Model Checking Logics of Social Commitments for Agent Communication

Mohamed El Menshawy Mohamed

A Thesis
In the Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Electrical and Computer Engineering) at
Concordia University
Montréal, Québec, Canada

September 2012

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Mr. Mohamed El Menshawy Mohamed

Entitled: Model Checking Logics of Social Commitments for Agent Communication

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Electrical and Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________ Chair
Dr. Lyes Kadem

______________________ External Examiner
Dr. Hanifa Boucheneb

______________________ External to Program
Dr. Olga Ormandjieva

______________________ Examiner
Dr. Benjamin C. M. Fung

______________________ Examiner
Dr. Ferhat Khendek

______________________ Thesis Co-Supervisor
Dr. Jamal Bentahar

______________________ Thesis Co-Supervisor
Dr. Rachida Dssouli

Approved by ______________________
Chair of Department or Graduate Program Director

____
Dr. Robin A. L. Drew, Dean
Faculty of Engineering and Computer Science

Date __________ 2012

Abstract

Model Checking Logics of Social Commitments for Agent Communication

Mohamed El Menshawy Mohamed, Ph.D.

Concordia University, 2012

This thesis is about specifying and verifying communications among autonomous and possibly heterogeneous agents, which are the key principle for constructing effective open multi-agent systems (MASs). Effective systems are those that successfully achieve applicability, feasibility, error-freeness and balance between expressiveness and verification efficiency aspects. Over the last two decades, the MAS community has advocated social commitments, which successfully provide a powerful representation for modeling communications in the figure of business contracts from one agent to another. While modeling communications using commitments provides a fundamental basis for capturing flexible communications and helps address the challenge of ensuring compliance with specifications, the designers and business process modelers of the system as a whole cannot guarantee that an agent complies with its commitments as supposed to or at least not wantonly violate or cancel them. They may still wish to first formulate the notion of commitment-based protocols that regulate communications among agents and then establish formal verification (e.g., model checking) by which compliance verification in those protocols is possible.

In this thesis, we address the aforementioned challenges by firstly developing a new branching-time temporal logic—called ACTL^{*c}—that extends CTL* with modal operators for representing and reasoning about commitments and all associated actions. The proposed semantics for ACL (agent communication language) messages in terms of commitments and their actions is formal, declarative, meaningful, verifiable and semi-computationally grounded. We use ACTL^{*c} to derive a new specification language of commitment-based protocols, which is expressive and suitable for model checking. We introduce a reduction method to formally transform the problem of model checking ACTL^{*c} to the problem of model checking GCTL* so that the use of the CWB-NC model checker is possible. We prove the soundness of our reduction method and implement it on top of CWB-NC. To check the effectiveness of our reduction method, we report the verification results of the NetBill protocol and Contract Net protocol against some properties. In addition to the reduction method, we develop a new symbolic algorithm to perform model checking ACTL^{*c}.

To balance between expressiveness and verification efficiency, we secondly adopt a refined fragment of ACTL^{*c}, called CTLC, an extension of CTL with modalities for commitments and their fulfillment. We extend the formalism of interpreted systems introduced to develop MASs with shared and unshared variables and considered agents' local states in the definition of a full-computationally grounded semantics for ACL messages using commitments. We present reasonable axioms of commitment and fulfillment modalities. In our verification technique, the problem of model checking CTLC is reduced into the problems of model checking ARCTL and GCTL* so that respectively extended NuSMV and CWB-NC (as a benchmark) are usable. We prove the soundness of our reduction methods and then implement them on top of the extended NuSMV and CWB-NC model checkers. To evaluate the effectiveness of our reduction methods, we verified the correctness of two business case studies.

We finally proceed to develop a new symbolic model checking algorithm to directly verify commitments and their fulfillment and commitment-based protocols. We analyze the time complexity of CTLC model checking for explicit models and its space complexity for concurrent programs that provide compact representations. We prove that although CTLC extends CTL, their model checking algorithms still have the same time complexity for explicit models, and the same space complexity for concurrent programs. We fully implement the proposed algorithm on top of MCMAS, a model checker for the verification of MASs, and then check its efficiency and scalability using an industrial case study.

Acknowledgments

I would like to warmly thank first of all my thesis supervisor, Jamal Bentahar, for his invaluable support and patience, for his precious and never lacking enthusiasm for research and for his continuous assistance in preparing, writing papers and this thesis. I want to thank him for having so strongly believed in me and provided me with insights which helped me solve many of the problems I encountered in my research. Also, I would like to express my gratitude to my co-supervisor, Rachida Dssouli, whose expertise, understanding and patience, added considerably to my graduate experience. I want to thank her for highly stimulating discussions that have led to the discovery of many results in this thesis. I would like to acknowledge the Ministry of Higher Education, Egypt and my supervisors for respectively supporting and assisting me financially during my Ph.D which gave me opportunities to complete this thesis and attend several international conferences and workshops.

A very special thanks goes out to Hongyang Qu, Oxford University, Computing Laboratory, UK, for his extremely appreciated collaboration that allowed me to implement the model checker presented in this thesis on top of the MCMAS model checker.

I wish to gratefully thank all members in my Faculty of Computers and Information, Menoufia University, Egypt. I also would like to thank all members in the Bureau of Cultural and Educational Affairs of Egypt in Montréal, Canada for their kindly help during my Ph.D study.

My years as a Ph.D student would not have been as much fun without my friends, particularly my colleagues: Sina Honari, Wei Wan, Mohamed Amine and Babak Khosravifar with whom I shared the laboratory and unforgettable moments. From the administrative staff, Mireille Wahba, Lilia Pernatozzi, and Pamela J. Fox are especially thanked for their care and attention.

I would like to thank all my family (Ibrahim, Hamedia, Naema, Naglaa, Hadia, and Ahmed) for the support they provided me through my entire life and for their endless patience.

No, I did not forget you: special thanks goes out to my wife, Warda, and my children (Abdallah and Omar) for being so patient with me, without whose love, encouragement and editing assistance, I would not have finished this thesis; now we can go on holiday.

To the memory of my parents and to my family,

with eternally love and gratitude

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we introduce the context of our research which is about defining a computationally grounded semantics for ACL (agent communication language) messages in terms of social commitments within multi-agent systems (MASs), and developing and implementing model checking techniques to automatically verify commitments and associated actions and commitment-based protocols. We identify the motivations, problems and research questions that we address in the thesis. Moreover, we present the objectives, methodology and contributions of our research work. We conclude with the structure of the thesis.

1.1 Context of Research

1.1.1 Agents and Multi-Agent Systems

An obvious way to open this chapter would be by presenting the term *agent*. After all, this is a thesis about agent communication in MASs, we surely must all agree on what is an agent. Indeed, there is no universally agreed on the actual meaning of the term agent, and there is much ongoing controversy on this very subject. Several computer scientists have observed that the term agent typically denotes a persistent computational entity enjoying some form of *autonomy* and other properties. An agent has in a general consensus the following properties (adapted from [154]):

– *Autonomy*: an agent is capable of operating without direct intervention of others and has a control over its local states and actions.

– *Reactivity*: an agent is capable of responding to external changes in the environment at certain times to reach its goals.

– *Proactivity*: an agent is capable of behaving with respect to its goals by taking the initiative.

– *Social ability*: an agent is capable of interacting with other agents via exchanging messages in an expressive ACL to reach the goals.

With those properties, Wooldridge [153] defined the term agent as follows.

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives.

Nowadays, the world is witnessing an explosive growth in the number of *Web services*. Such services are, for example, paying utility bills, booking airline and hotel reservations, buying and selling goods, managing bank accounts, etc. Human's services can be implemented with greater efficiency and at a lower cost by software agents. Ideally, agents should be sufficiently intelligent to be able to anticipate, adapt and actively seek ways to support human users. However, it is not necessary for an agent to have human level intelligence. An *intelligent agent* is the one that particularly satisfies the capabilities of reactivity, proactivity and social ability [154].

The term *multi-agent system* (MAS) is defined as a set of autonomous agents that are capable of interacting with each other and operating in some *environment* [153]. Figure 1 illustrates the typical structure of a MAS. In the figure, the system contains a number of agents, which can communicate with one another. The agents are able to act in an environment and will also typically be linked by

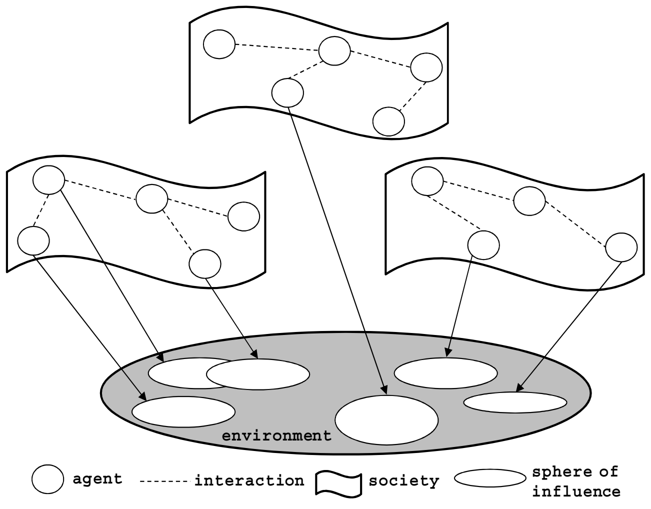


Figure 1: Typical structure of a MAS [153].

other relationships; various agents have different "spheres of influence", i.e., they will have control over different parts of the environment. These spheres of influence may give rise to dependencies among agents [153]. In some formalisms, including the ones employed in the thesis, the environment itself may be modeled as an agent. Moreover, an agent may possibly satisfy the heterogeneity property, which refers to the diversity of the agent constructions in terms of its goals. This property, in turn, leads to a wide range of practical MASs.

We understand MASs as being inherently *open*: in general, MASs are designed in terms of roles and the communication among them modeled with social relations and regulated by multi-agent interaction protocols without reference to any specific agent. In this perspective, MASs can be naturally employed in the description of complex business scenarios, which can be abstracted successfully by ascribing high-level qualities to each agent in the system, and by assuming that agents communicate with other agents to satisfy their goals.

1.1.2 Agent Communication and Protocols

Communication among autonomous and heterogeneous agents is a fundamental aspect to construct effective open MASs. The reason for that is not only because of agents have the property of social ability and our natural attitude to simulate a *human society* where communication is essential, but also because of dynamic systems evolve through communication among participating entities. In such systems and depending on the underlying objectives, entities must communicate to negotiate deals, exchange information, cooperate and even compete with each other in order to satisfy their individual and social goals that they cannot achieve alone. Another important reason is that most applications of MASs ranging from digital libraries [23], cooperative engineering [154], artificial institutions [64], electronic commerce [153], Web service compositions [93] to cross-organizational business processes [139], have one thing in common: the agents operating in these systems have to communicate. These systems consist of multiple agents that communicate in order to solve some problems. If a problem is particularly complex, large, or unpredictable, the only way it can reasonably be addressed is to develop a number of functionally specific and modular components (agents) which are able to solve a particular problem aspect [136]. This decomposition allows each agent to use the most appropriate paradigm to solve its particular problem. When interdependent problems arise, agents must somehow communicate in order to coordinate with one another to ensure that interdependencies are properly managed. Therefore, it is clear that the success of these systems requires commonly understood languages: *lingua franca* for agents to 'talk' to each other to decide what information to exchange or what action to take as well as powerful mechanisms (protocols) to regulate and structure communication among participants within dialogues and conversations.

In the early days of MAS research, a promising way to model agent communication was widely inspired by Searle's speech act theory [116], which was particulary concerned with identifying actions performed by agents to satisfy their intentions. This is the so-called *mental approach* that focuses on achieving a rational balance between some notions of interacting agents such as beliefs, goals, desires and intentions. Under this doctrine, it became apparent that a purely mental semantics for ACL messages in terms of pre- and post-conditions of the agents' mental states would necessarily impose significant restrictions on the operational behavior of agents. For example, in the Agent Communication Language of the Foundation for Intelligent Physical Agents (FIPA-ACL)[1] that uses this doctrine, the semantics of *Inform act*, where the sender tells the receiver a proposition p, says that such an act may be only uttered if the sender believes the proposition p to be true in the pre-condition part, called specifically sincerity condition. The problem with this approach is that the addressee agents (or an external observer agent) cannot verify whether the speaker agent violates the pre-conditions defined in such mental semantics or not [125] as of course there is no access to agents' mental states. This problem is also known as the *semantics verification* problem [152]. Accordingly, such a pure mental semantics cannot be used in proprietary open systems wherein sincerity cannot be taken for granted as the interacting agents are heterogeneous [123, 125]. Moreover, the pure mental semantics makes ACLs not general enough to capture the interoperability among heterogeneous systems [123].

When conversing, agents do not exchange isolated messages, but a sequence of messages. The developers of FIPA-ACL, for example, have addressed the challenge of incorporating ACLs and protocols by proposing a set of conversation protocols, called FIPA-ACL protocols[2]. These protocols can be viewed as specific ACLs designed for particular purposes, such as *Request Interaction protocol*, *English Auction Interaction protocol* and *Contract Net protocol*. For instance, the English Auction Interaction protocol is designed to help auction goods get a higher market price that is recently used in mobile commerce and the Contract Net protocol is designed from online business point of view to reach agreements among interacting agents. FIPA-ACL protocols have succeeded in specifying the rules governing interactions and coordinating dialogues among agents by: (1) restricting the range of allowed follow-up communicative acts at any stage during a dialogue; and (2) describing the sequence of messages that FIPA compliant agents can exchange for particular applications. However, these protocols are too rigid to be used by autonomous agents because they are specified

[1]See FIPA-ACL specifications (1997,1999,2001,2002), `http://www.fipa.org/repository/aclspecs.php3`
[2]See FIPA-ACL Interaction Protocols (2001,2002), `http://www.fipa.org/repository/ips.php3`

so that agents must execute them without possibility of handling exceptions that appear at run time, which restricts the protocols' flexibility.

Therefore, the MAS community witnessed a shift from individual agent representations to social interaction approaches to particulary overcome the shortcomings and inconveniences incorporated with mental approaches [123]. Indeed, such a shift is recently asserted by agent communication community in [29]. Commitments are employed in some of these social approaches, which successfully provide a powerful representation for modeling communications among autonomous and heterogeneous agents. In broad terms, commitments are social, public, objective [35] and help represent the states of affairs at different instants in the course of multi-agent interactions. Conventionally, a social commitment is made by an agent playing the role of debtor and directed towards another agent playing the role of creditor to bring about a certain condition [124]. Recent communication approaches based on social commitments have seen progress in a variety of areas, such as modeling business processes [40, 137, 139], developing artificial institutions [64], defining programming languages [128], modeling service-oriented computing [129], enhancing agent-oriented software engineering methodologies [97, 138, 45], developing Web-based MASs [143], developing agent-based Web services and their communities [10, 45], specifying commitment-based protocols [159, 160, 5, 39, 98, 49], and specifying business protocols [40, 42, 47]. In particular, commitment-based protocols, specified simply as a set of commitments and their actions, are more suitable for regulating and coordinating agent interactions than FIPA-ACL protocols and traditional computer protocols formalized using *finite state machines* and *Petri nets*, which only capture legal ordering of exchanged messages without unduly considering the meaning of those messages. Missing such meaning limits the ability to verify the compliance of agent behaviors with a given protocol. Also, expressing protocols using commitments allows these protocols to be flexible, meaningful and verifiable [11, 48, 50, 160, 157] without over-constraints on the communications [159, 160]. In commitment-based protocols, agents will not reason about legal sequences but about concrete commitment states and possible paths[3] to reach them [158].

1.1.3 Verification of Agent Communication Protocols

The verification problem of agent communication protocols is fundamental for the MAS community [11, 50]. It is unrealistic—in open systems—to presuppose that all autonomous agents will behave according to the given protocols as they may not behave as they are committed to or at least not wantonly violate or cancel their commitments. In this perspective, verification process is

[3]A path is an infinite sequence of system's states.

a noteworthy attempt to help protocol designers either detect unwanted and bad agents' behaviors to eliminate them or enforce desirable agents' behaviors so that protocols comply with particular specifications at design time. Verification process also reduces the cost of development process and increases confidence on the safety, efficiency and robustness of protocols. In contrast to MASs, verification techniques—especially formal ones—for commitment-based protocols, social commitments, and associated actions that manipulate commitments and capture dynamic behavior of interacting agents, are still in their infancy because most contributed proposals: (1) abstract commitments as predicates [35, 99, 135, 145], fluents [24, 67] or domain variables [6, 39, 65, 98, 107, 139, 156], which lack real and concrete meaning of commitments; (2) model commitment actions as axioms on the top of the commitment semantics [24, 26, 35, 41, 140, 160, 143], which do not account for interactions that are central to real-life business scenarios and waive the interoperability and verification issues; and (3) devote action logical languages to specify, model, execute and reason about protocols [24, 26, 41, 140, 160], which cannot be directly model checked.

1.2 Motivations

To be able to communicate, agents should use a common communication mechanism, such as commitment-based protocols. These protocols support flexible executions that enable agents to exercise their autonomy by reasoning about their actions and making choices [159]. However, the languages [24, 26, 41, 140, 160] used to specify such protocols are not suitable for model checking, a formal and automatic verification technique. Our first motivation is to define a new specification language of commitment-based protocols, which can be model checked and balance between expressiveness and verification efficiency aspects.

In addition, in the domain of agent communication, formal semantics conventionally lays down the foundation for a neat, concise and unambiguous meaning of agent messages, and provides capabilities to verify at design time if agent behaviors comply with the defined semantics and also facilitates and improves the applicability of the proposed semantics. The only way to achieve these goals is to follow current proposals that model commitments as modal operators and give them a computationally grounded semantics by linking real situations to computational models. Thus, the second motivation of this thesis is to develop a new computational logic (branching-time temporal logic) by extending existing temporal logics with modalities to represent and reason about social commitments and associated actions. The introduction of a new temporal logic is motivated by the fact that the needed modal operators for reasoning about social commitments and associated actions cannot be expressed using existing temporal logics. The resulting logical language aims to derive a formal specification language of commitment-based protocols, required in the first motivation.

Our third motivation is to contribute to the advance of research in this domain by investigating the possibility of applying different model checking techniques to verify commitments and associated actions together with commitment-based protocols, given some desirable properties so that the specification of those protocols is error-free.

1.3 Problems and Research Questions

Current modal logic proposals for commitments [11, 12, 16, 17, 35, 46, 48, 125, 127] define first an accessibility relation and then use it to define the semantic rules of the commitment modality. For instance, in Singh's proposal [125], the accessibility relation produces the set of accessible paths along which the commitments made by one agent towards another agent hold. When the content of commitment is true along every accessible path produced by the accessibility relation and emanated from the commitment state, then the commitment modal formula holds. This type of rules capture the semantics of commitment modality. However, these proposals do not clarify the intuition an accessible path and state capture and how they are computed, which limits their applicability for verification purposes. Thus, the first problem that we address in this thesis is to define a formal (which is based on computational logic), declarative (which focuses on what the message means instead of how the message is exchanged), meaningful (in which every message is expressed in terms of commitments), and computationally grounded (which means modal formulae should be interpreted directly with respect to a computational model and vice versa) semantics for ACL messages in terms of commitments and associated actions. Indeed, defining a computationally grounded semantics for commitments is entirely missing in the literature of agent communication. The *computationally grounded theories* of agency have been first introduced by Wooldridge [151]. The notion of computationally grounded semantics is very useful when we are to consider agents' commitments as *a defacto* formal specifications for computational MASs. As we mentioned, the semantics of commitment actions is defined as axioms. These axioms are represented either as reasoning rules [26, 66, 134, 159, 160], updating rules [36, 62, 63] or postulating rules [127] to evolve the truth of commitment states and to reason about commitment actions to explicitly accommodate exceptions that may arise at run time [159, 160]. However, the real meaning of commitment actions themselves are not considered. Thus, our initial research questions are: How to define a suitable and meaningful semantics of social commitments and associated actions? and How the semantic link between commitment and its fulfillment is defined?

