that allows the agent to evaluate a request.
Response: The opponent agent evaluates the incoming request.

L5: process(Four-acts, out_accept(.)) law:
Locution: $process(Four - acts, send(accept(Resource, Sender)))$
Preconditions: An incoming request has been positively evaluated.
Meaning: The answer to a request by the *Sender* agent is acceptance.
Response: The requester agent is enabled to obtain the *Resource*.

L6: process(Four-acts, out_refuse(.)) law:
Locution: $process(Four - acts, refuse(Resource, Sender))$
Preconditions: An incoming request has been negatively evaluated.
Meaning: The answer to a request by the *Sender* agent is refusal.
Response: The requester agent is not enabled to obtain the *Resource*.

L7: verify_request(.) law:
Locution: $verify\_request(Resource, Sender)$
Preconditions: A request has been received.
Meaning: The opponent agent evaluates an incoming request by checking condition C.
Response: If condition C is fulfilled, the answer will be positive.

L8: process(Four-acts, in_accept(.)) law:
Locution: $process(Four-acts, receive(accept(Resource, Sender)))$
Preconditions: A request has been issued.
Meaning: The answer to a former request is acceptance.
Response: The requester agent is enabled to reach an ok state.

L9: process(Four-acts, in_refuse(.)) law:
Locution: $process(Four-acts, receive(refuse(Resource, Sender)))$
Preconditions: A request has been issued.
Meaning: The answer to a former request is refusal.
Response: The requester agent will try again.

L10: process(Four-acts, ok(.)) law:
Locution: $process(Four-acts, ok(Resource, Sender))$
Preconditions: The agent has obtained the resource.
Meaning: This law allows an agent to conclude the dialogue having obtained the desired resource.
Response: The dialogue terminates.

The following state transition rules modify the states of the interpreters of the participating agents, according to the application of the above laws. As we have not discussed the general laws and rules for message exchange, we cannot go in depth into the change of state. We just indicate that a state IS is transformed into a new state NIS. The change in the modality instead is clearly visible, showing a shift from what the interpreter was about to do before to what the interpreter is expected to do next.

$$R2 :$$
$$< Ag, < P, IS, process\_communication(Four-acts) >> \xrightarrow{L_2, L_{23}}$$
$$< Ag, < P, NIS, sent(Four-acts, ask(Resource, Sender)) >>$$

The agent receiving a resource request, accepts the proposal if law L_7 permits this:

$$R3 :$$
$$< Ag, < P, IS, received\_request(Four-acts, ask(Resource, Sender) >> \xrightarrow{L_4, L_7, L_5, L_{23}}$$
$$< Ag, < P, NIS, sent(Four-acts, accept(Resource, Sender)) >>$$

If the internal state does not authorize the resource transfer, the agent sends a *refuse* message. Notice that the transition is determined by the inapplicability of a law (because condition C is *not* fulfilled).

$$R4 :$$
$$< Ag, < P, IS, received\_request(Four-acts, ask(Resource, Sender) >> \xrightarrow{L_4, not(L_7), L_6, L_{23}}$$
$$< Ag, < P, NIS, sent(Four-acts, refuse(Resource, Sender)) >>$$

A positive response to the assignment problem concludes the process and the interpreter goes into the *ok* modality.

$$R5 :$$
$$< Ag, < P, IS, sent(Four-acts, ask(Resource, Sender) >> \xrightarrow{L_8, L_10, L_{23}}$$
$$< Ag, < P, NIS, have\_resource(Four-acts, ok(Resource, Sender)) >>$$

An agent that receives a refusal, persists stubbornly in its intent by returning to the *ask* modality:

$R6:$
$$< Ag, < P, IS, sent(Four - acts, ask(Resource, Sender) >> \overset{L_9,L_4,L_{23}}{\longrightarrow}$$
$$< Ag, < P, NIS, sent(Four - acts, ask(Resource, Sender)) >>$$

5 Abstracting Agent Properties

We now propose some abstractions over the operational semantics behavior, that can be useful in order to define and prove properties. As a basic step, we define a function describing the agent behavior as the composition of the application of a given set of transition rules, starting from a specific agent operational state S_{Ag_x}. This abstraction process consists in formalizing the *Operational Semantics Call*:

Definition 6 (Operational Semantic Call). *Given a list of transition rules* $(R_1, ..., R_n)$ *and an initial operational state* $S0_{Ag_x}$ *of an agent* Ag_x*, we define the Operational Semantics Call* $\Psi_{[k,...,n]}(Ag_x, S0_{Ag_x}, (R_1, ..., R_n))$ *as follows:*

$\Psi_{[R_1,...,R_n]}(Ag_x, S0_{Ag_x}, (R_1, ..., R_n)) = \langle Ag_x, Sf_{Ag_x} \rangle$
such that
$\langle Ag_x, S0_{Ag_x} \rangle \xrightarrow{R_1} \langle Ag_x, S1_{Ag_x} \rangle \xrightarrow{R_2} ... \xrightarrow{R_n} \langle Ag_x, Sf_{Ag_x} \rangle$ *where* Sf_{Ag_x} *is the final operational state determined by the subsequent application of the given transition rules.*

We then introduce a semantic function *Behavior*, that allows us to explore agent coherence by relating the perceptions that an agent receives in input to the actions that the agent performs, which constitute its observable output behavior. We consider the agent as a "black box" whose evolution is completely determined by the *Operational Semantics Call* Ψ computed on a given a list of transition rules and a given set of perceptions. Notice that perceptions are partly recorded in the "events" component of the initial state, and partly related to the laws which are applied by the given transition rules: as we have seen, there are laws that are applicable if there is an incoming perception, for instance an incoming message. Perceptions can take different forms and can be described in different ways depending upon the specific agent-oriented language/formalism and the application context at hand. Thus, we will not commit to any specific description. We only notice that in general: (i) A particular kind of perception consists in the reception of a message. (ii) A perception will be annotated with a time-stamp that indicates when the agent has become aware of it; then, perceptions are assumed to be totally ordered w.r.t. the time-stamp and can be compared on this basis.

The function *Behavior* operates an abstraction process that considers only the perceptions in input and the actions in output. As a preliminary step, we define the *Action Semantic function* that returns all operational states Sf_{Ag_x} whose modality indicates that an action has been performed. A particular kind of action consists in sending a message.

Definition 7 (Action Semantic function). *Given the Operational Semantics Call* Ψ *computed on a list* $(R_1, ..., R_n)$ *of transition rules, we define the Action Semantic function* Ξ_{Ag_x} *as a function that, given* $\Psi_{[R_1,...,R_n]}$*, returns all* Sf_{Ag_x}*'s related to performing an action:*

$\Xi_{Ag_x}(p, \Psi_{[R_1,\dots,R_n]}) =$
$\{S_{f_{Ag_x}} |$
$\quad modality(S_{f_{Ag_x}}) = made(Action) \vee$
$\quad modality(S_{f_{Ag_x}}) = sent(Message/Action)\}$
where function "modality" extracts the modality from $S_{f_{Ag_x}}$.

We now consider the set of actions that an agent performs *in response to a certain perception* p present in the initial state. It is obtained as the set of actions contained in the states specified by the above *Action Semantic Function* when the agent receives the perception p as input.

Definition 8 (Semantic Behavior with respect to the perception p**).** *Let* $\Xi_{Ag_x}(p, \Psi_L)$ *be the Action Semantic Function applied to a list L of transition rules and let the agent initial state contain only perception p. We define the Semantic Behavior* $Beh^p_{Ag_x}(p, L)$ *with respect the perception p as the set of actions belonging to the* $S_{f_{Ag_x}}$*'s selected by* Ξ*:*
$Beh^p_{Ag_x}(p, L) =$
$\quad \{A | \; A \; is \; an \; action \; where \; \exists \; S_{f_{Ag_x}} \in \Xi_{Ag_x}(p, \Psi_L) \; such \; that \; A \in S_{f_{Ag_x}}\}$

Below we extend the definition to consider a set P of perceptions.

Definition 9 (Semantic Behavior). *Let* $Beh^p_{Ag_x}(X, L)$ *be the Semantic Behavior of the agent with respect to the single perception, let L be a list of transition rules, let* $P = \{p_1, ..., p_n\}$ *be the set of perceptions contained in the agent initial state. We define the Semantic Behavior* $Beh_{Ag_x}(P, L)$ *with respect to P as*

$Beh_{Ag_x}(P, L) = \bigcup_{k=1,\dots,n} Beh^p_{Ag_x}(p_k, L)$

The definition of the agent behavior through the operational semantics allows us to consider properties of our agents. An important property is conformance of an agent actual behavior with respect to the expected behavior (however defined, we do not enter into details here). We can say that an agent is *coherent* if it performs in response to a set of perceptions the set of actions that the designer expects from it.