Developing a new modal logic for commitments and their fulfillment is not an easy task as we need to define our deemed appropriate axioms of commitments and their fulfillment, which correspond to the properties of social accessibility relation. These axioms usually define a class of well-formed formulae that are valid (i.e., those that are true in every state of every model) and capture the properties of commitments and their fulfillment in a reasonable way without any reference to their meanings. The reasonable question that we explore here is: How can we present deemed axioms of commitments and their fulfillment?

In the literature about commitment representations, a compelling view is adopted to model the interacting agents from high-level abstractions without considering agents' states that constitute the life cycle of each agent, agents's actions and how agents can select their choices at each state. All the above concepts need to be explored and the interacting agents should be modeled. The questions that we explore here are: How should an agent and a system of agents be formalized? and How the resulting formalism can be extended to account for communication?

To address the challenge of applying model checking to verify commitments and commitment-based protocols, some proposals used: local testing technique [143]; static verification technique [24, 98, 140, 160]; and semi-automatic verification technique [157] to identify the compliant and non-compliant agents at the end of the protocol. These verification techniques can be considered as simple reasoning tools about the correctness of protocols. Other proposals verified commitment-based protocols using different model checking techniques. However, these techniques are based on reducing commitments and their actions into domain variables using informal translation-based approaches to be able to use existing model checkers. Such informal translation-based approaches [12, 11, 48, 49, 66, 67] have the problem of forbidding verifying the real semantics of commitments and associated actions as defined in the underlying logics. They only provide partial solution to the problem of model checking commitments as they reduce commitment modalities into simple variables [11, 39, 48, 65, 139]. Moreover, there are no tools supporting the informal translation-based approaches to carry out the translation process. We then believe that a formal and automatic translation-based approach is more suitable as it allows representing commitment modality in other temporal modalities, which still preserve its meaning. Thus, the questions that we explore here are: How can we formally define a new specification language of commitment-based protocols? and How can we formally reduce the problem of model checking the proposed logic into an existing one? Another solution to the problem is to develop new model checking algorithms for new modalities and associated actions. The question that we consider here is: How can we develop and implement an algorithm for model checking commitments and their actions and commitment-based protocols? The complexity analysis in terms of time and space of the new model checking algorithm of the proposed logic must be addressed as well.

1.4 Objectives

The main objectives of this thesis are:

1. To develop a new branching-time temporal logic, which is particularly used to cater a rigorous and well-understood framework for representing and reasoning about time, social commitments and associated actions: this objective comprises the extension of *Full Computation Tree Logic* (CTL*) introduced by Emerson and Halpern [58] with modalities for social commitments and associated actions. The election of CTL* is motivated by our objective to achieve a high expressive and succinct logic as CTL* subsumes *Linear Temporal Logic* (LTL) [109] and *Computation Tree Logic* (CTL) [31, 56].

2. To specify commitment-based protocols: this objective consists in using the proposed logic to derive a new specification language of commitment-based protocols.

3. To investigate model checking techniques for the verification of commitment-based protocols: this objective comprises the development of reduction tools that transform the problem of model checking our logic into the problem of existing model checking algorithms in order to directly use existing model checkers. We also aim to use our tools to automatically verify the correctness of different business protocols, given desirable properties to evaluate the effectiveness of our tools.

4. To develop a dedicated model checker for commitments and their fulfillment and commitment-based protocols: this objective consists the development of a new symbolic algorithm for our logic to implement a software tool for model checking commitments and their fulfillment and commitment-based protocols so that industrial case studies can be verified.

1.5 Methodology

At the beginning of this thesis, we have studied research work done in the domain of agent communication, particulary how computational logics can be devoted to define a formal semantics for agent communication in terms of social commitments and their actions. We have noticed that the current formal semantics of commitments [12, 11, 48, 49, 125, 127] and their actions [26, 36, 62, 63, 66, 134, 159, 160, 127, 145] used in this field are not for the most part grounded (i.e., clear in their assumptions and computations). For this reason, such semantics are hard-wired in their implementation and evaluation, and then restrictive the applicability of verification and interoperability issues. More precisely, such a semantics cannot be directly verified using model checking as they need to be translated first into a dedicated input language of an existing model checker. With respect to the notion of computationally grounded theories of agency, we defined a new computationally grounded semantics for commitments and their actions, which has the advantages of having a concrete interpretation and being semantically verifiable. Thus, we believe that such a semantics is useful for reasoning about commitments and computational MASs. Moreover, we noticed that all logical models representing commitments and their actions waive discussing some theoretical issues, such as valid axioms. We had the idea of presenting a set of reasonable axioms capturing our deemed properties of commitment and fulfillment modal operators.

As the internal structure of an agent is a silent aspect in all existing agent communication models, we used the formalism of interpreted systems [59], which provides a mainstream framework to model MASs and explore their fundamental classes such as synchronous and asynchronous. It is also used to interpret MAS properties expressed in different temporal logics, such as the logic of knowledge (or epistemic logic). We advocated this formalism for many reasons, for instance, it allows us to abstract from the details of the components and focus only on modeling key characteristics of the agents and the evolution of their social commitments. More particulary, we extended this formalism to account for communication among agents and introduced a set of shared and unshared variables that can be seen as an abstraction of *message-passing systems* described in [59].

Although certain researchers recently started to emphasize the importance of verifying MAS, this aspect has yet to be explored in the field of agent communication. In this field, only a small amount of research work addressed this complicated issue, for examples [11, 39, 48, 65, 139]. Technically, these examples suffer from many limitations resulting from the use of informal translation-based approaches, as we mentioned above. For this reason, we have advocated two methods: (1) formal translation-based approaches by developing reduction tools, which we call indirect method; and (2) direct method by developing a new model checking algorithm in order to verify commitments and their actions and commitment-based protocols. Indeed, each method introduces a number of benefits with respect to existing approaches and allows us to automatically verify commitments and their actions and protocols effectively and efficiently as we will show through this thesis. We implemented these tools on top of the CWB-NC model checker[4], and the extended version of NuSMV model checker [104], while our algorithm is implemented on top of the MCMAS model checker [92]. We also analyzed its time and space complexity, which have not been addressed yet.

1.6 Contributions

The majority of the results presented in this thesis have been published in the proceedings of various international conferences and workshops and international journals. In summary, the main contributions are:

1. A semantics of commitments and all associated actions formalized by extending CTL* with commitments and their actions to ACTL^{*c} [46, 48, 55], a formalization of commitment-based protocols in the underlying logical model, and an automatic verification technique of these protocols by means of reduction-based model checking the underlying logic [49, 47, 53, 54, 55].

2. A semantics of commitments formalized by extending CTL to CTLC (CTL for commitments), which is the first proposal that uses the formalism of interpreted systems to model MAS in which the communication among its members is modeled by social commitments [50].

3. Two model checking reduction methods for CTLC along with the model checking computational complexity [51].

4. A dedicated symbolic model checking algorithm for CTLC and its implementation on top of MCMAS [8, 52].

The next section shows how different chapters are linked to those contributions.

1.7 Overview of the Dissertation

This thesis is divided into two parts.

Part I is about the state of the art, and it consists of two chapters:

– **Chapter 2** introduces the notion of agent communication and philosophical foundations of agent communication languages (ACLs). The chapter discusses very briefly two examples of ACLs and their semantics as well as the notion of conversation protocols. Furthermore, the chapter summarizes some background material on computational logics (e.g., propositional

[4]`http://www.cs.sunysb.edu/cwb/`.

logic, predicate logic, etc.) and temporal model checking techniques and tools. This material enables the introduction of some technical details needed in Chapter 3.

– **Chapter 3** presents literature review of prominent proposals that have been devoted to computational logics to define a semantics for ACL messages in terms of social commitments and to specify commitment-based protocols. The chapter reviews how commitments are modeled and their formal semantics (if any) as well as different verification techniques proposed to verify these protocols. Also, the chapter explores other action logical languages proposed to specify and execute protocols. The key point of this review is to highlight advantages, and limitations in existing proposals. These limitations will be addressed in our thesis.

Part II consists of three chapters in which we present our contributions:

– **Chapter 4** proposes a new logic-based language to specify commitment-based protocols, which is derived from ACTL^{*c}, a logic extending CTL* with modalities to represent and reason about commitments and all associated actions. To verify commitment-based protocols, the problem of model checking ACTL^{*c} is reduced to the problem of model checking GCTL* so that the CWB-NC model checker is possible. The soundness of our reduction method is proved. This chapter presents our contribution 1.

– **Chapter 5** investigates two different implementations for the problem of model checking a fragment of ACTL^{*c} (CTLC) by means of a reduction method with respect to both automata-based techniques and symbolic techniques and proves the correctness of the employed reduction methods. This chapter presents our contributions 2 and 3.

– **Chapter 6** presents a new devoted symbolic algorithm to perform model checking of CTLC and analyzes the time complexity of CTLC model checking in explicit models and its space complexity in concurrent programs, which provide compact representations. The implementation of the model checker MCMASC, an extended version of the MCMAS model checker, is presented. The chapter then discusses the effectiveness and scalability of MCMASC against an industrial case study. This chapter presents our contributions 3 and 4.

Chapter 7 summarizes the obtained results in this thesis, presents open issues and sketches possible extension of this work.

Appendix A presents a new symbolic model checking algorithm dedicated to ACTL^{*c}.

Chapter 2

Background

In this chapter, we present some preliminaries, which are relevant for the rest of the thesis. This chapter is organized as follows. In Section 2.1, we discuss the notion of agent communication, the philosophical foundations of agent communication languages (ACLs), the two popular languages developed for ACLs and their semantics and the notion of conversation protocols and their formalisms used for years in proprietary multi-agent systems (MASs). We finally highlight their limitations, which we address in details in the next chapters. In Section 2.2, we briefly explore computational logics exploited as the main ingredient in existing approaches to agent communication, which we will review in Chapter 3. These logics are mainly used to define a formal semantics for ACL messages in terms of social commitments. In Section 2.3, we provide the relevant background of model checking techniques and tools, which we will be using in Chapter 3 to verify commitments and associated actions and commitment-based protocols. Section 2.4 ends the chapter by a discussion about the motivations of the thesis and the link of the present chapter with the rest of the thesis.

2.1 Agent Communication

In a typical multi-agent system, the knowledge and functionality of the system is distributed among the constituent agents. Agents must have methods for sharing knowledge and taking advantage of each other capabilities as needed. It would not be feasible for every agent to have complete knowledge of the entire system, nor to have all possible capabilities. In order to cooperate effectively, agents need to have standardized methods for exploiting each other resources. When an agent desires to achieve a goal which it cannot satisfy alone, it needs to be able to find an agent which may be able to help and a method by which it can expect to get that help. That is, it must communicate its needs using a standardized language and the recipient must be able to understand the request, and respond appropriately. The language used needs to be sufficiently expressive to allow agents to transmit complex information and goals. There is also a need for a system of agents to communicate to negotiate some attributes of their deals such as price of goods, delivery terms or payment conditions in order to achieve a mutual agreement about their internal goals. Agents also need to cooperate with other peers in order to satisfy both their internal and collective goals generated by virtue of their participation in the social community [83]. As a result, communication among agents is now considered one of the key mechanisms for building MASs.

Wooldridge [153] has pointed out that communication has long been recognized as a topic of central importance in concurrent systems, evident by the existence of many formalisms developed for dealing with systems whose members can synchronously interact with one another. The communication in such systems is treated in a low-level detail through shared data structures. Because agents are autonomous, they can neither force other agents to perform some action, nor write data into the internal state of other agents. However, this does not mean that agents cannot communicate. Indeed, what they can do is performing actions in an attempt to influence other agents appropriately, i.e., linguistic communication is just a special type of actions.

Singh [123] has contended that interoperability and autonomy are the key concepts that set agents apart from conventional objects, which always satisfy every method invoked on them and also communicate by method invocation, while agents should be able to refuse an action. In fact, an agent-based software integration was conceived to help heterogeneous software components modeled as agents interoperate within modern applications like electronic commerce. So, how is communication performed among interoperable agents? There is no choice for an agent, except to affect another agent or similarly be affected by another agent, by performing an action.

We underline that communication differs from physical interactions, and is unique among the kinds of agents' actions that those agents may perform through their participation in conversations and dialogues. *Acting by speaking* is the essence of communication. The artificial languages used by agents for their communications are called *agent communication languages* (ACLs). Some ACLs have already been developed but none has been widely adopted as a standard [123]. When agents join a group, we need mechanisms for interactions, which have the properties we desire. A mechanism (protocol) for an interaction is a set of rules by which the interaction will be conducted. These protocols are inherently tied to an ACL. Research on agent communication has devoted a lot of attention to languages and protocols; the literature contains many more proposals for semantic definitions and protocol specification methods than it does complete ACLs.

The remainder of this section is organized as follows. In Section 2.1.1, we present the philosophical foundations of ACLs and the two most popular ACLs. In Section 2.1.2, we briefly look at different approaches to the task of specifying conversation protocols and their limitations. In Section 2.1.3, we discuss the main issues in mental approaches proposed to define the semantics for ACL messages and social approaches and then establish the link with the next chapter.

2.1.1 Philosophical Foundations of ACLs

ACLs are based on a philosophical theory called *Speech Act Theory* (or sometimes called *Illocutionary Act Theory*). This theory is a high-level theoretical framework that was developed by philosophers to account for certain class of natural language utterances. It has originally been formulated by the British philosopher John Langshaw Austin [3] and reproduced and extended by the American John Rogers Searle [116] and by Searle and Vanderveken [117] to be a more general model for communication among artificial agents.

Speech act theory defines a set of actions and associates a meaning with these actions. These actions, called *speech acts*, have the characteristics of changing the state of the world in a way analogous to physical actions. For this reason, speech acts or communicative acts are often known as performative verbs, but indeed they are not classified as being true or false. Austin distinguished three different aspects of speech acts: (1) *Locutionary act*: saying something with a certain meaning in a traditional sense, e.g., saying please make some tea; (2) *Illocutionary act*: have a certain 'force' to the receiver in saying something, e.g., informing, ordering, warning, undertaking; and (3) *Perlocutionary act*: bring about or achieve something, e.g. convincing, persuading, deterring. For example, if the illocutionary act was arguing, and if the receiver was convinced, then the perlocutionary act is an act of convincing. Recall that all locutions do not count as illocutions, since some of them may occur in an inappropriate state of the world, e.g., when no receiver is available. Also, all perlocutions are not caused by appropriate illocutions, since some of them may occur because of other contextual features. Searle [116] identified five illocutionary points:

– *Assertives*: statements may be judged true or false because they aim to describe a state of affairs in the world (e.g., tell and claim).

– *Directives*: statements attempt to make the other person's actions fit the propositional content (e.g., command and request).

– *Commissives*: statements which commit the speaker to course of action as described by the propositional content (e.g., commit and promise).

– *Expressives*: statements that express the “sincerity condition of the speech act” (e.g., thank).

– *Declaratives*: statements that attempt to change the world by “representing it as having been changed” (e.g., declare).

The illocutionary act is the minimal unit of linguistic communication. Illocutionary acts can be decomposed into an illocutionary force and a propositional content [117]. Consider the acts performed by the following two sentences: (1) John will leave the room; (2) John, leave the room. The propositional content is clearly the same which is John performing the act of leaving the room; so we can say that the proposition specifies the state of the world that it holds currently. The illocutionary force is different, the first is a prediction (a type of assertion) while the second is an order. Moreover, Searle [117] distinguished between two rules of interaction: (1) *Constitutive rules or Definition rules* that create or define new forms of behavior and they do so by defining the semantics of acts; and (2) *Regulative or Behavior rules* that govern types of behavior that already exist. In other words, they rule the flow of previously constituted behavior.

Speech act theory has been adopted in the field of MASs in the early of 1990s, being used as an important piece of the design of languages for communication among agents. Following the ideas of the speech act theory [117], ACLs make a distinction between:

– The intentional part of the message (the message container). An ACL defines the syntax of different types of messages, in a way that allows agents communicate their intentions. Usually this is done by adding a performative field in the message structure to describe the illocutionary force of the message. Each ACL defines a set of valid performatives and their semantics. These semantics can be used by the receiver agent to interpret, from the performative in the message, the intentions of the sender agent.

– The propositional content (the message body). Some ACLs define how the content of the message should be expressed. When an ACL is mainly focused in the message body, it is called *Content Language*.

Agent communication specifications usually deal with agent communication languages, messages exchange, conversation protocols, communicative acts, and content language representations. Agent communication languages define how messages should be structured in order to avert misunderstanding between parties. The most popular ACLs are: KQML and FIPA-ACL. Both rely on speech act theory, described in the above. The content of their performatives is not standardized, but varies from system to system.

A) KQML

In the early of 1990s, the DARPA knowledge sharing effort (KSE) began to develop the Knowledge Query and Manipulation Language (KQML), which became the first *de facto* standard for ACL [60] in several areas. KQML is a language and protocol for exchanging information and knowledge. The overarching aim of the KSE is to develop techniques and methodologies for building large scale knowledge bases which are sharable and reusable. Knowledge Interchange Format (KIF) is the content language suggested by the KSE, but KQML messages can use any language for content.

The main primitives of this language are performatives (i.e., message types). As the term suggests, the concept is related to the speech act theory. Performatives define the permissible actions such as *ask-about, tell, ask-if* and *ask-one* that agents may attempt when communicating with each other. The syntax of KQML messages consists of a performative and a number of parameters. The following is an example of KQML message:

```
(ask-about
            :sender Customer
            :receiver Merchant
            :language LPROLOG
            :ontology selling and delivery goods
            :reply-with id12345
            :content(PRICE AAMAS PROC? price)
)
```

In KQML terminology, `ask-about` is a performative. The intuitive interpretation of the `ask-about` message is that the sending agent is asking the receiving agent about some information where exactly one reply is needed. The example indicates that the customer agent asks the merchant about the price of `AAMAS PROC` (proceedings). The various other components of this message represent its attributes. The message uses the `LPROLOG` language to describe the content and a particular ontology named `selling and delivery goods`, which indicates the significance of the parameter `PRICE AAMAS PROC` that will hold the value of the requested price. Any response to this KQML message will be identified by `id12345` in the filed `reply-with`.

The KQML does not specify how the language should be implemented, but it does define some abstract requirements that are part of the KQML model, these include: the ability to send and receive KQML messages that access the knowledge of the agent; the ability to interact asynchronously with more than one agent at the same time; and the presence of a facilitator agent. The communication facilitator is a special agent whose role is to coordinate the interactions of other agents. The original specification for KQML did not give it a semantics, this was one of the major criticisms leveled against it. Near the end of the 1990s, Labrou and Finin have proposed a framework to define KQML's semantics and limit the use of some performatives in order to avert confusion and ambiguity problems [87]. This semantics is defined in terms of: (1) preconditions on the mental states of the sender and receiver before exchanging the message; (2) postconditions that should hold after the message is sent; and (3) completion conditions that indicate the message effect.

The semantics of KQML were never rigorously defined, in such a way that it was possible to achieve the interoperability and heterogeneity issues [123]. The meaning of KQML performatives was merely defined using informal, English language descriptions, which opens the door to different interpretations. Moreover, Wooldridge has pointed out that the language of KQML was missing the class of *commissives*, and it is difficult to implement many multi-agent scenarios without considering commissives that appear to be important when agents are called to coordinate their actions or to communicate with one another [153]. These grounded criticisms led researchers to define a new language: FIPA-ACL.

B) FIPA-ACL

In the end of the 1990s, the Agent Communication Languages of the Foundation for Intelligent Physical Agents (FIPA-ACL)[1] arose from attempts to develop an industry and academia standard for agent systems. It provides a set of specifications that can be utilized by agent system developers as part of their solutions. The term performative is identified by a verb such as *tell* or *ask* which is the core meaning of a speech act.

FIPA-ACL distinguishes two levels in communication messages. At the inner level, the content of messages can be expressed in any logical language. The outer level describes the locutions that agents can use in their communication. The FIPA-ACL's message contains the following attributes: performative, sender, receiver, content, ontology, and language, which closely resemble that of KQML. FIPA-ACL specifies 22 performatives. For instance, the informal meaning of the *request* act is that the sender requests the receiver to perform some action. One important class of the request act is to request the receiver to perform another communicative act.

Given that one of the most frequent criticisms of KQML was lack of an adequate semantics; it is perhaps not surprising that the developers of FIPA-ACL felt the importance to give a formal semantics to their language. Also, the hope is that FIPA-ACL's formal semantics will offer a rigorous basis

[1]FIPA-ACL message structure specification. Available: http://www.fipa.org/specs/fipa00061/

for interoperability. Such formal semantics were given with respect to a well-defined formal language called *Semantic Language* (SL). FIPA-SL[2] is a quantified multi-modal logic with several referential (*any, all*) and action (*feasible, done*) operators. FIPA-SL is capable of representing propositions, actions and representing objects including identifying expressions to describe the objects. Also, FIPA-SL contains modal operators for beliefs, uncertain-beliefs, persistent goals, and intentions, which form the mental model of an agent in which the semantics of these operators are given using possible world approach. FIPA-SL2 is the most complete at the moment and allows first-order and modal operators to be expressed. Indeed, the idea of SL semantics is to map each ACL message into a formula of SL, which defines a constraint that the sender of the message must satisfy when it is to be considered as conforming to the precondition of FIPA-ACL standard. FIPA-SL refers to this constraint as the *feasibility precondition.* The SL semantics also map the postcondition of message into the *rational effect* of the action: what an agent will be attempting to achieve by sending the message. For example, the FIPA-SL semantics for the *request act* is defined as follows:

$\langle i, request(j, \alpha)\rangle$

feasibility precondition: $B_i\ Agent(\alpha, j) \wedge \neg B_i I_j Done(\alpha)$

rational effect: $Done(\alpha)$

The SL expression $Agent(\alpha, j)$ means that j is the agent who performs the action α; and $Done(\alpha)$ means that the action α has been done. Thus, agent i requesting agent j to perform action α means that agent i believes that the agent of α is j, and so it is sending the message to the right agent and agent i believes that agent j does not currently intend that α is done. The rational effect—what i wants to achieve by sending this request message—is that the action is done. As FIPA-SL semantics is belief-based [147], it appears to be repeating the same mistake of KQML of emphasizing mental agency [123]. It is impossible to make such a semantics work for the interoperability of agents that must be autonomous and heterogeneous as it requires the agents to reason about each other's beliefs and intensions and behave cooperatively and sincerely (i.e., it assumes that agents can read (or at least model) each other's minds [123]). FIPA's assumption of the sincerity of agents has often been criticized [81, 152]. For the fact that its semantics is not public and fails to be verifiable where there is no way to test whether or not agents respect the semantics of the language whenever they communicate [15, 29, 123, 125, 152]. These criticisms encouraged researchers to define new formal semantics for ACLs to overcome these problems.

[2]FIPA SL content language specification. Available: http://www.fipa.org/specs/fipa00008/index.html

2.1.2 Conversation Protocols

In the previous section, we have discussed the issues primarily related to the generation and interpretation of individual ACL messages. However, agent communication is generally regarded as the foundation for cooperative and competitive behavior of autonomous and heterogeneous agents, which requires a sequence of interdependent messages, or conversations, rather than individual messages. To take into account this aspect, every agent must implement decision procedures, which allow for the selection and production of ACL messages that are appropriate to the agent's goals and intentions. These decision procedures can be used to ensure reliable communication among heterogeneous agents by taking into consideration the context of prior messages to know which communicative acts are possible for the next message in a conversational sequence. In fact, a communicative act does not have absolute meaning, but its semantics relies on the conversation. Furthermore, Dignum and Greaves [43] pointed out that decision procedures define patterns of communication, which "can actually simplify the computational complexity of ACL message selection" by providing a context in which ACL messages are interpreted. The specification of these conversations is achieved by means of *conversation protocols* (also called *conversation policies*).

Many ACL designers have included conversation policies as part of the ACL definition. The specification of conversation protocols has been traditionally done by making use of natural language descriptions [60]. A definite clause grammar has subsequently been used for KQML policy specification [87]. Petri nets have also been used to formalize policies [37]. FIPA has additionally produced a finite state machine style method by which conversation protocols, called *agent interaction protocols* (AIPs), are specified diagrammatically. These specifications were only semi-formal, relying heavily on accompanying text descriptions. To tackle these shortcomings, FIPA has subsequently employed AUML(Agent UML)[3] style diagrams and powerful graphic design languages for describing protocol specifications in which a sequence diagram in AUML shows the chronological sequences of messages between agents in a sequential order.

Unfortunately, the aforementioned formalisms vary widely in the degree of conversational rigidity they entail, due to the fact that such protocols are specified so that agents must execute them without possibility of handling exceptions that appear at run time, which restricts the protocols' flexibility [15, 70]. Such rigid protocols are not suitable for structuring and coordinating interactions among autonomous agents. Technically, such formalisms provide procedural meanings, which are low-level details and are more appropriate for computational entities which lack the rationality to

[3]FIPA interaction protocol library specification, 2003. Available: http://www.fipa.org/specs/fipa00025/

plan their own behavior. Multi agent systems are supposed to provide higher level abstractions than traditional distributing systems [123]. When the protocol is the unit of communication and speech acts are defined only in terms of possible replies, then communication becomes essentially an ordered exchange of meaningless tokens and the language is not sufficiently expressive [125]. To solve this problem, several researchers defined protocols, for example, using dialogue games. Dialogue games are interactions between players in which each player moves by performing utterances according to a predefined set of roles. The flexibility is achieved by combining different games to construct complete and more complex protocols. For instance, Bentahar et al. formalized these dialogue games as a set of conversation policies to define *persuasion dialogue game* protocol [15] and *persuasive-negotiation* protocol [9].

2.1.3 Discussion

From the syntactic perspective (the different illocutionary acts, speech acts or performatives as developed in speech act theory that agents can perform during a conversation), the definition of an ACL poses no problem. Suppose that we are given an agent and an ACL with a well-defined semantics. And we aim at determining whether or not the agent respects the semantics of the ACL. Wooldridge [152] pointed out that *syntactic conformance testing* is of course easy. The situation is very difficult when we need to determine whether or not a particular agent program respects the semantics of the ACL. This is called *semantic conformance testing* [152, 153]. The reason for the difficulty is that we would have to be able to talk about the mental states of agents—what they believed, intended and so on. In fact, the FIPA-ACL specification acknowledges that conformance testing "is not a problem which has been solved in this FIPA specification. Conformance testing will be the subject of further work by FIPA". While any ACL must have a well-defined semantics, given that agents in a MAS may be heterogeneous, a clear understanding of semantics is essential. Both KQML and FIPA-ACL have been given a mental semantics because they define the meaning of speech acts in terms of the mental states of participants. This semantic definitions could be said to constitute the mental approach that is fraught with the following issues:

1. It is difficult to verify compliance[4] with a semantics which is based on internal states if those states are inaccessible, as is the case in open systems where agents from different design need

[4]The issue of developing semantics for ACLs has been examined in [152], by considering the problem of giving a verifiable semantics, i.e. a semantics grounded on the computational models. The author gives an abstract formal framework, in which he defines what it means for an agent program, in sending a message while in some particular state, to be respecting the semantics of the communicative action. The author also points out the difficulties of carrying out the verification of this property when the semantics is given in terms of mental states, since we do not understand how such states can be systematically attributed to programs.

to understand each other [13, 125, 152, 153].

2. Agent's autonomy is limited [123]. Indeed, any ACL specification limits execution autonomy by requiring agents to be cooperative, competitive, and so on. However, the execution autonomy is very low in FIPA-ACL as it constrains agents to satisfy sincerity condition.
3. Meaning of communication is locked in context [123, 153]. When a speech act is given a precise meaning in terms of the participants' mental states, there is a lack of flexibility to use the act in a different context. Ideally, the meaning of an act should depend on the context in which it is uttered [43].

To address these issues, the most promising approach, first proposed by Singh in [123], is the social approach to define the semantics of ACLs in terms of social notions, emphasizing conventional meaning and public perspective [125]. Following social approach, communicative actions affect the "social state" of the system, rather than the internal (mental) states of the agents. The social state records social facts, such as the commitments of interacting agents. The social approach allows a high level specification of the protocol in terms of a set of commitments and their actions and does not require the rigid specification of the allowed action sequences as in the above formalisms. Representing commitments as an explicit first class object leads to flexibility when protocols are realized on the fly in changing situations, for example electronic commerce. Commitment-based protocols are flexibly formalized by means of *commitment machines* [158], which enable agents to exercise their autonomy by dealing with exceptions and making choices. In addition to providing flexibility during run time, social approach makes it possible to provide a meaningful basis for compliance of agents with given protocols. This is because commitments can be stored publicly and agents that do not satisfy their commitments at the end of the protocol can be identified as non-compliant [160, 50]. To this end, social approach is well suited to deal with "open" multi-agent systems, where the history of communications is observable, but the internal states of the single agents may not be accessible. We conclude with the following three consensuses that have been recently identified by the agent communication research community [29]: (1) "any semantics for communication in open systems must be underpinned in social, not mentalist abstractions"; (2) "the FIPA-ACL semantics is not suitable for specifying communication protocols in open settings"; and (3) "the notion of commitment is an important abstraction for formalizing agent communication". In Chapter 3, we review and evaluate some proposals advocating social approach to define a formal semantics for ACL messages in terms of commitments, and specify and verify commitment-based protocols.

2.2 Computational Logics

Software systems particulary those including intelligent components such as autonomous agents will inevitably grow in scale and complexity. Because of this increase, the likelihood of introducing errors is considerable. Moreover, some of these errors may cause a system failure, which leads to catastrophic situations in terms of both human lives loss and economic damages (think of failing safety property in control systems of nuclear power plants or aircraft control systems). Despite this complexity, the major goal of software engineering is to enable developers to build systems that operate reliably. One way of satisfying this goal is by making use of *formal methods*, which are mathematically based languages, techniques, and tools for representing, specifying and verifying those systems [61]. Also, formal methods can be used to reason about the interactive behaviors of ongoing autonomous agents and cater their key functionality. Reasoning about agents' behaviors means constructing arguments about them so that the arguments are valid and can be defended rigorously, or invalid and need to be avoided [78]. However, as pointed out in [61, 78] formal methods do not a priori guarantee that the systems will be error-free, but they can greatly increase our realizing and understanding of those systems by means of revealing inconsistencies and incompleteness in those systems, which might otherwise go undetected. To this end, such formal methods are best described by means of computational logics, which are specifically designed to suit different scenarios.

In the last two decades, computational logics have gained popularity and many contributions have been made in the field of agent communication from theoretical and practical perspectives. They are particularly used to cater rigorous, unambiguous and well-understood frameworks for representing and reasoning about agents' mental states such as beliefs, desires, and intentions, which are represented by modal operators (as we discussed in Section 2.1), and agents' social states such as protocols, social commitments (as we will discuss in Chapter 3) in the context of MASs. When an appropriate logic is chosen, it can provide a level of abstraction close to the key concepts of the software to be developed. Furthermore, computational logics play a fundamental role in debugging specifications, validating system implementations and developing powerful tools such as model checkers to verify the behaviors of interacting agents against given protocol specifications [11].

Our presentation is mainly focused on two aspects, which are incorporated with any logic: (1) the *well-formed formulae* (wff) of the logic, which are the statements that can be made in it; and (2) the *model-theory* that gives the formal meaning of the wff. The model-theory is usually called semantics.

The remainder of this section is organized as follows. In Section 2.2.1, we present propositional logic which forms the main basis for the next coming logics. In Section 2.2.2, we study predicate logic, also called first-order logic, a much more expressive language than propositional logic. In Section 2.2.3, we proceed to present modal logic that extends propositional logic with new operators, usually called *possibility* and *necessity*, in front of formulae. In Section 2.2.4, we present two versions of temporal logic: linear-time temporal logic, and branching-time temporal logic, which extend modal logic by temporal modalities.

2.2.1 Propositional Logic

Propositional logic (PL for short) is a very simple form of logic to construct arguments about situations we encounter. It is sometimes called *two-valued* logic [77]. Conventionally, when developing logical languages, we are not concerned with what the declarative sentences truly mean, but only with their logical structure.

Syntax of PL. Let $\mathcal{PV}$=$\{p, q, \ldots\}$ be a nonempty countable set of atomic propositions (also denoted with the term *propositional variables*). In fact, the synonyms of proposition are *statement* and *assertion* and such words are used to refer to what is stated or asserted about the real world, the physical world we live in, or more abstract ones such as computer models, not to the act of stating or asserting [77]. Every proposition, in principle, is either true or false, and no proposition is both true and false. Truth and falsity are always said to be the *truth-values* of propositions. The set $\{\varphi, \psi, \ldots\}$ of wff[5] of propositional logic over $\mathcal{PV}$ is inductively defined by the following three formulation rules: (1) any atomic proposition $p \in \mathcal{PV}$ is a wff; (2) if φ is a wff, so is $\neg\varphi$; and (3) if φ and ψ are wff, so is $\varphi \vee \psi$.

Any formula stands for a proposition that might hold or not, depending on which of the atomic propositions are assumed to hold. Intuitively, the formula p stands for the propositions stating that p holds. The intuitive meaning of the symbol $\vee$ is disjunction ("or"), i.e., $\varphi \vee \psi$ holds if and only if one of propositions φ or ψ holds. The symbol "$\neg$" denotes negation, i.e., $\neg\varphi$ holds if and only if φ does not hold. The remaining operators can be introduced via abbreviations in terms of the above as usual. In particular, $\varphi \wedge \psi \triangleq \neg(\neg\varphi \vee \neg\psi)$ ("conjunction"), $\top \triangleq p \vee \neg p$ ("propositional constant true"), $\bot \triangleq \neg\top$ ("propositional constant false"), $\varphi \supset \psi \triangleq \neg\varphi \vee \psi$("material implication" or implication for short), and $\varphi \equiv \psi \triangleq (\varphi \supset \psi) \wedge (\psi \supset \varphi)$ ("equivalence"). The above inductive definition of propositional formulae over the set $\mathcal{PV}$ can be rewritten as:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi$$

[5]We are concerned only with well-formed formulae, else are not.

where $p \in \mathcal{PV}$. The above can be understood as a casual notation for the Backus-Naur form of a context-free grammar over the alphabet $\Sigma = \mathcal{PV} \cup \{\neg, \vee\}$. In this short-form notation, "::=" and "|" are meta-symbols of the grammar and the symbol φ in the left hand side serves simultaneously for: (1) a nonterminal symbol (variable) of the grammar; and (2) its derived words over Σ^* (i.e., propositional formulae).

Semantics of PL. To formalize the intuitive meaning of propositional formulae, we first need a precise definition of the context that declares which atomic propositions hold and which do not hold. This is done by means of an evaluation, which assigns a truth value 0 ("false") or 1 ("true") to each atomic proposition. Formally, an evaluation for $\mathcal{PV}$ is a function $\mu : \mathcal{PV} \to \{0,1\}$, also called *truth function.* The semantics of PL is specified by a satisfaction relation "$\models$" indicating the evaluations μ for which a formula φ is true; so it is usually called truth value semantics. Intuitively, $\mu \models \varphi$ stands for the fact that φ is true under evaluation μ. The satisfaction relation $\models$ is inductively defined as follows:

- $\mu \models p$ *iff* $\mu(p) = 1$,
- $\mu \models \neg\varphi$ *iff* $\mu \not\models \varphi$,
- $\mu \models \varphi \vee \psi$ *iff* $\mu \models \varphi$ *or* $\mu \models \psi$.

Satisfiability and validity. Propositional formula φ is called *satisfiable* if there is an evaluation μ with $\mu \models \varphi$. A formula φ is *valid* (or it is often called a *tautology*) if $\mu \models \varphi$ for each evaluation μ. Hughes and Cresswell [77] have used *truth-table* method and *Reductio* method (aka proof by contradiction) to expeditiously test for validity from semantical perspective, which concerns with the relationship between formulae and what they stand for. φ is *unsatisfiable* if φ is not satisfiable. For example, $p \wedge \neg p$ is unsatisfiable, while $p \vee \neg(p \wedge q)$ is a tautology. The formulae $p \vee \neg q$ and $p \wedge \neg q$ are satisfiable, but not tautologies. Thus, φ is satisfiable if and only if $\neg\varphi$ is not tautology.

2.2.2 Predicate Logic

Predicate logic (which is also called first-order logic) takes the foundation of propositional logic and builds a more expressive logic on that foundation, borrowing representational ideas from natural languages while avoiding their drawbacks, such as ambiguity. What can propositional logic do with modifiers like *there exists* and *for all*? Let us consider the following sentence:

Example 1 *[78] Every son of my father is my brother.*

The only way to represent this assertion in propositional logic is by using an atomic proposition p, for example. However, this brute way does not reflect the finer logical structure of that sentence. Predicate logic addresses propositional logic's limitation by introducing quantifiers ($\forall$ reading "for all" and $\exists$ reading "there exists"), and using variables, constants, functions, and predicates in lieu of atomic propositions. Predicate logic is expressive enough to be advocated in a number of useful programming languages, such as Prolog.

Syntax of predicate logic. Following [78], predicate logic formulae are made up of two sorts: *terms* and *formulae*. The terms consist of predicate symbols or variables $\mathbb{P}$, and function symbols $\mathbb{F}$ applied to those. The part of formulae denotes truth values and expressions comprised of logical connectives. Now the set of well-formed terms is inductively defined by the following rules: (1) any variable in $\mathbb{P}$ is a well-formed term; and (2) if $t_1, t_2, \dots, t_n$ are well-formed terms and $f \in \mathbb{F}$ has n arguments, so $f(t_1, t_2, \dots, t_n)$ is a well-formed term. In BNF grammar, we may write

$$t ::= x \mid f(t, \dots, t)$$

where x is a variable in $\mathbb{P}$, and $f \in \mathbb{F}$ has n arguments. Recall that when a function symbol does not take any arguments, it maybe used to denote *constant symbol*.

Given well-formed terms, we can now proceed to inductively define the set of formulae of predicate logic over the two sets $\mathbb{P}$ and $\mathbb{F}$ as follows: (1) if P is a predicate taking n arguments where $n \geq 1$ and if $t_1, t_2, \dots, t_n$ are well-formed terms, so $P(t_1, t_2, \dots, t_n)$ is a formula; (2) if φ is a formula, then so is $\neg\varphi$; (3) if φ and ψ are formulae, then so is $\varphi \vee \psi$; and (4) if φ is a formula and x is a variable in $\mathbb{P}$, then $\forall x \varphi$ is a formula. We can condense this definition using a BNF grammar:

$$\varphi ::= P(t_1, \dots, t_n) \mid \neg\varphi \mid \varphi \vee \varphi \mid \forall x \varphi$$

Notice that the arguments given to predicates are always terms. Other operators can be introduced in a usual way (as in propositional logic). In particular, $\exists x \varphi \triangleq \neg\forall x \neg\varphi$. The declarative sentence in Example 1 can be translated into a predicate logic formula as follows: we choose a constant m for 'me', and we choose further the set $\{S, F, B\}$ of predicates with the following meanings:

$S(x, y)$: x is a son of y, $F(x, y)$: x is the father of y, and $B(x, y)$: x is a brother of y.

Thus, the symbolic encoding of this sentence is

$$\forall x \forall y (F(x, m) \wedge S(y, x) \supset B(y, m))$$

which means that for all x and all y, if x is a father of m and if y is a son of x, then y is a brother of m. Given a variable x, a term t and a formula φ, we define $\varphi[t/x]$ to be a formula obtained by substituting each *free* occurrence of variable x in φ with t wherein x is free in φ means that its value has to be specified by some additional information.

Semantics of predicate logic. To compute the truth value of formulae in predicate logic that involve only function and predicate symbols, a model M of the pair $(\mathbb{F}, \mathbb{P})$ consists of the following set of data [78]: (1) a nonempty set A, including the universe of concrete values; (2) for each $f \in \mathbb{F}$ with n arguments, a concrete function: $f_M : A^n \to A$ from A^n (i.e., the set of n-tuples over A) to A; and (3) for each $P \in \mathbb{P}$ with n arguments, a subset $P_M \subseteq A^n$ of n-tuples over A.