Definition 10 (Coherent agent). *Let* $Beh_{Ag_x}(P, L)$ *be the behavior (set of actions) of the agent* Ag_x *with respect to a set of perceptions P and a list L of transition rules. The agent* Ag_x *is coherent with respect to the expected behavior* β_{Ag_x} *if* $\beta_{Ag_x} \subseteq Beh_{Ag_x}(P, L)$.

Definition 11 (Strongly Coherent agent). *A coherent agent* Ag_x *is strongly coherent with respect to the expected behavior* β_{Ag_x} *if* $\beta_{Ag_x} = Beh_{Ag_x}(P, L)$.

Let us restrict the set of perception that we consider to the incoming messages, and the expected behavior β_{Ag_x} to the out-coming message. Then, these two elements together may be assumes to describe the communication protocol Prt_{Ag_x} that the agent is expected to follow by an external observer. Than, strong coherence coincides with compliance w.r.t. the protocol.

Definition 12 (Protocol Compliance). *Consider a protocol* $Prt_{Ag_x} = \langle P, \beta_{Ag_x} \rangle$ *where let* P *be a the set of incoming messages and* β_{Ag_x} *the corresponding expected set of out-coming messages.* Ag_x *is compliant w.r.t.* Prt_{Ag_x} *if it is strongly coherent w.r.t.* P *and* β_{Ag_x}*.*

As a simple example of protocol compliance, it is easy to see that, if we let OP be the set of laws and rules for the Four-act protocol as defined in the previous section, we have:

Theorem 1. *Agent* Ag_x *whose semantic behavior* Beh_{Ag_x} *is defined according to the transition rules in* OP *is compliant w.r.t. the Four-act protocol.*

Intuitively, this is the case as laws L2-L10 correspond to the nine actions which are present in the protocol definition, and each transition rule corresponds to a rule of the protocol itself (plus the commencement rules R1 and R2).

6 Conclusions

We have proposed a new approach to the operational description of the interpreter (or, more generally, of a computational device) underlying a logic agent-oriented language. This description can be used to prove properties of the interpreter, e.g., correctness w.r.t. the language semantics or compliance to a communication protocol, and, indirectly, also properties of programs or classes of programs.

The formalism chosen for providing this description is that of dialogue games, composed of laws with an associated state transition system. The "added value" of adopting a dialogue game is manifold: laws have pre- and post-conditions, therefore conditioning state transitions; it is possible to specify how the state changes if a rule is applicable, but also when it is *not* applicable; the various components of the interpreter can be formalized in a modular and elaboration-tolerant way as different "players" that play an innocent game in order to contribute to the overall interpreter behavior; the interaction with other agents is coped with in a uniform way; game theory can be exploited for proving properties.

We have experimented the approach in the formalization of the DALI languages. We mean to consider other languages/formalisms and other less simple formal properties of single agents or of multi-agent systems.

References

1. Alechina, N., Dastani, M., Logan, B., Meyer, J.-J.C.: A Logic of Agent Programs. In: Proc. of the Twenty-Second National Conference on Artificial Intelligence (AAAI 2007), pp. 795–800. AAAI Press, Menlo Park (2007)
2. Apt, K.R., Bol, R.: Logic programming and negation: a survey. J. of Logic Programming 19/20 (1994)
3. Arisha, K.A., Ozcan, F., Ross, R., Subrahmanian, V.S., Eiter, T., Kraus, S.: IMPACT: a platform for collaborating agents. IEEE Intell. Systems 14(2) (1999)
4. Baldoni, M., Baroglio, C., Martelli, M., Patti, V.: Reasoning about logic-based agent interaction protocols. In: Proc. of the Italian Conf. on Comp. Logic, CILC 2004 (2004)