To interpret formulae of the form $\forall x \varphi$ or $\exists x \varphi$, we need to check whether φ holds for all (resp. some) value a in our model. While this is intuitive, we don't have a way of expressing this notion in the syntax of predicate logic. Therefore, we are forced to interpret formulae relative to an environment $e : var \to A$, which is a function (or a look-up table) from the set of variables *var* to the universe of values A of the underlying model. Given a model M for a pair $(\mathbb{F}, \mathbb{P})$ and an environment e, we inductively define the satisfaction relation, denoted by $M \models_e \varphi$ meaning that φ computes to 1 ("true") in the model M with respect to the environment e, as follows [78]:

– $M \models_e P(t_1, t_2, \ldots, t_n)$ *holds iff by replacing all variables in the terms* $(t_1, t_2, \ldots, t_n)$ *of the set* A *with their concrete values and by computing any function symbol by means of* f_M *according to* e*, the resulting tuple of values* $(a_1, a_2, \ldots, a_n)$ *is in the set* P_M*.*

– $M \models_e \neg\varphi$ *holds iff it is not the case that* $M \models_e \varphi$ *holds,*

– $M \models_e \varphi \vee \psi$ *holds iff* $M \models_e \varphi$ *or* $M \models_e \psi$ *holds,*

– $M \models_e \forall x \varphi$ *holds iff* $M \models_{e[x \to a]} \varphi$ *holds for all* $a \in A$ *where the look up table* $e[x \to a]$ *maps* x *to* a*.*

We conclude our introduction to predicate logic with the negative result.

Satisfiability and validity. Predicate logic formula φ[6] is called satisfiable if there is some model M with $M \models \varphi$. φ is called valid if $M \models \varphi$ for each model M, so we can write $\models \varphi$. However, the decision problem, does $\models \varphi$ hold?, tends to be undecidable [56] as we cannot write a procedure that works for all φ.

[6]When φ has no free variables at all, so $M \models_e \varphi$ holds or does not hold, regardless of the choice of an environment e. In this case, [78] writes $M \models \varphi$.

2.2.3 Modal Logic

Formulae of propositional and predicate logics are either true, or false, in any model. That is, they do not allow for any further possibilities. From many points of view, however, this is inadequate. For example, as in natural language, we often distinguish between various *modes* of truth, such as *necessarily true* and *true in the future.* Recall that modal logic was originally developed by philosophers to study different modes of truth. Specifically, modal logic extends the language of propositional logic by adding to it some new monadic operators, which are interpreted in a non-truth-functional way. In what follows, we only study a normal modal logic.

Syntax of normal modal logic. The language of normal modal logic is defined by the following formulation rules: (1) if $p \in \mathcal{PV}$, then p is a formula; (2) if φ, ψ are formulae, then so are $\neg\varphi$, $\varphi \vee \psi$; and (3) if φ is a formula, then so is $\Box\varphi$. We can rewrite this definition using a BNF grammar as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \Box\varphi$$

The remaining connectives can be defined as abbreviations in the usual way as we shown in Section 2.2.1 for propositional logic. In particular, $\Diamond\varphi \triangleq \neg\Box\neg\varphi$. The formula $\Box\varphi$ is read 'necessarily φ', and the formula $\Diamond\varphi$ is read 'possibly φ'. Necessity is called a modal notion, presumably because being necessarily true has been thought of as a mode (or manner) in which a proposition can be true [77]. Note that $\Box$ and $\Diamond$ are dual modalities, and any one of them can be defined from the other like the quantifiers $\forall$ and $\exists$ in the predicate logic, but they do not take variables as arguments.

Semantics of normal modal logic. Formulae of propositional logic are interpreted by assigning a truth value to each of the atomic formulae present in these formulae by means of an evaluation function. In contrast, the interpretation of modal formulae requires labeled state-transition graphs, known as *Kripke models*, which enable us to distinguish between different modes of truth. Given a set $\mathcal{PV}$, a kripke model for normal modal logic is an ordered triple $M = (W, R, V)$, where W is a nonempty set of *possible worlds*, $R \subseteq W \times W$ is a binary relation (or an *accessibility relation*), and $V : W \to 2^{\mathcal{PV}}$ is an evaluation function, which says for each world $w \in W$ which atomic propositions are true in w. The semantics of the language is given via the satisfaction relation, denoted by $M, w \models \varphi$, which means a modal formula φ is true (or holds, or it is satisfied) at a world w in a model M. The relation $\models$ is inductively defined as follows:

– $M, w \models p$ *iff* $p \in V(w)$,

– $M, w \models \neg\varphi$ *iff* $M, w \not\models \varphi$,

– $M, w \models \varphi \vee \psi$ *iff* $M, w \models \varphi$ *or* $M, w \models \psi$,

– $M, w \models \Box\varphi$ *iff* *for all* $w' \in W, (w, w') \in R$ *implies* $M, w' \models \varphi$.

Intuitively, $\Box\varphi$ is true at a world w in a model M iff φ is true at all the worlds w' that are accessible via R from w. It is obvious that the truth-value of φ itself is not always sufficient to determine the truth-value of $\Box\varphi$. Henceforth, we cannot define $\Box$ in terms of any combination of the propositional logic operators. Notice that a formula φ is true in a model M, denoted by $M \models \varphi$, iff $M, w \models \varphi$ for all $w \in W$.

The syntax specifies exactly the wff of normal modal logic, given a set of atomic propositions. It is sometimes useful to talk about a set of formulae which have the same syntactic form; this is called *schema*. For example, $\varphi \supset \Box\Diamond\varphi$ is a schema. Any formula which has the syntactic of a certain schema is called *instance* of the schema. For example, $p \supset \Box\Diamond p$, and $(p \wedge \Diamond q) \supset \Box\Diamond(p \wedge \Diamond q)$ are instances of the schema $\varphi \supset \Box\Diamond\varphi$. We can say that a model satisfies a schema if it satisfies all of its instances.

Validity in normal modal logic. A formula φ of normal modal logic is said to be valid if it is true in every world of every model; this is indicated by writing $\models \varphi$. Any propositional tautology is a valid formula and so is any substitution instance of it. For example, since $p \vee \neg p$ is a tautology, we may perform the substitution: p is defined as $\Box p \wedge (q \supset p)$ and obtain the valid formula $(\Box p \wedge (q \supset p)) \vee \neg(\Box p \wedge (q \supset p))$.

Generally, to build a *modal logic* L, choose the schemas which you would like to have inside it. These schemas are called *axioms* (or concrete valid formulae) of the logic. Then 'close' it under the following conditions [78]:

1. L is closed under all tautologies of propositional logic.
2. L contains all instances of the schema K: $\Box(\varphi \supset \psi) \supset (\Box\varphi \supset \Box\psi)$
3. L is closed under *Modus Ponens*: if $A, A \supset B \in L$, then $B \in L$.
4. L is closed under the rule of *necessitation*: if φ is a theorem of L (this is usually denoted by $L \vdash \varphi$ and read as "φ is derivable from the axioms of L"), then $L \vdash \Box\varphi$.

Since the weakest modal logic must contain all propositional tautologies and all instances of the schema K, together with other formulae which come from applying the above conditions 3 and 4, this logic is conventionally called normal modal logic $\mathcal{K}$. By convention, the set of axiom schemas, together with the set of rules are denoted with the name of *Hilbert-style inference system* (or with the name of *axiomatic system*). In this system, the theorems are those wff which can be derived from the axioms using appropriate inference rules of the logic L. A proof of a theorem φ in an axiomatic system, say S, consists of a finite sequence of wff, each of which is either (1) an axiom of S or (2) a wff derived from one or more wff occurring earlier in the sequence, by one of the inference rules, φ itself being the last wff in the sequence.

Correspondence theory of normal modal logic. The axioms of any modal logic follow from the properties of the accessibility relation R. Indeed, to every axiom (or schema, or simply formula) there corresponds a property of R, this mathematical relationship is important as it helps understand the logic being studied and makes sense for the application, too. The properties of binary relation R may be:

– *Reflexive*: *if* $\forall\ w \in W,$ *we have* $(w, w) \in R$.

– *Symmetric*: *if* $\forall\ w, w' \in W,$ *we have* $(w, w') \in R$ *implies* $(w', w) \in R$.

– *Transitive*: *if* $\forall\ w, w', w'' \in W,$ *we have* $(w, w') \in R$ *and* $(w', w'') \in R$ *imply* $(w, w'') \in R$.

– *Euclidean*: *if* $\forall\ w, w', w'' \in W$ *with* $(w, w') \in R$ *and* $(w, w'') \in R,$ *we have* $(w', w'') \in R$.

– *Serial*: *if* $\forall\ w \in W,$ *there is a* $w' \in W$ *such that* $(w, w') \in R$.

A relation R is an *equivalence* if it is reflexive, symmetric, and transitive. To state the relationship between formulae and their corresponding properties, we need the notion of a modal frame. A *frame* F is an ordered pair $F = (W, R)$ where W is a set of worlds and $R \subseteq W \times W$ is a binary relation. Simply put, a frame is just a set of worlds and the relationship among them with no information about what atomic propositions are true at the various worlds. From any model, we can extract a frame, by just forgetting about the evaluation function. A frame is reflexive (resp. symmetric, transitive, serial) if its accessibility relation is reflexive (resp. symmetric, transitive, serial). However, it is useful to say sometimes that the frame, as a whole, satisfies a formula. A normal modal logic formula φ is satisfied in a frame F if, for each evaluation function and each $w \in W$, we have $M, w \models \varphi$. Indeed, if a frame satisfies an instance of a schema, then it satisfies the whole schema. This contrasts markedly with models. This claim is in fact true because frames do not contain any information about truth or falsity of atomic propositions and cannot distinguish between different atoms. To summarize, the following statements are equivalent: (1) if R has the property, such as reflexive, transitive, and so forth, then the frame satisfies the schema; (2) if the frame satisfies the schema, then it satisfies the instance of it; and (3) if the frame satisfies the formula, then R has the property. The proof is presented in Theorem 5.14 [78].

With respect to definition of frame F, a model can be seen as a pair $M = (F, V)$ where V is an evaluation function as we defined above; in this case, the model M is said to be based on F. A formula is valid in a frame F, denoted by $F \models \varphi$, if $M \models \varphi$ for every model $M = (W, R, V)$ based on $F = (W, R)$ and every $w \in W$, $M, w \models \varphi$. Typically, to define a validity in a more general way, we specify a certain class C of frames (e.g., the class of all frames in which the accessibility relation is serial). A formula φ is valid with respect to C, denoted by $C \models \varphi$, if $F \models \varphi$ for all frames $F \in C$. Hughes and Cresswell [77] have extended *Reductio* method of propositional logic to test for validity of normal modal logic. The idea is to attempt to find, for a given wff, a falsifying model based on a frame in a class of frames. Some well-known correspondence results are presented as follows:

Table 1: Some correspondence theory and frame classes.

Logic Name	**Schema (axiom)**	**Property of R**	**Class of Frames**
T	$\Box\varphi \supset \varphi$	Reflexive	All reflexive frames
B	$\varphi \supset \Box\Diamond\varphi$	Symmetric	All symmetric frames
D	$\Box\varphi \supset \Diamond\varphi$	Serial	All serial frames
4	$\Box\varphi \supset \Box\Box\varphi$	Transitive	All transitive frames
5	$\Diamond\varphi \supset \Box\Diamond\varphi$	Euclidean	All Euclidean frames

2.2.4 Temporal Logics

Temporal logic (TL, for short) has been investigated and analyzed as a branch of logic and can be seen as a special case of modal logic for several decades. TL was developed as a framework to reason about temporal relations, *qualitative* aspects of time and to deal not only with the input/output behavior of a program, but also with its entire execution sequence [109]. However, the semantics of modalities, such as $\Box p$ and $\Diamond g$ is not limited to the evaluation of a single-step accessibility relation, but it is provided to qualitatively describe and reason about how the truth values of assertions (i.e., p and q) vary with time [56]. Furthermore, TL is a useful formalism for specifying and verifying correctness of computer programs [108, 109]. Such a formalism is particulary appropriate for reasoning about nonterminating or continuously operating concurrent programs [31] such as hardware circuits, microprocessors, operating systems, banking networks, communication protocols, automotive electronics, and more generally reactive systems.

While the term *temporal* suggests a relationship with the real-time behavior of program, this is only true in an abstract sense in the term of the occurrence of events in the past and future without referring to the precise timing of events. One can say that the modalities in TL are *time-abstract* [4], which makes the formulation of specifications more natural and convenient. In most temporal logics, modeling time is *discrete*. From the first introduction of TL formalism, there arose an essential question which later almost developed into a controversy. More precisely, the question regards to the nature of the underlying structure of time on which the formalism is based. The controversy is between the *linear time approach*, which considers time to be a linear sequence, and the *branching time approach*, which advocates a tree-like structure time, allowing some instants to have more than a single successor. The choice among linear and branching models should be prescribed by the type of programs and properties, which one wishes to study.

A) LTL

Linear Temporal Logic (LTL, for short) was first proposed by Pnueli [109] as a formalism to specify properties of concurrent programs and to reason about their behavior. Since then it has been widely used for these purposes. In LTL perspective, each moment in time has a unique possible future. Temporal modalities are thus provided for describing events along a single time line. The basic ingredients of LTL formulae are built up from atomic propositions, Boolean connectives ($\wedge, \vee, \neg$, and so forth) and two basic temporal modalities $\Diamond\varphi$ ("sometime" φ; also read as "eventually" φ), $\Box\varphi$ ("always" φ; also read as "globally" φ), $\bigcirc\varphi$ ("next time" φ), and $\varphi\ U\ \psi$ (φ "until" ψ). Such temporal modalities are also called LTL path temporal modalities. The unary operators bind stronger than binary ones. Operators $\neg, \bigcirc, \Box$, and $\Diamond$ bind equally strong. As an example, the formula $\neg\varphi\ U\ \psi$ is interpreted as $(\neg\varphi)\ U\ \psi$. Temporal binary operators have stronger precedence over $\wedge, \vee$, and $\supset$.

Syntax of LTL. The set of formulae of LTL is generated by the following syntactic rules: (1) each atomic proposition $p \in \mathcal{PV}$ is a wff; (2) if φ and ψ are wff, then $\neg\varphi$ and $\varphi \vee \psi$ are wff; and (3) if φ and ψ are wff, then $\bigcirc\varphi$ and $\varphi\ U\ \psi$ are wff. The other formulae can then be introduced as abbreviations in the usual way. In particular, the propositional connectives ($\wedge$, $\supset$) and logical constants false and true ($\top$, $\bot$) are abbreviated as in propositional logic (cf. Section 2.2.1). The temporal operator $\Diamond\varphi$ abbreviates $(\top\ U\ \varphi)$ and $\Box\varphi$ abbreviates $\neg\Diamond\neg\varphi$. Intuitively, $\Diamond\varphi$ ensures that φ will be true eventually in the future and $\Box\varphi$ is satisfied iff it is not the case that eventually φ holds. Given a set $\mathcal{PV}$ of atomic propositions, we can condense this definition using a BNF grammar as follows [108]:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid \varphi\ U\ \varphi$$

In this definition, $p \in \mathcal{PV}$ is an atomic proposition; the formula $\bigcirc\varphi$ holds at the current moment, if φ holds in the next moment. The formula $\varphi\ U\ \psi$ holds at the current moment, if there is some future moment for which ψ holds and φ holds at all moments until that future moment.

Semantics of LTL. The semantics of LTL-formulae is given in terms of *transition systems*. A transition system $T = (S, R_t, V, I)$ is a tuple in which S is a nonempty set of states, $R_t \subseteq S \times S$

is a transition relation, $V : S \to 2^{\mathcal{PV}}$ is an evaluation function, and $I \subseteq S$ is a set of initial states. The transition relation R_t models temporal transitions among states: given two states $s, s' \in S$, $(s, s') \in R_t$ means that s' is an immediate successor of s. Also, it is assumed that every state has a successor state, i.e., the transition relation is serial (or total). Because most concurrent systems are designed not to halt during normal execution, one can model a computation *path* π in T as an infinite sequence $\pi = (s_0, s_1, \ldots)$ of states such that $(s_i, s_{i+1}) \in R_t$ for all $i \geq 0$. $\pi(i)$ is the i-th state in the path π. π_i is the suffix of π starting at $\pi(i)$, i.e., $\pi_i = \pi(i), \pi(i+1), \ldots$. Satisfaction of an LTL-formula φ with respect to a path π in a transition system T, denoted by $T, \pi \models \varphi$, is inductively defined as follows:

– $T, \pi \models p$ *iff* $p \in V(\pi(0))$,

– $T, \pi \models \neg\varphi$ *iff* $T, \pi \not\models \varphi$,

– $T, \pi \models \varphi \vee \psi$ *iff* $T, \pi \models \varphi$ *or* $\pi \models \psi$,

– $T, \pi \models \bigcirc\varphi$ *iff* $T, \pi_1 \models \varphi$,

– $T, \pi \models (\varphi\ U\ \psi)$ *iff there exists* $k \geq 0$ *such that* $T, \pi_k \models \psi$ *and* $T, \pi_i \models \varphi\ \forall 0 \leq i < k$.

For the propositions, Boolean connectives and temporal modalities, the relation $\models$ is defined in the standard manner. According to this satisfaction relation, an LTL-formula φ holds at s iff all paths starting at s satisfy φ, it is written as $s \models \varphi$. The transition system T satisfies φ when φ holds in all paths starting from an initial state.

Satisfiability and validity of LTL. An LTL formula φ is satisfiable iff there exists a transition system $T = (S, R_t, V, I)$ and a state $s \in S$ such that $s \models \varphi$. In that case, such a transition system defines a model of φ. We say that φ is valid and write $\models \varphi$ for all transition systems $T = (S, R_t, V, I)$ and for all $s \in S$, we have $s \models \varphi$. As in modal logic, we can study the satisfiability and validity of LTL formula in a frame $F = (S, R_t, I)$ of a transition system $T = (S, R_t, V, I)$.

B) CTL

In the early of 1980s, another strand of temporal logics called *computation tree logic* (CTL, for short) for specification and verification purposes was introduced in [31] and studied in [57]. The underlying structure of time in CTL is assumed to have a branching tree-like nature, which refers to the fact that each moment in time may split into several possible futures matching those resulting from nondeterminism in a program. In fact, this structure over which CTL formulae are interpreted can be viewed as an infinite tree in which each node in the tree has at most one predecessor; so it is known that the past of each node is linear. Also, there exists a unique node, called the *root*, from which all other nodes are reachable and that has no predecessors.

Syntax of CTL. CTL allows basic temporal operators of the form: a path quantifier—either A ("for all paths") or E ("for some path")—immediately followed by a single one of the usual linear temporal operators $\Box$ ("always"), $\Diamond$ ("sometime"), $\bigcirc$ ("next time") or U ("until") to obtain a well-formed state formula. The set of CTL formulae over the underlying set $\mathcal{PV}$ of atomic propositions is inductively given by the following syntactic rules: (1) every atomic proposition $p \in \mathcal{PV}$ is a formula; (2) if φ and ψ are formulae, then so are $\neg\varphi$ and $\varphi \vee \psi$; and (3) if φ and ψ are formulae, then so are $E \bigcirc \varphi$, $E\Box\varphi$, and $E(\varphi\ U\ \psi)$. The symbols $\neg$ and $\vee$ have their usual meanings. $E \bigcirc \varphi$ is read "there exists a path such that at the next state φ holds", $E\Box\varphi$ is read "there exists a path such that φ holds globally along the path", and $E(\varphi\ U\ \psi)$ is read "there exists a path such that φ holds until ψ holds". Notice that CTL operators are composed of a pair of symbols: the first symbol is a quantifier over paths (E), while the second symbol expresses some constraint over paths. We can rewrite this definition using a BNF grammar as follows [57]:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid E \bigcirc \varphi \mid E\Box\varphi \mid E(\varphi\ U\ \varphi)$$

The syntax of CTL traditionally includes the following temporal operators as well: $A \bigcirc \varphi$, $A\Box\varphi$, $A(\varphi\ U\ \psi)$, $E\Diamond\varphi$ and $A\Diamond\varphi$. These are read, respectively: "for all paths, in the next state φ holds", "for all paths, φ holds globally", "for all paths, φ holds until ψ holds", "there exists a path such that φ holds at some future point" and "for all paths, φ holds at some point in the future". These additional CTL temporal operators can be used to ease the specification process of various requirements but they are in fact definable in terms of the minimal set of CTL temporal operators $E\bigcirc$, $E\Box$, and EU.

Semantics of CTL. As in LTL, the semantics of CTL-formulae is given in terms of *transition systems*. A transition system $T = (S, R_t, V, I)$ is a tuple in which S is a nonempty set of states, $R_t \subseteq S \times S$ is a transition relation that must be serial, $V : S \to 2^{\mathcal{PV}}$ is an evaluation function, and $I \subseteq S$ is a set of initial states. As in LTL, a *path* π in T is an infinite sequence $\pi = (s_0, s_1, \ldots)$ of states such that $(s_i, s_{i+1}) \in R_t$ for all $i \geq 0$. $\pi(i)$ is the i-th state in the path π. We use the standard notation to indicate truth in a transition system: $T, s \models \varphi$ means that formula φ holds at state s in a transition system T. The relation $\models$ is defined inductively as follows:

- $T, s \models p$ *iff* $p \in V(s)$,
- $T, s \models \neg\varphi$ *iff* $T, s \not\models \varphi$,
- $T, s \models \varphi \vee \psi$ *iff* $T, s \models \varphi$ *or* $T, s \models \psi$,

– $T, s \models E \bigcirc \varphi$ *iff there exists a path* π *such that* $\pi(0) = s$ *and* $T, \pi(1) \models \varphi$,

– $T, s \models E\Box\varphi$ *iff there exists a path* π *such that* $\pi(0) = s$ *and* $T, \pi(i) \models \varphi \ \forall i \geq 0$,

– $T, s \models E(\varphi\ U\ \psi)$ *iff there exists a path* π *and for some* $k \geq 0$ *such that* $\pi(0) = s$ *we have* $T, \pi(k) \models \psi$ *and* $T, \pi(i) \models \varphi \ \forall 0 \leq i < k$.

The interpretations of atomic propositions, Boolean operators and temporal modalities are as usual (cf. for example [56]). Given the semantics above, the following equivalences hold for CTL [4]:

$$E\Diamond\varphi \equiv E(\top\ U\varphi),$$

$$A \bigcirc \varphi \equiv \neg E \bigcirc \neg\varphi,$$

$$A\Box\varphi \equiv \neg E\Diamond\neg\varphi,$$

$$A(\varphi\ U\psi) \equiv \neg E(\neg\psi\ U\ (\neg\varphi \wedge \neg\psi)) \wedge \neg E\Box\neg\psi,$$

$$A\Diamond\varphi \equiv A(\top\ U\ \varphi) \equiv \neg E\Box\neg\varphi.$$

C) CTL*

CTL*—called *full computation tree logic*—extends CTL by allowing basic temporal operators in which the path quantifier (E or A) can prefix an assertion composed of unrestricted combinations (i.e., involving arbitrary nestings and Boolean connectives) of the linear time operators $\Diamond$, $\Box$, $\bigcirc$ and U. It was proposed as a unifying framework in [58], subsuming CTL and LTL. Recall that the underlying nature of time in CTL* is branching: each moment in time may split into alternative courses representing different possible futures, which is suitable for reasoning about nondeterministic and concurrent programs.

In CTL*, one can distinguish between two types of formulae: state formulae and path formulae. The state formulae—true or false of states—are assertions about the atomic propositions in the states and their branching structure, while path formulae—true or false of paths—express temporal properties of paths. Importantly, path formulae can be turned into state formulae by prefixing them with the path quantifiers. Thus, in the branching time interpretation, a formula is true or false in a state whereas in the linear time interpretation, a formula is true or false along a path.

Syntax of CTL*. Given the set $\mathcal{PV}$ of atomic propositions, we inductively define a class of state formulae using rules S1–3 and a class of path formulae using rules P1–3 as follows [58]:

S1 Each atomic proposition $p \in \mathcal{PV}$ is a state formula.

S2 If φ and ψ are state formulae, then so are $\neg\varphi$ and $\varphi \vee \psi$.

S3 If φ is a path formula, then $E\varphi$ is a state formula.

P1 Each state formula is also a path formula.

P2 If φ and ψ are path formulae, then so are $\neg\varphi$ and $\varphi \vee \psi$.

P3 If φ and ψ are path formulae, then so are $\bigcirc\varphi$ and $\varphi\ U\ \psi$.

The set of state formulae generated by the above rules forms the language CTL*. The other Boolean connectives are introduced as above while the other temporal operators can then be introduced as abbreviations in the usual way. Also, the universal path quantifier A can be defined using the existential path quantifier and negation: $A\varphi \triangleq \neg E\neg\varphi$. Thus, we could give a substantially more terse syntax and semantics for CTL* language by defining all the other operators in terms of just the primitive operators $\bigcirc, U, \neg$, and $\vee$ and the quantifier E. This definition can be rewritten using a BNF grammar as follows:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid E\psi$$

$$\psi ::= \varphi \mid \neg\psi \mid \psi \vee \psi \mid \ \bigcirc\psi \mid \psi\ U\ \psi$$

Semantics of CTL*. A formula of CTL* is interpreted with respect to a transition system $T = (S, R_t, V, I)$ where S, R_t, V and I have the same meaning as in CTL. We write $T, s \models \varphi$ (respectively, $T, \pi \models \varphi$) to mean that a state formula φ (respectively, a path formula φ) is true in a transition system T at state s (respectively, of path π). We define the relation $\models$ inductively as follows:

- $T, s \models p$ *iff* $p \in V(s)$,
- $T, s \models \neg\varphi$ *iff* $T, s \not\models \varphi$,
- $T, s \models \varphi \vee \psi$ *iff* $T, s \models \varphi$ *or* $T, s \models \psi$,
- $T, s \models E\varphi$ *iff there exists a path* π *such that* $\pi(0) = s$ *and* $T, \pi \models \varphi$,
- $T, \pi \models p$ *iff* $T, \pi(0) \models p$,
- $T, \pi \models \neg\psi$ *iff* $T, \pi \not\models \psi$,
- $T, \pi \models \varphi \vee \psi$ *iff* $T, \pi \models \varphi$ *or* $T, \pi \models \psi$,
- $T, \pi \models \bigcirc\psi$ *iff* $T, \pi_1 \models \psi$,
- $T, \pi \models \varphi\ U\ \psi$ *iff for some* $k \geq 0$ *we have* $T, \pi_k \models \psi$ *and* $T, \pi_i \models \varphi\ \forall 0 \leq i < k$.

A state s satisfies $E\varphi$ if there exists a path π starting at s satisfies φ whilst a path satisfies a state formula if the initial state of the path so does. $\bigcirc\psi$ holds on a path π when ψ is satisfied on the path starting from the next state of the path π, and $\varphi\ U\ \psi$ holds on a path if φ holds on the path until ψ becomes true.

It is worth noticing that a transition system is a Kripke model. If some state in a transition system is designated as the initial state, then the Kripke model (which is a finite computational structure) can be unwound into an infinite tree with the initial state as the root. Since paths in the tree represent the possible computations of the program, we will refer to the infinite tree obtained in this manner as the computation tree of the program. However, the essential difference between semantics of temporal logics and traditional Kripke semantics lies in how they handle branching in the underlying computation tree. In LTL, temporal operators are provided for describing events along a single computation path, whereas in CTL, temporal operators quantify over the paths that are possible from the given state. Simply put, in Kripke semantics, formulae are interpreted directly on the finite-state model (Kripke model) of the internal architecture of the system, while the semantics of temporal logics' formulae are interpreted on the possible computation paths arising from the Kripke model. Figure 2 displays an example of a simple Kripke structure $M = (S, R_t, V, I)$

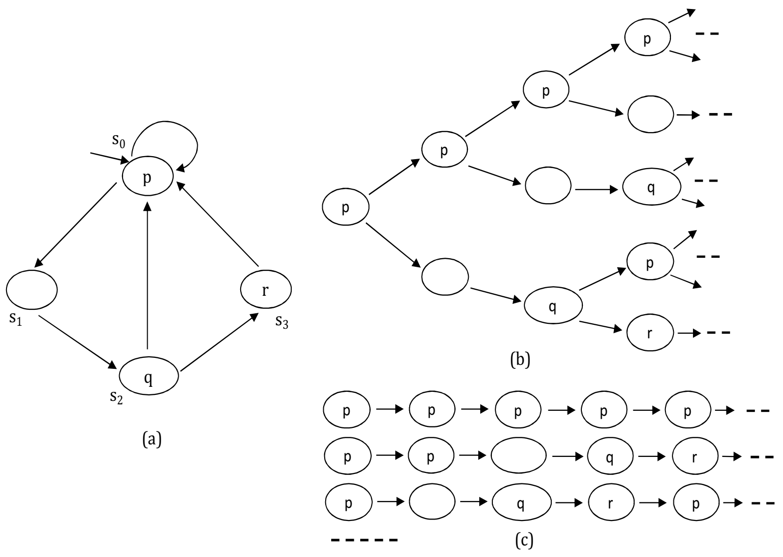


Figure 2: The model M (a) and its corresponding branching (b) and its linear (c) computations.

with its branching and linear computations. In the figure, time flows from left to right along the computations in the tree. The example uses a set $\mathcal{PV}$ =$\{p, q, r\}$ of atomic propositions. M has $S = \{s_0, s_1, s_2, s_3\}$, $I = \{s_0\}$ (indicated by the incoming arrow), and V given by $V(s_0) = \{p\}$, $V(s_1) = \emptyset$, $V(s_2) = \{q\}$, and $V(s_3) = \{r\}$. The transitions in R_t are all the directed edges between states.

Satisfiability and validity of CTL*. We say that a state formula φ (resp. a path formula φ) is valid provided that for every transition system T and every state s (resp. path π), we have $T, s \models \varphi$ (resp. $T, \pi \models \varphi$). A state formula φ (resp. a path formula φ) is satisfiable provided that for some transition system T and some state s (resp. path π), we have $T, s \models \varphi$ (resp. $T, \pi \models \varphi$).

The following results in CTL* is proven: CTL* is more expressive than both CTL and LTL [58]. It is arguable that when considering conjunctions or disjunctions of some of CTL and LTL formulae, we can construct a formula that is CTL* formula, but neither CTL nor LTL. Also, the main distinction between CTL and CTL* is the type of LTL formula that can appear in the scope of a path quantifier. In CTL, the formulae appearing in the scope of path quantifiers are restricted to be a single temporal operator. In CTL*, they can be arbitrary LTL formulae. Figure 3 graphically depicts that the

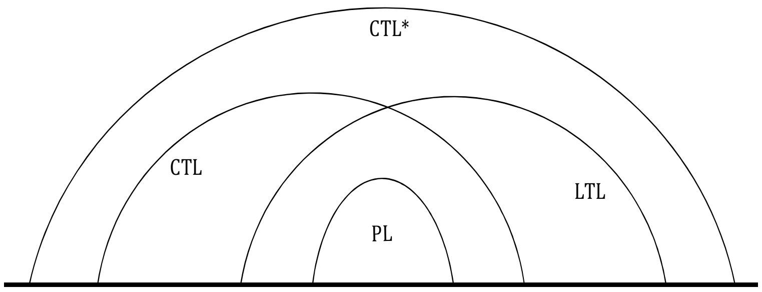


Figure 3: Expressive powers of PL, LTL, CTL, and CTL*.

aforementioned languages can be arranged in the following hierarchy of expressive power (where $<$ indicates “strictly less expressive than” and $\asymp$ indicates “as incomparable as”): PL $<$ LTL $\asymp$ CTL $<$ CTL*. Notice that the expressiveness of linear- and branching-time logics is incomparable. This means that some properties that are expressible in LTL cannot be expressed in CTL and vice versa. For example, there is no CTL formula that is equivalent to the LTL formula $A(\Diamond\Box\varphi)$. Likewise, there is no LTL formula that is equivalent to the CTL formula $A\Box(E\Diamond\varphi)$.

2.3 Model Checking

As technology allows systems to grow and be more complex, *verification*, as the process of verifying that a system complies with given specifications, has to play a fundamental role in the development process to identify and remove unwanted behaviors. In particular, verification is more important in MAS and especially agent communication, which involves autonomous interacting agents. Verification includes a number of different techniques. The most common verification technique is called

Testing. In this technique, the verification process is performed by running a number of possible test cases and then checking that the required specifications hold in all possible tested runs. While testing is sensitive to the defined set of test cases, it is generally hard and even impossible sometimes to define all the possible test cases in the specification.

This thesis will not be concerned with the problem of testing the behaviors of interacting agents; instead, the problem of formal verification for multi-agent interaction protocols will be investigated. Formal verification can be performed by theorem proving and model checking. In theorem proving approach, a proof construction, typically using various axioms and inference rules defined in an axiomatic system, is performed manually. Such a proof construction is in general difficult and requires a good deal of human ingenuity [78]. This is approach really worked for small programs [33]. However, manual proof construction is not scale up well to large programs and fails "to be of much help due to the inherent complexity of testing validity for even the simplest logics" [32]. The only extent of automation that one can hope for, is that the proof be checked by a machine. A formal verification via model checking was developed independently by Clarke and Emerson [31] and by Queille and Sifakis [110] in the early of 1980s to provide a practical alternative approach that aims at addressing difficulties of manual proof construction.

2.3.1 Model Checking Problem

In a nutshell, the technical formulation of the model checking problem is as follows: Given an abstract model $M = (S, R_t, V, I)$ representing, for example, a hardware or software design, and a specification expressed as a temporal logic formula φ, does $M \models \varphi$? for all $s \in I$ such that $M, s \models \varphi$. An alternative formulation is, given M and φ, calculating the set $\{s \in S \mid M, s \models \varphi\}$ of states that satisfy φ. Notice that:

- Model checking, unlike theorem proving, avoids proofs and requires no user expertise. In effect, it is algorithmic in nature and can be completely automated [31].
- As checking that a single model satisfies a formula is much easier than proving the validity of a formula for all models, it is possible to implement this technique very efficiently [31, 32].
- No particular requirements are usually imposed on M, even though, in most cases, M is required to be finite. The fact that the system is finite is not a limitation [32] because many concrete systems, such as communication protocols, hardware circuits and control systems can be modeled as finite-state systems, which are amenable to automatic verification. As a result, reasoning about real systems often implies reasoning about finite-state systems [56, 33].

Model checkers typically have three main components: (1) a specification language, based on temporal logic; (2) a way of encoding a state machine representing the system to be verified; and (3) a verification procedure, that uses an intelligent exhaustive search of the state space to determine if the specification holds. Model checkers provide automated methods for verification of correctness and for bug detection. They terminate with the answer true, indicating that the model satisfies the specification, or give a counterexample execution that demonstrates a behavior which falsifies the specification. This faulty trace feature provides a priceless insight to understand the real reason for

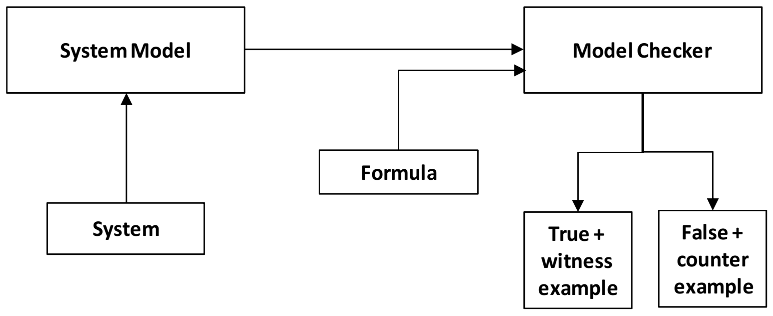


Figure 4: A model checker with counter and witness examples.

the failure as well as important clues for fixing the problem. Generally, model checkers are quite fast and sometimes produce an answer in a matter of minutes! Furthermore, partial specifications can be checked, so it is unnecessary to specify completely the system before useful information can be obtained regarding its correctness [33]. Above all, most recent model checkers produce witness examples for existential properties that are true (cf. Figure 4). As a practical matter, the separation of system development from verification and debugging has facilitated model checking's industrial acceptance. The development team can focus on various aspects of the system under design and the team of verification engineers can conduct verification independently.

2.3.2 Model Checking Techniques

Two general techniques to model checking are used in practice today. The first technique is called *temporal model checking* and developed in [31, 110]. In this technique, specifications are expressed in a temporal logic [108] in which systems as we mentioned are modeled as finite-state transition systems. The idea is to develop an efficient search algorithm to check if a given transition system is a model for the specification. In the second technique, the specification is defined as an *automaton* and also the system is modeled as an automaton. The main idea of this technique is to compare the specification and system to determine whether system behavior conforms to that of the specification. Vardi and Wolper [141] have showed how the temporal model checking problem could be recast in terms of automata; so linking these two approaches.

A) Model Checking with Automata

Definition 1 *A finite automaton over finite words is a tuple* $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$ *where* Σ *is a finite alphabet,* Q *is a finite set of states,* $Q_0 \subseteq Q$ *is a set of initial states,* $\delta \subseteq Q \times \Sigma \times Q$ *is a transition relation, and* $F \subseteq Q$ *is a set of accepting states.*

Essentially, an automaton can be viewed as a graph where the set of nodes is Q and the edges are given by δ. The automata with labeling states are commonly referred to as Kripke structures and the automata with labeled transitions are referred to as labeled transition systems. Figure 5 depicts the automaton with labeled transitions. Such an automaton is characterized by $\Sigma = \{a, b\}$, $Q = \{q_0, q_1, q_2\}$, $Q_0 = \{q_0\}$ (the initial state is marked with an incoming arrow), $F = \{q_2\}$ (the accepting state is marked with a double circle), and $\delta = \{(q_0, a, q_0), (q_0, b, q_0), (q_0, b, q_1), (q_1, a, q_2), (q_1, b, q_2)\}$.

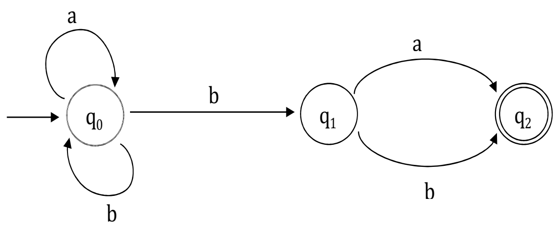


Figure 5: A finite automaton example.

Let $\sigma = a_1, \ldots, a_n \in \Sigma^*$ be a finite word (or string) of length $|\sigma| = n$. A *run* of $\mathcal{A}$ over σ is a finite sequence of states $q_0, q_1, \ldots, q_n$ such that $q_0 \in Q_0$ and $(q_i, a_i, q_{i+1}) \in \delta$ for all $0 \leq i < n$. A run $q_0, q_1, \ldots, q_n$ over σ is called *accepting* if $q_n \in F$. A finite word $\sigma \in \Sigma^*$ is called *accepted* by $\mathcal{A}$ iff there exists an accepting run of $\mathcal{A}$ over σ. The *accepted language* by $\mathcal{A}$, denoted with $L(\mathcal{A})$, is the set of finite words in Σ^* accepted by $\mathcal{A}$. Examples of runs of the automaton $\mathcal{A}$ in Figure 5 are q_0 for the empty word λ; q_0, q_1 for the word consisting of the symbol b; q_0, q_0, q_0, q_0 for, e.g., the words *aba* and *bba*; and q_0, q_1, q_2 for the words *ba*, and *bb*. The runs q_0, q_1, q_2 for *ba* and *bb* and q_0, q_0, q_1, q_2 for the words *abb*, *aba*, *bba*, and *bbb* are accepting. Thus, these words belong to $L(\mathcal{A})$. The word *aaa* is not accepted by $\mathcal{A}$ since it only has single run, namely q_0, q_0, q_0, q_0, which is not accepting.