5. Baldoni, M., Baroglio, C., Martelli, M., Patti, V., Schifanella, C.: Verifying protocol conformance for logic-based communicating agents. In: Leite, J., Torroni, P. (eds.) CLIMA 2004. LNCS (LNAI), vol. 3487, pp. 196–212. Springer, Heidelberg (2005)
6. Boerger, E., Rosenzweig, D.: A mathematical definition of full Prolog. Science of Computer Programming, vol. 24. Elsevier, Amsterdam (1995)
7. Bordini, R.H., Fisher, M., Pardavila, C., Wooldridge, M.: Model checking AgentSpeak. In: Proc. of Second Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems. ACM Press, New York (2003)
8. Bordini, R.H., Moreira, A.F.: Proving BDI properties of Agent-Oriented Programming Languages. Ann. Math. Artif. Intell. 42(1-3) (2004)
9. Bracciali, A., Mancarella, P., Stathis, K., Toni, F.: On modelling declaratively multi-agent systems. In: Leite, J., Omicini, A., Torroni, P., Yolum, p. (eds.) DALT 2004. LNCS (LNAI), vol. 3476, pp. 53–68. Springer, Heidelberg (2005)
10. Bracciali, A., Demetriou, N., Endriss, U., Kakas, A., Lu, W., Mancarella, P., Sadri, F., Stathis, K., Terreni, G., Toni, F.: The KGP model of agency: Computational model and prototype implementation. In: Priami, C., Quaglia, P. (eds.) GC 2004. LNCS (LNAI), vol. 3267, pp. 340–367. Springer, Heidelberg (2005)
11. Costantini, S., Tocchio, A.: A Logic Programming Language for Multi-agent Systems. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, p. 1. Springer, Heidelberg (2002)
12. Costantini, S., Tocchio, A.: About declarative semantics of logic-based agent languages. In: Baldoni, M., Endriss, U., Omicini, A., Torroni, P. (eds.) DALT 2005. LNCS (LNAI), vol. 3904, pp. 106–123. Springer, Heidelberg (2006)
13. Costantini, S., Tocchio, A., Verticchio, A.: Communication and Trust in the DALI Logic Programming Agent-Oriented Language. Intelligenza Artificiale, J. of the Italian Association, Year 2 n. 1 (2005)
14. Debray, S.K., Mishra, P.: Denotational and Operational Semantics for Prolog. J. of Logic Programming 5, 61–91 (1988)
15. d' Inverno, M., Kinny, D., Luck, M., Wooldridge, M.: Formal Specification of dMARS. In: Rao, A., Singh, M.P., Wooldridge, M.J. (eds.) ATAL 1997. LNCS, vol. 1365, pp. 155–176. Springer, Heidelberg (1998)
16. d' Inverno, M., Luck, M.: Formalising the Contract Net as a Goal-Directed System. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038. Springer, Heidelberg (1996)
17. d' Inverno, M., Luck, M.: Engineering AgentSpeak(L): A Formal Computational Model. J. of Logic and Computation 8(3) (1998)
18. Georgeff, M.P., Lansky, A.L.: Reactive Reasoning and Planning. In: Readings in Planning. Morgan-Kaufmann, San Francisco (1990)
19. Goodwin, R.: A formal specification of agent properties. J. of Logic and Computation 5(6) (1995)
20. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.-J.C.: Formal semantics for an abstract agent programming language. In: Rao, A., Singh, M.P., Wooldridge, M.J. (eds.) ATAL 1997. LNCS, vol. 1365, pp. 215–229. Springer, Heidelberg (1998)
21. Hindriks, K.V., de Boer, F.S., van der Hoek, W., Meyer, J.-J.C.: Agent programming in 3APL. Autonomous Agents and Multi-Agent Systems 2(4) (1999)
22. Kowalski, R.A.: How to be Artificially Intelligent - the Logical Way, Draft (2010), http://www-lp.doc.ic.ac.uk/UserPages/staff/rak/rak.html
23. Larson, K., Sandholm, T.: Bargaining with limited computation: Deliberation equilibrium. Artificial Intelligence 132(2) (2000)