Finite automata over infinite words accept words of infinite length from Σ^ω, where ω indicates an infinite number of repetitions. For instance, the infinite repetition of the finite word *ab* yields the infinite word *abababab*... and is denoted by $(ab)^\omega$. The simplest automata over infinite words are called *Büchi automata*. The components of Büchi automata are similar to the components of automata over finite words presented above, the only difference being the acceptance condition. Different kind of acceptance conditions can be defined, and these correspond to different kind of automata. A run ρ of a Büchi automaton over infinite word $\sigma \in \Sigma^\omega$ is defined in a similar way as a run of a finite automaton over a finite word, except that $|\sigma| = \omega$, thus ρ is infinite, i.e., it visits some state $q \in Q$ infinitely often. Let $\mathit{inf}(\rho)$ be the set of states appearing infinitely often in the run ρ. Given a set F of accepting states, a run ρ of a Büchi automaton $\mathcal{B} = (\Sigma, Q, Q_0, \delta, F)$ over an infinite word is *accepting* iff $\mathit{inf}(\rho) \cap F$ is not equal to the empty set, which means that at least one accepting state from F occurs infinitely often in ρ.

LTL Model Checking. It was first proposed in the 1980s by Vardi and Wolper [141] using the automata-theoretic approach. In this approach, the model and specification are represented in the same way. The connection between automata and temporal formula may not be apparent. Vardi and Wolper pointed out this connection by viewing LTL formulae as *language acceptors* and LTL model as *model generators*. More precisely, an infinite word σ is in the language of an LTL-formula φ iff σ satisfies φ. They also shown that every LTL-formula φ can be translated into a Büchi automaton that accepts the set of words satisfying the formula φ. As an example, the Büchi automaton corresponding to the LTL-formula $\bigcirc p$ is depicted in Figure 6.

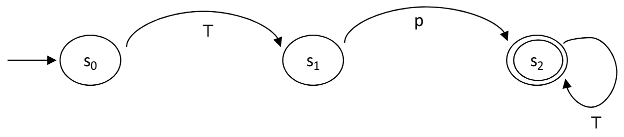


Figure 6: Büchi automaton for the formula $\bigcirc p$.

Given a model M and an LTL formula φ represented by a Büchi automaton, do all paths in M satisfy the formula φ? With automata-based approaches, this problem reduces to check if all the runs of M are accepted by the automaton. Suppose $\mathcal{B}(\varphi)$ is the automaton corresponding to φ. From [33], the first easy task is to transform a model M to a Büchi automaton $\mathcal{B}(M)$ such that $\mathcal{B}(M)$ accepts exactly the runs of M and all states of M are accepting. With this result, the problem of model checking can be defined as language containment for Büchi automata representing M and φ, i.e., does $L(\mathcal{B}(M)) \subseteq L(\mathcal{B}(\varphi))$? However, checking for language containment involves complementation of Büchi automaton of formula, which causes exponential complexity. Fortunately, Büchi automata intersection can be done with linear complexity in the size of product automaton [141]. So instead of complementing the automaton $\mathcal{B}(\varphi)$ corresponding to the formula φ, we negate the formula itself and then check if $L(\mathcal{B}(M) \cap \mathcal{B}(\neg\varphi))$ is empty. From the acceptance condition, it follows directly that an automaton is nonempty iff there is at least one accepting run that ends in some accepting state. Thus, checking for nonemptiness of an automaton is "equivalent to finding a strongly-connected component[7] that is reachable from an initial state and contains an accepting state" [33], i.e., there exists a reachable cycle in the product automaton. When the interaction is nonempty, then there is a counterexample, which is a run, generated from a periodic of finite sequence of states. Algorithm 1 describes a high-level description of model checking based on automata.

Algorithm 1 $MC(\varphi, M)$: "yes" if the automaton is empty or "no" plus a counterexample

```
1: Construct the automaton $\mathcal{B}(M)$,
2: Construct the automaton accepting the language $L(\mathcal{B}(\neg\varphi))$,
3: Construct the automaton accepting the language $L(\mathcal{B}(M) \cap \mathcal{B}(\neg\varphi))$,
4: Check emptiness of $L(\mathcal{B}(M) \cap \mathcal{B}(\neg\varphi))$.
```

The procedures in steps 2, 3, 4, and 5 are presented in [33]. Different optimization techniques can be employed to enhance the efficiency of Algorithm 1. For instance, the emptiness checking for Büchi automaton can be done *on-the-fly* (or locally) [38]. On-the-fly model checking is an optimization technique which avoids building the automaton $\mathcal{B}(M)$ in the first step of Algorithm 1. Instead, using this technique the automaton $L(\mathcal{B}(\neg\varphi))$ is constructed first, and then the automaton accepting $L(\mathcal{B}(M) \cap \mathcal{B}(\neg\varphi))$ is iteratively constructed by adding states of M to its automaton $\mathcal{B}(M)$ when needed [33]. In this way, the intersection of the two automata is constructed 'on demand', i.e., a new state is only considered if no accepting cycle has been encountered yet in the partially constructed intersection of the automata.

CTL and CTL* Model Checking. Since branching-time temporal logics are interpreted over Kripke structures, word automata are not appropriate for model checking such logics. Instead, the automata-theoretic counterpart over infinite trees are presented (see for example [86] and references therein). These automata, called tree automata, are designed to accept infinite trees as input.

Definition 2 *A tree automaton is of the form $(\Sigma, Q, Q_0, \delta, F)$ where Σ is an alphabet, Q is the set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \to 2^{Q^k}$ is a transition function, and F represents an acceptance condition wherein k represents the branching degree of the trees accepted by the automaton.*

A run of a tree automaton on a k-ary tree is a k-ary tree satisfying the following conditions:

– The root of the tree is labeled with Q_0.

[7]A strongly-connected component (SCC) is a maximal subgraph in which every node in SCC is reachable from every other node in SCC through a directed path completely contained in SCC [33].

– If node σ (where σ is a k-ary word representing a node in a k-ary tree) in a run is labeled by q and if node σ in an input tree is labeled by a then the i-th successor of σ is labeled by q_i where $(q_1, \ldots, q_k) \in \delta(q, a)$.

A run of a tree automaton is accepting iff every path in the run satisfies the acceptance condition. A different automata-theoretic approach to branching-time model checking, based on the concepts of *amorphous automata*, was first introduced by Bernholtz and Grumberg [1993]. Technically, the definition of an amorphous automaton is very similar to the above Definition 2, except that transition function is from $Q \times \Sigma \times int$ to 2^{Q^k}. For instance, $\delta(q, a, k) = \{q_1, \ldots, q_k\}$ means from state q (which has a k-ary transition on a) to the set of states $q_1, \ldots, q_k$. While this constitutes a meaningful first step towards applying automata-theoretic techniques to branching-time temporal model checking, it is not quite satisfactory because the complexity of the resulting algorithm for CTL model checking is quadratic in both the size of the specification and the size of the model, which makes this algorithm impractical.

Kupferman et al. [86] have argued that *alternating tree automata* are the key to a comprehensive and satisfactory automata-theoretic framework for branching-time temporal logics. Alternating tree automata generalize the standard notion of nondeterministic tree automata by allowing several successor states to go down along the same branch of the tree, so transitions are of the form: $\delta(q, a) = \{(q_{11}, \ldots, q_{1k}), \ldots, (q_{m1}, \ldots, q_{mk})\}$ where each k-tuple represents a possible next configuration of the automaton. An alternative way of representing this transition is by making use of Boolean expressions. In terms of Boolean expressions, the above transitions will be as follows: $((1, q_{11}) \wedge \ldots \wedge (k, q_{1k})) \vee \ldots \vee ((1, q_{m1}) \wedge \ldots \wedge (k, q_{mk}))$, for each pair of the form (i, q), i represents the direction and q represents the state in that direction.

When a tree automaton is used, the model checking problem reduces to the membership problem for the tree automaton. The membership checking problem itself can be reduced to the emptiness checking problem for a *1-letter alternating tree automaton* (a 1-letter automaton refers to an automaton whose alphabet has exactly one symbol). However, emptiness checking for a general 1-letter alternating automaton is not efficient [86]. But for special restricted versions of the automaton known as *Weak Alternating Automaton* (WAA) and *Hesitant Alternating Automaton* (HAA), the emptiness checking can be done in a linear time [86]. Thus, for CTL (resp. CTL*), this approach gives rise to a model checking technique which has time complexity linear (resp. exponential) in the length of the formula and linear (resp. linear) in the size of the model.

The automata-theoretic techniques can also be used to derive bounds on the space required for model checking CTL and CTL*. From [86], it can be shown that the emptiness checking problem for weak and hesitant alternating automata with bounded alternation is NLOGSPACE-complete. Moreover, if we fix the length of the formula, then the model checking problems for both CTL and CTL* (also called program complexity) are NLOGSPACE-complete. Further, if we consider a concurrent program, the size of the global state space is "at most exponential in the size of the individual components". Therefore, CTL and CTL* model checking problems for a concurrent program are PSPACE-complete.

State Explosion Problem. State explosion is a phenomenon often encountered when using model checking based on automata-theoretic techniques [32, 33]. Such a problem means that the number of global states in a model grows exponentially with the number of variables, or concurrent components, constituting the modeled system. In fact, the state explosion problem has been the driving force behind much of the research in the model checking approach and the development of new model checkers. Great steps have been made on this problem over the past 28 years by what is now a very large international research community.

In what follows, we discuss the *Ordered Binary Decision Diagram* (OBDD) technique, one of existing techniques proposed to combat and reduce state space explosion during model checking.

B) Symbolic Model Checking with OBDDs

Before we present OBDDs based techniques for model checking, we study briefly labeling algorithms for CTL, LTL and CTL*.

CTL model checking was introduced in [31] and [32]. Let $M = (S, R_t, V)$ be a CTL model and let φ be a CTL formula. CTL model checking algorithm, called *labeling algorithm*, takes φ and M as input and operates by labeling each state $s \in S$ with the set $label(s)$ of subformulae of φ that are true in s. The algorithm works on stages, i.e., during the $i-$th stage, subformulae with i-1 nested CTL operators are processed. When a subformula is processed, it is then added to the labeling of each state in which it is true. For example, if $\varphi \equiv p \wedge q$, then φ should be placed in the $label(s)$ precisely when p and q are already added to $label(s)$ in the previous stage. When the algorithm terminates, we will have that $M, s \models \varphi$ iff $\varphi \in label(s)$. This explicit state model checking algorithm runs in linear time in both the length of the formula and the model, that is in deterministic polynomial time [31, 33]. In fact, CTL model checking is P-complete [115]. Notice that in some cases, a CTL model includes a set of initial states. In these cases, formulae need to be only evaluated in the set of reachable states from the initial states [33].

Lichtenstein and Pnueli [89] have implicity used a tableau to develop LTL model checking algorithm. Simply put, a tableau is a graph derived from the formula from which a model for the formula can be extracted iff the formula is satisfiable. Let $M = (S, R_t, V)$ be an LTL model with $s \in S$ and let $A\varphi$ be an LTL formula such that φ is a restricted path formula, i.e., its state subformulae are only atomic propositions. The problem is to determine if $M, s \models A\varphi$. But since $M, s \models A\varphi$ iff $M, s \models \neg E \neg \varphi$, it is sufficient to be able to check the truth of formulae of the form $E\psi$ where ψ is a restricted path formula. Lichtenstein and Pnueli's algorithm consists of two steps. The first step is to construct a graph that contains a path corresponding to the path in the model M satisfying the formula φ. The second step is to identify "valid" nodes in the graph. To achieve this aim, Lichtenstein and Pnueli defined α-SCC (a maximal strongly-connected component) with the property that for every formula of the form $\varphi\ U\ \psi$ that occurs in a node in the SCC, there exists a node in the SCC that contains ψ. Then, they identified all nodes which are in an α-SCC or have a path to an α-SCC as valid nodes. Hence, the problem of model checking is reduced to the problem of finding the α-SCCs. Sistla and Clarke [132] have showed that the problem of model checking LTL is PSPACE-complete.

Clarke et al. [33] have combined CTL model checking [31, 32] with LTL model checking [89] to develop a state labeling algorithm for CTL* model checking. The idea is whether or not $M, s \models E\varphi$ wherein φ contains "arbitrary state subformulae". The algorithm starts by assuming that the state subformulae of φ is processed and then the state labels are updated. Then, each state subformula is replaced by a "fresh atomic proposition" in both the labeling of the model and formula. Now, let the new formula be denoted by $E\varphi'$. If the formula $E\varphi'$ is in CTL, then the algorithm applies the CTL model checking. Otherwise, φ' is a pure LTL path formula and the algorithm for LTL model checking is used. In case of $E\varphi$ is a complex CTL* formula, then the algorithm is repeated after replacing $E\varphi$ by a fresh atomic proposition. Such a process is repeated until the complete formula is processed. The time complexity of CTL* model checking is linear in the size of the model and exponential in the length of the formula (i.e., $|M| \cdot 2^{|\varphi|}$) [33, 115], and its space complexity is PSPACE-complete [56, 115].

It is worth noticing that for concurrent systems with small numbers of processes, the number of states is usually small and then CTL model checking algorithm [31, 32] is often quite practical. In systems with many concurrent parts, the number of states in the global state-transition system is too large to handle; so an explosion in the size of the model may occur [33].

Binary Decision Diagrams (BDDs). One solution to the state explosion problem is to find alternative representations for the state space that use any form of regularity in the state space [33]. Often practical systems do have significant regularity, so by using an appropriate method, it may be possible to represent the state space compactly. *Binary Decision Diagrams* (BDDs) [102] are one such solution.

BDDs represent the state space symbolically. BDDs are basically a canonical representation of Boolean formulae [21]. Unlike binary decision trees, which are also used to represent Boolean functions (i.e., functions from n Boolean variables $\{0,1\}^n$ to Boolean variable $\{0,1\}$), BDDs are finite, *Directed Acyclic Graphs* (DAGs) with the property that any path from the root to a leaf there is a strict ordering of Boolean variables, i.e., depending on the value assigned to the variable, we take the left or the right branch. The leaves (terminals) in the BDD, which are either 0 or 1, represent the result of evaluating a Boolean function. So, to evaluate a Boolean function for a given assignment of the variables, we traverse the graph from the root, branching at each nonterminal node depending on the value assigned to the corresponding variable, till we hit a leaf. The function value then is the value of the terminal node we reach [102, 33, 78].

A reduced BDD (RBDD) is a BDD that has undergone the following optimizations, repeatedly, until a fixed point has been reached [78]: (1) removing duplicate terminals; (2) removing redundant tests; and (3) removing duplicate non-terminals. With these three reductions, BDDs can often be quite compact representations of Boolean functions. It is also possible to perform many Boolean operations on BDDs efficiently [21]. An ordered BDD (OBDD) is one which has an ordering for some list of variables. The ROBDD representing a given Boolean function f is unique [21, 78]. To understand the uniqueness property, let B_1 and B_2 be two ROBBDs with compatible variable orderings. If B_1 and B_2 represent the same Boolean function, then they have identical structure. Finally, the operation of Boolean quantification can be applied on a BDD representing a Boolean function. Formally, given a Boolean function $f(x)$ where x is a Boolean variable, the operation $\exists x.f(x)$ is defined as $f[0/x] + f[1/x]$; that is, $\exists x.f(x)$ is true if f could be made true by putting x to 0 or 1. The definition of the Boolean quantification can be extended to a quantification over a set of variables $\overline{x} = (x_1, \ldots, x_n)$ (cf. [33, 78] for more details).

Symbolic model checking CTL. The application of OBDDs techniques to model checking for CTL has been investigated from the beginning of the 1990s by McMillan [102]. Model checking using OBDDs is called *symbolic model checking*. The term emphasizes that individual states are not represented; instead, sets of states are represented symbolically, namely, those which satisfy the formula being checked. Let $[\![\varphi]\!] = \{s \in S \mid M, s \models \varphi\}$ be a set of states satisfying φ.

Given a CTL formula φ and a CTL model $M = (S, R_t, V, I)$, the idea of model checking using OBDDs is to compute the set $[\![\varphi]\!]$ of states satisfying φ in M, which is represented in OBDDs and then comparing it against the set of initial states I in M that is also represented in OBDD. If $I \subseteq [\![\varphi]\!]$, then the model M satisfies the formula; otherwise a counterexample can be generated showing why the model does not satisfy the formula. The proof of the correctness of this approach can be found in [33, 78].

As the key idea of this technique is to represent states and set of states as Boolean formulae which, in turn, can be easily encoded as OBDDs. Following [33, 78], we show how sets of states and transition relations among states are represented with OBDDs. Let S be the set of states of a model $M = (S, R_t, V, I)$ in which the set of states of M is assumed to be finite, and let $N = \lceil log_2|S| \rceil$ be the number of Boolean variables needed to represent the elements in S. Each element $s \in S$ is associated with a unique vector of Boolean variables $\overline{v} = (v_1, \dots, v_N)$, i.e., each element of s is associated with a tuple of $\{0,1\}^N$. Each tuple $\overline{v} = (v_1, \dots, v_N)$ is then identified with a Boolean formula, represented by a conjunction of variables or their negation. It is assumed that the value 0 in a tuple corresponds to a negation. Given both a model $M = (S, R_t, V, I)$ and an encoding of the set of states S using N Boolean variables $(v_1, \dots, v_N)$, the transition given by R_t may be encoded as a Boolean function. To do this, a new set of"primed" variables $(v'_1, \dots, v'_N)$ is introduced to encode the transition among two states $s, s' \in S$. In particular, if $(s, s') \in R_t$ holds, then s is encoded using the non-primed variables, s' is encoded using the primed variables and the transition step $(s, s') \in R_t$ is expressed as a Boolean formula by taking the conjunction of the encodings of s and s'. The whole transition relation $R_t \subseteq S \times S$ is encoded as a Boolean formula by taking the disjunction of all the transition steps.

The standard symbolic model checking algorithm for CTL was introduced in [102, 33, 78]. It takes the model M and the formula φ as input and returns the set of states in M satisfying the formula φ (cf. Algorithm 2). In particular, the algorithm operates recursively on the structure of φ and builds the set $[\![\varphi]\!]$ of states using the following operations on sets: *complementation, union*, and *existential quantification*. When sets of states are encoded using Boolean formulae, all these operations on sets are translated into operations on Boolean formulae (e.g., the union of two sets corresponds to the disjunction of the Boolean formulae encoding these two sets). In the basic cases (i.e., when φ is an atomic proposition (line 1) and has Boolean connectives: negation and disjunction (lines 2 and 3), the algorithm directly returns the Boolean formulae encoding the set $[\![\varphi]\!]$ of states satisfying these cases. In lines 4 to 6, the algorithm calls the standard procedures $SMC_{E\bigcirc}(\varphi_1, M)$,

Algorithm 2 $SMC(\varphi, M)$: the set $[\![\varphi]\!]$ satisfying the CTL formula φ

```
1: $\varphi$ is an atomic formula: return $V(\varphi)$,
2: $\varphi$ is $\neg\varphi_1$: return $S \backslash SMC(\varphi_1, M)$,
3: $\varphi$ is $\varphi_1 \vee \varphi_2$: return $SMC(\varphi_1, M) \cup SMC(\varphi_2, M)$,
4: $\varphi$ is $E \bigcirc \varphi_1$: return $SMC_{E\bigcirc}(\varphi_1, M)$,
5: $\varphi$ is $E(\varphi_1\ U\ \varphi_2)$: return $SMC_{EU}(\varphi_1, \varphi_2, M)$,
6: $\varphi$ is $E\Box\varphi_1$: return $SMC_{E\Box}(\varphi_1, M)$.
```

$SMC_{EU}(\varphi_1, \varphi_2, M)$ and $SMC_{E\Box}(\varphi_1, M)$ introduced in [78] to compute Boolean formulae encoding the set $[\![\varphi]\!]$ of states satisfying the following forms of φ: $E \bigcirc \varphi_1$, $E(\varphi_1 U \varphi_2)$ and $E\Box\varphi_1$. All the computed Boolean formulae can then be easily represented using OBBDs [21]. Notice that Algorithm 2 provides a methodology to build the OBDD corresponding to the set $[\![\varphi]\!]$ in which a formula φ holds in a given model M.

2.3.3 Model Checking Tools

The work of Clarke, Emerson, and Sifakis continues to be central to the success of this research area. Their work over the years has led to the creation of new verification algorithms and tools. A huge effort, performed by both academic and industrial teams, has been, and is being, devoted to the development of model checking tools (aka model checkers), which can verify larger models and deal with a wide variety of extended frameworks. Currently, there are tremendous tools developed to serve different purposes. In this section, we briefly summarize four tools and their capabilities along with their specification languages. The tools reviewed below have been employed by peer research efforts to verify commitment-based protocols and thereby they have a relevant role with the material we shall present in the next chapter.

A) SPIN

The model checker SPIN (Simple Promela INterpreter) is one of the most used model checkers. It was originally developed in the early of 1980s at Bell Labs. SPIN has been available to the general public since 1991, and it has been continually developed since then. Recall that the theoretical foundation of SPIN for the verification is based on the automata-theoretic approach. So, SPIN employs the explicit state model checking technique to do this automata-based verification. A general introduction to the tool can be found in [74], while the theoretical foundations and a detailed user manual are presented in the book [75]. The main characteristics of SPIN are:

– It is a model checker for LTL.

– It is mainly aimed at efficient verification of protocols and software rather than hardware verification. SPIN uses the programming language PROMELA (PROcess MEta LAnguage) as its input language, which allows nondeterminism behaviors that reflects this intended use.

– It uses C language to implement an automata-based algorithm for model checking.

– It adopts various optimization techniques, such as on-the-fly and partial order reduction to reduce the state explosion problem.

– It provides a graphical user interface (Xspin) developed using Tcl/Tk to the model checker together with an interactive simulator.

The structure of a PROMELA program includes a declaration section for message types, channel types, and global variables of various types. Notice that each command in PROMELA can be seen as a guarded command. Various processes may be defined in a PROMELA program; each process is defined by a name and a list of accepted arguments. The behavior of each process is defined in its body, and each process may include a list of local variables. Processes communicate using global variables and channels. Processes are initially created in the `init` section of the program. They are executed concurrently and can be created by other processes. Since version 4, embedded C code can be included into PROMELA models. Thus, it is possible to verify the implementation (in the C language) of a system directly. A limitation of SPIN is that time and memory cost is still very high when it verifies large systems.

B) CWB-NC

The model checker CWB-NC (Concurrency WorkBench of the New Century, it is previously called Concurrency Workbench of North Carolina) is based on *Alternating Büchi Tableau Automata* (ABTA), which are a variation of alternating tree automata, such as deterministic and nondeterministic Büchi automata. The original release was occurred in the middle of the 1990s at Stony Brook. The tool and a detailed user manual can be download from `http://www.cs.sunysb.edu/~cwb/DOWNLOADS/CWB/current/user.ps`. The main characteristics of CWB-NC are:

– It is a model checker for a variety of temporal logics (e.g., CTL, CTL*, and GCTL*) and modal μ-calculus.

– It supports several process algebra languages for specifying the system behavior, such as Milner's Calculus of Communicating Systems (CCS), version of CCS with prioritized actions

(PCCS), version of CCS with times actions (TCCS), Hoare's Communicating Sequential Processes (CSP), and basic Lotos (LOTOS).

– It implements an automata-based algorithm for model checking.

– It adopts an on-the-fly technique so that the algorithm searches only the part of the state space that needs to be explored to prove or disprove a certain formula. The state space is never constructed a priori.

– It has a text-based user interface, which offers a set of Unix-like command for invoking various analysis routines offered by the tool.

– It provides a general reachability analysis capability.

C) NuSMV

NuSMV [30] is a symbolic model checker written in ANSI C language. It is a reengineering, reimplementation and extension of SMV (Symbolic Model Verifier), the very first model checker developed at the beginning of the 1990s in CMU to implement the BDD-based symbolic model checking techniques. NuSMV has been designed to be an open architecture for model checking, which can be reliably used for the verification of industrial designs [30], as a core for custom verification tools, as a testbed for formal verification techniques, and applied to other research areas. It is a well structured, open and flexible platform for model checking. The different components and functionalities of NuSMV have been isolated and separated in modules. Interfaces between modules have been provided. This reduces the effort needed to extend NuSMV. The main novelty in the current version of NuSMV is the integration of model checking techniques based on propositional satisfiability (SAT), used in bounded model checking, and BDD.

NuSMV is able to process files written in an extension of the SMV language. In this input language, it is possible to describe finite state machines by means of declaration and instantiation mechanisms for modules and processes, corresponding to synchronous and asynchronous composition and to express a set of requirements in CTL and LTL. NuSMV can work in batch mode or interactively, with a textual interaction shell. A NuSMV module is identified by a string, it may accept input parameters, and it may include local variables. The initial value of the module and the evolution of the variables are also defined. It is possible to introduce a derived variable, i.e., a variable which is not part of the state space, but whose value may be derived from other variables. The behavior of the system is described in the mandatory `MODULE main`. Finally, formulae to be checked are provided at the end of the main module. Further information on NuSMV can be found at `http://nusmv.irst.itc.it/`.

D) MCMAS

MCMAS (Model Checker for Multi-Agent Systems) [92] is an OBDD-based symbolic model checker developed particulary for multi-agent systems at the end of 2005s. It can check a variety of properties specified as CTL formulae, epistemic, correctness and cooperation modalities that are distinctive of agents and multi-agent systems. MCMAS is available for download from `http://www-lai.doc.ic.ac.uk/mcmas/download.html`. Interpreted systems provide the formal semantics for MCMAS programs. The dedicated programming language used for describing multi-agent systems in MCMAS is ISPL (Interpreted Systems Programming Language). MCMAS has been implemented in the C++ language. The verification algorithm is presented in [111]. The main components and their structure of implementation are summarized with the following order:

– The input to the model checker is an ISPL file, which is parsed using standard tools. The main parameters of an ISPL file are stored in temporary lists. These parameters are:

    1. Agents' declarations to define a list of ISPL agents.
    2. Evaluation function is defined as `Evaluation <proposition> if <condition_on_states> end Evaluation` where `<proposition>` is an ISPL proposition and `<condition_on_states>` is a truth condition that defines a set of global states for atomic propositions.
    3. Initial states that define the set of initial state conditions.
    4. List of formulae that need to be verified.

– The lists are traversed to build the OBDDs for the verification algorithm. OBDDs are created and manipulated using the CUDD library. The set of reachable states is also computed.

– The formulae to be checked are read from the ISPL file and parsed appropriately.

– Verification is performed by running the algorithm. An OBDD representing the set of states in which a formula holds is computed.

– The OBDD for the set of initial states is then compared to the OBDD corresponding to each formula. In case of an equivalence the tool reports a positive output, otherwise a negative output is produced.

2.4 Discussion

Each application of temporal logic to verification can be investigated and analyzed by answering the following questions: how is the program described? how is the desired property expressed? and what is the model checking complexity? Expressiveness and efficiency are important criteria for answering these questions.

Expressiveness means what properties can and cannot be captured by a temporal logic. Two types of specifications were originally recognized [88]: *safety* and *liveness* properties. Manna and Pnueli [100] have extended the liveness properties to give a standard classification of LTL formulae in an appropriate way, such as *guarantee, response, persistence, reactivity* and *fairness.* The fairness property is divided into *unconditional, weak* and *strong fairness.* In Figure 3, we showed that LTL and CTL cannot be compared in expressiveness, here we present some examples supporting such a fact. The safety properties have equivalent CTL formulae. For instance, from *mutual exclusion* problem, the two processes will never be in their critical regions at the same time. This requirement can be seen as safety property and expressed as: $A\Box(\neg C_1 \vee \neg C_2)$ where C_i is the critical region of the process i. Notably, this formula is CTL-formula, which has an equivalent LTL-formula: $\Box(\neg C_1 \vee \neg C_2)$. However, the guarantee property expressed as $A\Diamond(p \wedge \bigcirc p)$ is not expressible in CTL [58]. On the other hand, every CTL formula that includes an existential path quantifier E cannot be expressed in LTL. For example, the formula $A\Box E\Diamond p$, which means that from all states in the model it is possible to reach a state in which p holds, has not an equivalent LTL formula. Finally, the conjunctions or disjunctions of LTL and CTL formulae can construct a CTL* formula (e.g., $A\Diamond\Box p \wedge A\Diamond A\Box q$), which is not CTL and LTL.

When specifying properties that can be employed by a model checker, many aspects must be considered. First, the capabilities of the temporal logic that we use. For example, if we can express a property in both LTL and CTL, then LTL is preferable because LTL formulae are more succinct (or compact) than CTL. Also, LTL does not require unnecessary path quantifiers before all the temporal modalities. However, an argument against the use of LTL is that its model checking is exponential in the length of the formula (cf. Table 2). Unfortunately, finding an equivalent CTL formula for the LTL formula is not a good solution. For example, the LTL formula $A(\Box p_0 \vee \Box p_1)$ has a longer equivalent CTL formula $A\Box(p_0 \vee A\Box p_1) \wedge A\Box(p_1 \vee A\Box p_0)$. For n disjuncts, $A(\Box p_0 \vee \Box p_1 \vee \cdots \vee \Box p_n)$, the translation into CTL is exponential in n [58]. Notice that the formula $A(\Box p_0 \vee \Box p_1)$ is also a CTL* one. With these results, it is arguable that expressiveness in model checking is the most fundamental characteristic [56].

Efficiency is mainly related to the complexity of model checking algorithm for a logic. In the complexity theory, an algorithm that has a lower complexity in actual use is preferred to one with high complexity. Table 2 summarizes complexity results for the problems of model checking LTL, CTL and CTL*. It is worth noticing that the space complexity results in this table are analyzed with regard to concurrent systems with n processes. From the table, we can observe that: (1) a more expressive logic such as CTL* is less efficient than CTL; and (2) a more compact logic such as LTL is probably to be more convenient (i.e., its properties can be easily and naturally expressed) even if it is less efficient. Thus, it requires some experiences in order to reach a good tradeoff. Moreover, LTL provides the advantage of simplicity, but in terms of significantly less expressiveness than CTL*. The greater expressiveness of fully branching-time temporal logic may get greater computational complexity.

Table 2: The complexity of model checking CTL*, CTL and LTL.

Complexity	CTL*	CTL	LTL
Space	PSPACE-complete [32, 86, 132]	PSPACE-complete [86]	PSPACE-complete [32, 115, 132]
Time	Exponential in the size of the formula and linear in the size of the model [33, 115]	Linear in both the length of the formula and model [31, 32] [33, 115]	Exponential in the size of the formula and linear in the size of the model [89, 115, 141]

In this thesis, we shall investigate how agent communication modeled by social commitments is formalized as temporal modalities in an expressive and succinct logic, ACTL^{*c}, in a convenience manner (cf. Chapter 4). We also will investigate how to address the open issue of model checking commitments and commitment-based protocols using an effective and efficient tool, the CWB-NC model checker (cf. Chapter 4). As we mentioned, a model checking approach has been a very active field of research for the last three decades because of its important applications in verification. Once we have these experiences, we proceed to study and compare different techniques (automat-based and OBDDs-based) to check commitments and commitment-based protocols to identify the suitable one. We also study some theoretical results, such as axioms of commitments, and complexity analysis of model checking commitments: time and space. Before that in Chapter 3, we assess the state of the art in commitment logics and specification languages of commitment-based protocols in order to identify the current limitations, which will be addressed in the proposed approach.

Chapter 3

Computational Logics of Social Commitments

In this chapter, we go through some prominent and predominate proposals to explore the state of the art on how temporal logics can be devoted to define a formal semantics for ACL messages in terms of social commitments and related concepts. We explain each proposal and point out if and how it meets six crucial criteria, four of them introduced by Singh [125] to have a well-defined semantics for ACL messages. Far from deciding the best proposal, our aim is to present the advantages (strengths) and limitations of those proposals to designers and developers, so that they can make the best choice with regard to their needs. We also explore and evaluate current specification languages and different verification techniques that have been discussed within those proposals to specify and verify commitment-based protocols. We finally investigate logical languages of actions advocated to specify, model and execute commitment-based protocols in other contributed proposals.

3.1 Introduction

Having seen that agent communication languages (ACLs) are fundamental mechanisms that enable agents in MASs to interact with each other to satisfy their individual and social goals in a cooperative and competitive manner. Social approaches [123, 125, 13, 16] are advocated to overcome the shortcomings of ACL semantics defined using mental approaches. Commitments are employed in some of these social approaches, which successfully provide a powerful representation for modeling complex multi-agent interactions [125, 160, 98, 11]. In Longman Dictionary, commitment is:

- *a promise to do something or to behave in a particular way*; or
- *something that you have promised you will do or that you have to do.*

In such a definition, commitment imposes loyalty, dedication or devotion towards a person within a social community. In broad terms, commitments are social, public, objective and help represent the states of affairs at different instants in the course of multi-agent interactions. Conventionally, social commitment is an engagement made by an agent, called the debtor, and directed towards another agent, called the creditor, to bring about some condition [124]. While this concept reflects the intuition that the debtor is committed to do something for the creditor, a commitment may be directed toward one agent but the beneficiary might be another agent. For example, after a customer sends the payment for a good to a merchant, a shipper then commits to the merchant to deliver the good to the customer. It is obvious that the shipper is the debtor and the merchant is the creditor, while the beneficiary is the recipient of the good, i.e., the customer.

The notion of social commitments provides many benefits for agent communication. First, the main idea of social semantics is to solely define public aspects of ACL messages for further reference from high-level abstractions without entirely reasoning about agent's mental states and relinquishing part of agents autonomy. To clarify this idea, commitments help us accommodate the tradeoffs between autonomy and interdependency [29]. We would like to model our interacting agents as being autonomous with regard to each other. On the other hand, when the agents are completely autonomous, we then will not have an effective system of agents, but we will have only a number of agents that live in the same environment. Second, in open systems, autonomous agents must be able to cooperate and compete with each other. If there is no interdependence, the agents will be approximately useless. As a result, commitments provide a natural way to characterize the degrees of autonomy and interdependence without getting bogged down in low-level details. Interestingly, all social states of the MAS that contain social facts such as commitments are global, accessible and shared by all of the agents so that: (1) the satisfiability (soundness) [63] of agents' behaviors can be easily monitored and verified [134, 135, 140]; and (2) the agent interoperability between heterogeneous systems can be achieved [27, 5]. Finally, an important aspect of social commitments is that they can be manipulated by means of a set of actions called commitment actions, such as *Create, Discharge* and *Cancel* [124, 143]. More precisely, such actions capture the dynamic behavior of committing agents.

Current proposals considered defining the formal semantics of ACL messages in terms of social commitments by means of either temporal logics [13, 16, 17, 48, 96, 99, 125], which we call "logics of commitments"; or logics of actions [24, 26, 41, 159, 160]. Through this thesis, we call these two types of logics, *computational logics of commitments*. The main advantages of using computational logics can be summarized as follows:

– The notation has a well-defined, and usually well understood, semantics. In other terms, formal semantics conventionally lays down the foundation for a neat, concise and unambiguous meaning of agent messages.

– They provide a high-level, yet concise, form of description consisting of a small range of powerful constructs.

– They allow us to model not only static aspects of an agent state but also the dynamic behavior of agents.

– There is a uniformity of style between the description of the agent behavior and the required properties.

– They facilitate and improve the applicability of the proposed semantics.

Defining a clear and suitable semantics for ACL messages is not enough, we also need to use multi-agent interaction protocols to regulate and coordinate interactive behaviors among automatous and heterogeneous agents in conversations. This thesis adopts commitment-based protocols as they overcome the main limitations of FIPA-ACL protocols and traditional protocols modeled in purely operational terms such as through *finite state machines* or *Petri nets* that describe ordering and occurrence constraints on the exchanged messages, but not the meaning of such messages. The typical use of commitments in commitment-based protocols involves introducing the syntax for the exchanged messages along with a formalization of the meaning of those messages expressed in terms of the commitments of participants. Having such a declarative basis not only simplifies the modeling of protocols—without over-constraints on the communications [159, 160]—being designed or analyzed, but also provides a good basis for flexible specifications that can be searched for correctness [11, 48, 50, 160, 157]. Internally, in commitment-based protocols, agents will not reason about legal sequences but about concrete commitment states and possible paths[1] to reach them [158].

The motivation however is no longer just formally representing and reasoning about social commitments and commitment actions, but becomes the application of various verification techniques, such as full-automatic model checking, semi-automatic verification, local testing, static verification,

[1]A path is an infinite sequence of system's states.

monitoring, etc. in order to respectively: (1) verify the compliance of commitment-based protocols with given specifications at design time; (2) identify the compliant and non-compliant agents at the end of the commitment-based protocol at run time; or (3) track the evolution of commitment statuses at run time. This verification is significant because there is a possibility that an agent may fail to comply with its commitments as it is supposed to. More precisely, it is unrealistic—in open systems (e.g., e-business, e-negotiation) or in any application wherein interacting agents are implemented in different programming languages and having competing objectives—to assume that all autonomous agents will behave (or act) according to the given protocols as they may not behave as they are committed to or at least not wantonly violate or cancel commitments. Furthermore, a formal verification technique, such as model checking, is necessary to help protocol designers either detect unwanted and bad agents' behaviors to eliminate them or enforce desirable agents' behaviors so that such protocols comply with given specifications. Thus, ensuring that only the desirable interactions occur is one of the most challenging aspects of multi-agent system analysis and design.

The motivation and organization of this chapter are to go through prominent and predominate proposals in the literature to explore the state of the art on how temporal logics can be devoted to define a formal semantics for ACL messages in terms of social commitments and associated actions. In our methodology to achieve this objective, we first begin by exploring the historical development of commitments from philosophy, distributed systems and artificial intelligence (AI) perspectives up to the time it gets landed on MASs, including as a special case agent communication (Section 3.2). Second, in Sections 3.3, 3.4 and 3.5, we explore and evaluate current proposals that respectively advocate LTL, CTL and CTL* not only to represent and reason about commitments and associated actions, but also to specify commitment-based protocols. We specifically explain each proposal and point out if and how it meets six crucial criteria, presented in Section 3.2. We also aim at exploring and evaluating different techniques proposed to verify the correctness of commitment-based protocol specifications introduced in those proposals. The point is not to declare one proposal as a winner, but to highlight the advantages (strengths) and limitations of those proposals to designers and developers, so that they can make the best choice with regard to their needs. In Section 3.6, we proceed to succinctly present logical languages of actions such as event calculus, C$^+$ and MAD-P, which have been used to specify, model and execute commitment-based protocols. In Section 3.7, we finally conclude by summarizing current limitations, which we will address in the next chapters.

3.2 Historical Development of Commitments

In this section, we show that commitments in MASs and agent communication are different from the ones having been discussed in various fields such as philosophy, AI and distributed systems.
In philosophy, the notion of commitment to a proposition goes back to at least the philosopher Hamblin [73]. In particular, Hamblin introduced the concept of *commitment store* as a public repository for storing all commitments that have been made by each participant in the dialogue. Hamblin's notion has been further developed in contributions to dialogue games that cater a constructive proof-theory for propositions in a classical logic by the philosophers Walton and Krabbe [148]. In Walton and Krabbe's model, each dialogue game comprises of five basic components, one of them is called *commitment rules*, which define the circumstances under which participants express their commitments to propositions. These commitments reflect the idea that the debtors are accepting a claim as an assertion about the truth of those propositions.

In the AI planning literature, Sacerdoti [113] advocated an approach called *least commitment* planning. This commitment is a psychological one as it is a state of mind. A planner, an essential part of a single agent, creates plans that enable deferring decisions as only the partial ordering decisions are recorded. A key aspect of least commitment strategy is keeping the track of past decisions and their reasons. For example, if you purchase plane tickets (to satisfy the goal of boarding the plane), you should be sure to take them to the airport. If another goal (say, having your hands free to open the taxi door) causes you to drop the tickets, you should be sure to pick them up again. That is a fine notion for a single agent. When least commitments are canceled or stop work for any reasons, an agent is useless (as it is not capable of carrying out any decision).