24. Leite, J.A., Alferes, J.J., Pereira, L.M.: $\mathcal{MINERVA}$ - A dynamic logic programming agent architecture. In: Meyer, J.-J.C., Tambe, M. (eds.) ATAL 2001. LNCS (LNAI), vol. 2333, p. 141. Springer, Heidelberg (2002)
25. Lloyd, J.W.: Foundations of Logic Programming, 2nd edn. Springer, Berlin (1987)
26. Luck, M., Griffiths, N., d' Inverno, M.: From agent theory to agent construction: A case study. In: Jennings, N.R., Wooldridge, M.J., Müller, J.P. (eds.) ECAI-WS 1996 and ATAL 1996. LNCS, vol. 1193, pp. 49–64. Springer, Heidelberg (1997)
27. Parkes, D.C.: Optimal auction design for agents with hard valuation problems. In: Proc. IJCAI 1999 Workshop on Agent Mediated Electronic Commerce (AmEC 1999), Stockholm (1999)
28. McBurney, P., Parsons, S.: Dialogue Games in Multi-Agent Systems. Informal Logic, Special Issue on Applications of Argumentation in Computer Science 22(3) (2002)
29. McBurney, P., Parsons, S.: Dialogue games for agent argumentation. In: Argumentation in Artificial Intell., ch. 13, pp. 261–280. Springer, Berlin (2009)
30. McBurney, P., Parsons, S.: Games that agents play: A formal framework for dialogues between autonomous agents. J. of Logic, Language and Information, Special Issue on Logic and Games 11(3) (2002)
31. McBurney, P., van Eijk, R.M., Parsons, S., Amgoud, L.: A Dialogue Game protocol for agent purchase negotiation. Autonomuos Agents and Multi-Agent Systems 7(3) (2003)
32. Parsons, S., Wooldridge, M., Amgoud, L.: Properties and complexity of some formal inter-agent dialogues. J. of Logic and Computation 13(3), 347–376 (2003)
33. Plotkin, G.: A structural approach to operational semantics. Technical report DAIMI FN-19, Aarhus University, Computer Science Department (1981)
34. 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 (LNAI), vol. 1038. Springer, Heidelberg (1996)
35. Rao, A.S., Georgeff, M.P.: An Abstract Architecture for Rational Agents. In: Proc. of the 13th International Joint Conference on Artifical Intelligence, vol. 1, pp. 318–324. Morgan Kaufmann Publishers Inc., San Francisco (1993)
36. Sadri, F., Toni, F., Torroni, P.: Dialogues for negotiation: agent varieties and dialogue sequences. In: Meyer, J.-J.C., Tambe, M. (eds.) ATAL 2001. LNCS (LNAI), vol. 2333, p. 405. Springer, Heidelberg (2002)
37. Sandholm, T.: Unenforced e-commerce Transactions IEEE Internet Computing 1(6) (1997)
38. Sandholm, T., Suri, S., Gilpin, A., Levine, D.: CABOB: A fast optimal algorithm for combinatorial auctions. In: Proc. of IJCAI 2001, Seattle, WA (2001)
39. Tocchio, A.: Multi-agent systems in computational logic, Ph.D. Thesis, Dipartimento di Informatica, Universitá degli Studi di L'Aquila (2005)
40. Viroli, M., Omicini, A.: Multi-Agent Systems as Composition of Observable Systems. In: WOA 2001: Dagli oggetti agli agenti: tendenze evolutive dei sistemi software (2001), http://lia.deis.unibo.it/books/woa2001/pdf/22.pdf
41. Wooldridge, M., Jennings, N.R.: Formalizing the cooperative problem solving process. In: Readings in Agents, pp. 430–440. Morgan Kaufmann Publishers Inc., San Francisco (1997)
42. Wooldridge, M., Lomuscio, A.: A Logic of Visibility, Perception, and Knowledge: Completeness and Correspondence Results. Logic Journal of the IGPL 9(2) (2001)

Author Index

Ågotnes, Thomas 9
Apt, Krzysztof R. 186

Benzmüller, Christoph 33
Bianco, Alessandro 171
Boissier, Olivier 117
Bordini, Rafael H. 117
Broda, Krysia 49

Calta, Jan 135
Chesani, Federico 82
Costantini, Stefania 238

D'Aprile, Davide 99

Faella, Marco 171

Ghosh, Sujata 153
Giordano, Laura 99
Gliozzi, Valentina 99
Goebel, Randy 49
Góngora, Pedro Arturo 65

Hernández-Quiroz, Francisco 65
Hindriks, Koen V. 219
Hosobe, Hiroshi 49
Hübner, Jomi F. 117

Jongmans, Sung-Shik T.Q. 219

Kafalı, Özgür 82
Kowalski, Robert 200

Ma, Jiefei 49
Martelli, Alberto 99
Miner More, Sara 15
Mogavero, Fabio 171
Murano, Aniello 171

Naumov, Pavel 15

Pozzato, Gian Luca 99

Ramanujam, R. 153
Russo, Alessandra 49

Sadri, Fariba 200
Satoh, Ken 49
Schlingloff, Holger 135
Shkatov, Dmitry 135
Simon, Sunil 153

Theseider Dupré, Daniele 99
Tocchio, Arianna 238
Torroni, Paolo 82

Ufferman, Eric 65

van der Meyden, Ron 8
van Riemsdijk, M. Birna 219

Walsh, Toby 1
Woltran, Stefan 14

Zvesper, Jonathan A. 186