In distributed systems, mutual commit protocols ensuring that a number of distributed processes agree on some important action are widely acknowledged. The most famous of these protocols is called *two-phase commit* (2PC) [69]. 2PC uses a single coordinator to reach agreement. In the commit-request phase, each process votes Yes or No to perform some required changes. In the commit-outcome phase, these votes are collected by the coordinator. If all processes voted Yes, the coordinator announces a Yes decision. Otherwise a No decision is announced. This represents some kind of commitment as the processes will behave according to the decision of the coordinator. Representing a commitment as a flag inside each process is quite simple, but it is rigid, irrevocable and suitable only for a single agent. Jennings [80] presented commitments as a fundamental notion for efficient coordination in distributed systems. He also used the notion of conventions to monitor the commitments and to define the conditions under which commitments should be reassessed and then to specify the associated actions which should be undertaken in such situations. The author further reformulates different models of coordination in terms of commitments that provide a high-level goal for which the agent must find a suitable course of action. In this work, commitments are modeled as a mental notion, specifically as agent's intentions, purely for the purposes of coordination. In the early of 1990s, the notion of commitments was used in Belief-Desire-Intention (BDI) framework to understand agent's intentions by considering the relationship between agent's persistent goals and actions as stated in the *rational interaction theory* [34]. This approach is well suited to represent a level of joint intentions (equally mutual beliefs) in individual agents' architecture in order to continually use the mental notions. Suppose p is a proposition. A mutual belief notion between two agents means that each agent believes p and each agent believes that the other agent believes p in a nested form. However, satisfying this notion is more difficult in practical applications. Such a limitation (i.e., modeling commitments as a mental notion) incentives Singh [120] to differentiate between two kinds of commitments: *psychological* commitment (i.e., a commitment of one agent to itself) as it is exploited in AI and *social* commitment (i.e., a commitment of one agent to another *to do certain actions*). The author concluded with the psychological commitment is a very restricted form of commitments as it provides unidirectional relationship and once committed to a certain belief or intention, an agent cannot reconsider it, even it gets some positive new evidence or even the commitment contradicts its goal [121]. In order to overcome this problem, the author argued that mutual beliefs are extremely fragile and hard to establish. Though psychological commitments and social commitments are different notions, if the agents are not psychologically committed to their social commitments, they would fail to act as a system as a whole.

Singh's social notion of commitments has been further investigated by Castelfranchi [22] to understand the social interactions among members of groups and organizations by introducing another agent, called *witness* agent, in the context of social commitment. The witness agent certifies the creation of commitment and plays a very crucial role in identifying cheater agents. Castelfranchi forcefully argued that social relationships are irreducible to the mental attitudes. The author only focused on clarifying some concepts to be able to propose a *descriptive ontology* to the theory of organizations without considering the computational aspects of social commitments.

Castelfranchi's social and organizational metaphors provide a more straightforward way to think of MASs in terms of commitments. Singh [122] started focusing solely on social commitment-based MAS approach to tackle distributed artificial intelligence (DAI) problems in heterogeneous and open information environments, called *cooperative information systems*. He found out that the database approach for solving DAI problems is too hard-wired and restrictive while the agent approach gives flexible solutions. The author specifically gave the definition of commitments using the approach called *spheres of commitments*, which incorporates social policies to handle the creation, satisfaction and cancelation actions of commitments and relates commitments to organizational structures of MAS, in which agents can only acquire commitments after adopting their role through MAS execution.

Singh [123] was the first to clearly emphasize the need for defining the semantics for ACLs in terms of social notions. The author pointed out the problem that none of the current ACLs (e.g., KQML) presupposes heterogeneous agents can communicate with each other not because of the lack of practice towards building a formal and public semantics that promotes social agency, but because the current practice is based solely upon mental agency such as beliefs and intentions, which failed to make agents autonomous and heterogeneous. He also argued that reading agents' minds is core to the mental approach, which is not the right direction to deal with the problem. The author then gave many design advantages of following the social approach over the mental one in terms of the meaning of ACL messages and agents construction in the MAS. Singh proceeded to show that the meaning of exchanged messages should be characterized by the following seven communicative acts, namely *assertives, directives, commissives, permissives, prohibitives, declaratives*, and *expressives.*

Singh [124] refined his seminal work introduced in [122] in order to unify normative concepts and commitments and hence coming up with a rich *descriptive ontology* of commitments that technically generalizes Castelfranchi's social notion of groups [22]. In this descriptive ontology, the relationship between commitments and norms is given by making use of "meta-commitments"—a commitment about a commitment—to create a community and its norms. Singh denoted social commitment c by a 4-argument relation involving a proposition p and three agents (i, j, and G): $C(i, j, G, p)$, which means that i is committed towards j in the organizational context G—which maybe an agent or a system of agents such as eBay—to satisfy the proposition p. The agent i that actively makes the social commitment is called the *committer* (or debtor), the agent j to which the commitment is made is called the *committee* (or creditor), and p is called the *content* of the commitment. The context G includes the norms that apply to the group of agents wherein the commitment is established. Its main relational is to resolve disputes between the debtor and creditor, so unacceptable behaviors of the debtor and creditor can be managed. Recall that the debtor and creditor of the commitment are members of the social context. To make the analysis simpler, such a context has been removed from the notation of commitments in later proposals coming up from Singh and other researchers (a commitment c is denoted $C(i, j, p)$). Singh elaborated other three actions, which are in turn used with the actions defined in [122] to manipulate commitments. As we understood, such manipulations provide a principle way to capture the changes in the social state of commitments. El Menshawy et al. [48] classified those actions into: *two* and *three party* actions. The former ones need only two agents to be performed and the latter ones need an intermediate agent to be completed. In the following, we present the intended meaning of those actions and who has the right to perform them.

Two-party actions:

– $\mathit{Create}(i, c)$, performed by the debtor i to instantiate a new commitment c in the system and make the commitment active.
– $\mathit{Discharge}(i, c)$, when the commitment content is true, the commitment c is satisfied.
– $\mathit{Cancel}(i, c)$, performed by the debtor i to revoke its commitment c which is no longer active.
– $\mathit{Release}(j, c)$, performed by the creditor j to free the debtor from carrying out its commitment c which is no longer active.

Three-party actions:

– $\mathit{Delegate}(i, k, c)$, performed by the debtor i to shift its role to another debtor k, which creates a new commitment $c' = C(k, j, p)$ in order to satisfy the current commitment on behalf of i.
– $\mathit{Assign}(j, k, c)$, performed by the present creditor j to transfer the current commitment to another creditor k, which becomes the creditor of a new commitment $c' = C(i, k, p)$.

In Singh's approach [124], commitments and commitment actions are modeled respectively as *abstract objects* with names and *predicates*. For example, $\mathit{Cancel}(i, c)$ denotes a predicate-proposition variable (i.e., a term in the predicate logic), which is true precisely when agent i cancels its commitment c. However, there is no formal semantics for commitments.

Incorporating the notion of commitments into agent communication and advocating it to define the semantics for ACL messages was first introduced in the early of 2000s by Singh [125]. We return to Singh's approach in Section 3.4 along with a discussion of proposals that use CTL to define a formal semantics for ACL messages. A key strong point in Singh's approach [125] is the introduction of four crucial criteria to have a well-defined semantics for ACL messages. In this paper, we explore and evaluate how current proposals in this area of research are handling feature dimensions of those criteria. Our motivation is to highlight the advantages and limitations of those proposals (where they exist). These limitations (or disadvantages) should not be considered fatal, but only a consideration that must be taken. The criteria are:

1. **Formal**. The language must be formal (i.e., there exists a formal syntax and semantics) to eliminate the possibility of ambiguity in the meaning of ACL messages and allow agents to reason about them.

2. **Declarative**. The semantics should focus on what the message means instead of how the message is exchanged. Logics and reasoning techniques are central to this declarative aspect. Also, declarative semantics would have the benefit that the resulting language is not only easy to specify, model and implement, but also focusing on the aims of the interactions, which thus avoids designing unnecessarily over-constrained interaction patterns.
3. **Meaningful**. The semantics should focus on the content and meaning of messages, not on their representation as sequences of tokens.
4. **Verifiable**. We can check if agents are acting according to the given semantics, which implies that the semantics is based on computational models.

Those criteria are agent communication driven. Because our focus is on commitments, we consider two additional commitment-oriented criteria: (1) **Commitment Modeling** (i.e., how the commitment is modeled as, for example, proposition, predicate, fluent, temporal modality); and (2) **Commitment Semantics**, the motivation behind this criterion is to check whether or not there exists a formal semantics for commitments.

3.3 LTL and Commitment-based Agent Communication

In this section, LTL modalities are extended with propositional dynamic logic, new operators such as linear implication and linear operators for capabilities and resources, or past modality. The resulting logical language caters a natural mechanism to: (1) specify commitment-based protocols; (2) define protocol actions and their effects; and (3) express the content of commitments. LTL modalities are also enriched with modalities to represent and reason about commitments. Furthermore, LTL is used to express business properties and formalize regulative protocol specifications.

3.3.1 Giordano and Colleagues

Giordano et al. [66, 67] developed a logical framework based on *Dynamic Linear Time Temporal Logic* (DLTL), an extension of LTL with *propositional dynamic logic* (PDL) in which the *until* operator is indexed with the regular expressions in PDL, to specify and verify commitment-based protocols. In this framework, the protocol describes the meaning of communicative actions in terms of their effects on the system's social state, including commitments and some facts related to the protocol's execution. Such effects are expressed by means of "action laws". The protocol specifies a set of "precondition laws", which define the execution of actions, a set of "causal laws" capturing the dependencies among social states, and a set of temporal constraints, which provide restrictions on the possible correct behaviors of the protocol. In this approach, commitments are modeled as fluents, which are true or false in a state and may change their truth values with the execution of communicative actions [119]. For instance, the fluent $C(Pn, i, j, \alpha)$ means that agent i is committed to agent j to execute action α in a protocol Pn. However, the approach does not introduce conditional commitments (i.e., commitments that become active provided some condition holds) and explicit actions on commitments. Instead, the authors introduced a causal law to define the operational semantics of discharge action. The causal law for discharging the commitment $C(Pn, i, j, \alpha)$ is defined by indicating that in the next state (s), the fluent representing the commitment becomes false whenever α holds in that state (i.e. s). This would be expressed as follows:

$$\Box(\bigcirc\alpha \supset \bigcirc\neg C(Pn, i, j, \alpha))$$

In the verification part, the authors discussed different types of verification problems depending on whether the verification process is carried out at design time such as verifying protocol properties or at run time such as verifying agents compliance with a protocol. In particular, they informally translate DLTL formulae into standard LTL formulae, and commitment-based protocol into PROMELA (the input language of the SPIN model checker [74]) wherein commitments are represented as PROMELA processes and protocol actions are represented as PROMELA message channels, which are communicating interleaved processes. By informally, we mean that the translation rules are not defined in a systematic and formal way so the soundness of the mapping from DLTL formulae into LTL formulae can be proved. The semantics of this approach is formal, declarative, meaningful and verifiable using model checking. However, by modeling commitments as fluents, we will have the issue discussed in the following remark. Further, the use of the SPIN model checker has a limitation discussed in Remark 2

Remark 1 *The fluents (or atomic propositions) are not suitable for representing and reasoning about commitments and their actions as they do not reflect their intrinsic meaning and focus only on the reference (i.e., truth values). So, when using fluents to model commitments, which are treated as propositions, no formal semantics is given to commitments.*

Remark 2 *Because of the state explosion problem of automata-based model checking techniques [31, 33], the SPIN model checker is usually not applicable in open systems (e.g., protocols) having a large state space.*

3.3.2 Pham and Colleagues

Pham and Harland [107] introduced an approach that extends LTL with linear operators that model resources, actions, and capabilities of interacting agents, the resulting logical language is called TLL. This language is then used to flexibly specify commitment-based protocols from the internal view of participating agents in terms of a set of agents' capabilities, resources and pre-commitments. Those pre-commitments can be negotiated among each other using inference rules and upon being accepted, will form conditional commitments. The authors used negative formulae, which can be regarded as formulae in demand and linear implication expressing the casual relationship in the form of reasoning rules to represent and reason about unconditional and conditional commitments respectively. When the conditions are satisfied, the linear implication will ensure that unconditional commitments become effective. An unconditional commitment is fulfilled whenever the agent successfully carries out the actions required by its commitment. This means the corresponding positive formula—which can be regarded as formula in supply—in TLL is derived. For example, let an item (resource) "a" located at "@" and owned by a merchant Mer be denoted by $item_a@Mer$ and a commitment of Mer to deliver that item be represented in a negative formula as: $item_a@Mer^{\perp}$. Then, the positive formula $item_a@Mer$ will fulfill the commitment $item_a@Mer^{\perp}$ and the two formulae are automatically removed (i.e., the commitment is resolved):

$$item_a@Mer \otimes item_a@Mer^{\perp} \vdash \perp$$

where $\otimes$ is a linear *multiplicative conjunction* and $\vdash$ is an inference relation. A commitment $Com^{\perp}$ can be canceled, denoted by $(Cond \otimes Com)^{\perp}$ where the token $Cond$ is simply generated by the agent's internal database when the agent tries to commit itself toward the commitment Com. As for the case of fulfilling commitment, once $Cond$ is generated, due to the inference rule $Cond \otimes (Cond \otimes Com)^{\perp} \vdash \perp$, the commitment $Com^{\perp}$ is removed and henceforth canceling the commitment of deriving Com.

Constructing protocols from the view of participating agents by making use of proof search and putting the specification of commitments locally at the respective agents according to their roles has an interesting advantage toward avoiding the mapping "from protocol actions to commitment operations". However, some issues are missing in this approach. The first one is syntactic: the realm of commitment has only the debtor, which is not an effective representation as the two main features of commitment representation are social and directive. The second one is in modeling commitments as propositions, which waive formal semantics (cf. Remark 1). The third one is of adopting the linear implication to obtain an unconditional commitment from the conditional one when its condition is true. The linear implication "$\multimap$" represents capabilities of agents in producing and consuming resources. Let p and q be agents' resources, so $p \multimap q$ ensures that the condition before p will be transformed into the condition after q in which p is removed after producing q. We can think of the linear implication as the one dollar p is gone after using it to buy a chocolate bar q. From our perspective, a linear implication functions as the *material implication* ($p \supset q$), which is true when p is false, so we could have an unconditional commitment without satisfying its condition. To evaluate the semantics of this approach, it meets the formal (in terms of introducing logic-based approach to define the meaning of protocol actions via commitments) and meaningful criteria, but it does not consider the verifiability issue. However, since protocols are specified in terms of what is to be achieved rather than how the agents should act, the proposed semantics meets the declarative criterion.

3.3.3 Singh and Colleagues

Desai et al. [39] proposed an interesting way to model business protocols in terms of social commitments and their actions, which reduce the emphasis on the rigid sequence of actions that participating agents must take and facilitate loose coupling among these agents. They also used a particular language called OWL-P introduced in [40] to model business protocols. OWL-P supports the specification and composition constructs to build standard business protocols and incorporates the standard Web ontology language (OWL) for modeling well-defined business processes. The authors argued that the correct composition of business processes can be expressed via individual protocols and their composition constraints and thereby enabling the verification of a wide range of composed processes. To verify properties geared toward the composition of business protocols, the authors informally translated protocols into the PROMELA language, which allows modeling the composition of protocols in terms of a set of processes and analyzing the resulting model with the SPIN model checker. In this technique, as in [67], commitments and their actions are modeled in terms of OWL-P rules, which are mapped respectively into PROMELA processes without capturing all features of commitments such as evaluating the content of commitment at the proper time and PROMELA messages. The checked properties are classified into general properties such as *deadlock* and *livelock* freedom and *protocol-specific properties* that check a specific behavior of participating agents. For example, it is important to ensure that after the buyer sends the payment, the shipper should eventually ship the item represented as *shipper_shipOrder* in which the received item is the same as the one agreed before. This protocol-specific property would be expressed as follows:

$$\Box\big(gateway\_authOk \supset \Diamond(shipper\_shipOrder \wedge buyer\_acceptQuote\_itemID = receiver\_shipment\_item)\big)$$

In this formula, when the payment is received, the proposition *gateway_authOk* holds. It is clear that this semantics is formal, declarative, meaningful and verifiable. However, as far as commitment is considered, no formal semantics is given because commitments are simply considered as simple processes in the PROMELA language. In fact, a formal LTL-based framework is proposed to solely express protocol properties. Moreover, three missing points in this approach are: (1) a formal model for protocols; (2) a formal translation procedure; and (3) a formal semantics for commitment actions themselves.

Singh [127] delineated the model-theoretic semantics for social commitments by postulating some rules that can be used as ways to reason about commitments and solely detach and discharge actions. A commitment becomes detached when its antecedent is true. A commitment that is detached but fails to be discharged indicates a violation. This model particulary extends LTL with modalities for two kinds of commitments: *practical commitments*, which are about what is to be done, and *dialectical commitments*, which are about what holds. In this model, the semantics is interpreted using Segerberg's idea, which maps "each world into a set of set of worlds". For instance, to define the semantics of dialectical commitments, the author introduced a function

$$\mathbb{D} : \mathbb{T} \times \mathcal{A} \times \mathcal{A} \times \wp(\mathbb{T}) \rightarrow \wp(\wp(\mathbb{T}))$$

that produces a set of propositions for each moment, an ordered pair of two agents and proposition where $\mathbb{T}$ is a set of moments, $\mathcal{A}$ is a set of agents, and $\wp(\mathbb{T})$ is the powerset of $\mathbb{T}$. Each proposition is a set of moments. This function yields a set where each member is a consequent proposition, which captures what the debtor would be committed to if the antecedent is met. Thus, the semantics is defined by computing the set of moments where the content holds $[\![\varphi]\!]$ and testing if those moments are among the moments $[\![\tau]\!]$ computed by $\mathbb{D}$ on the commitment antecedent τ:

$$M, s \models C(i, j, \tau, \varphi) \textit{ iff } [\![\varphi]\!] \in \mathbb{D}(s, [\![\tau]\!])$$

Notably, the author focused on identifying the logical terms between the antecedent and content of commitment to avoid the problem of both the linear implication used in [107] and the "strict implication" advocated in [125] to define the semantics of conditional commitments (cf. Section 3.4 for the problem of strict implication). When τ is true, the commitment with the consequent φ comes into being, and when φ is true, the commitment is discharged and no longer active:

$$C(i, j, \tau, \varphi) \wedge \tau \supset C(i, j, \varphi)$$
$$\varphi \supset \neg C(i, j, \varphi)$$

It is obvious that this semantics is formal, declarative and meaningful. However, how in practice the function $\mathbb{D}$ is computed is not specified, which makes the possibility of implementing and applying such an approach unclear [8]. As the semantics is not based on Kripke structures, verifying such a semantics using model checking needs either defining an equivalent semantics using Kripke structures or interpreted systems, or defining a completely new model checking approach for the extended logic from scratch. The semantics of discharge action focusses upon checking whether the truth condition of the commitment content holds or not, but we believe this should be a part of the semantics of the commitment modality not the discharge one. The semantic rule of checking the truth value of the commitment content should not be considered as a precondition but in fact as a postcondition in the form of axiom, saying that when the commitment is discharged, then its content should be true immediately. Throughout the paper, we refer to this problem as *over-specification.* Simply put, such a problem occurs when some specifications are repeated in the semantics of different operators. In our case, this problem happens because the truth condition of the commitment content is part of not only the commitment operator, but also its discharge.

3.3.4 Baldoni and Colleagues

Baldoni et al. [5, 6, 7] observed that current specifications of commitment-based protocols do not account for the *regulative* specifications and focus only on the *constitutive* specifications. The constitutive specification defines the semantics of all agents' actions in terms of how these actions affect the social state while the regulative specification constraints the evolution of the social state. They forcefully argued that the two specifications together define the meaning of the interaction (because the only constraint by which we can say that an interaction governed by protocol is successful is that all commitments are discharged), but it is well suited to distinguish between them. The authors then introduced a framework to model commitment-based protocols that separates the constitutive and regulative specifications. They also showed that the regulative specifications should be based on commitments holding in social states not on the execution of actions in order to address the limitations raised in previous proposals [66, 67] regarding the *openness, interoperability*, and *modularity* of designing MASs, which in fact complicate the re-use of software agents' actions. Furthermore, the authors defined a *first-class, declarative* language, called 2CL, to represent the constraints among commitments (i.e., only the regulative specifications). 2CL provides seven kinds of constraint rules, such as *correlation, co-existence, response*, and *before.* Such constraint rules are defined by making use of equivalent LTL temporal modalities. For instance, the constraint rule [6, 7]:

$$C(i,p,\mathit{assigned\_task}(i,p)) \multimap\!\!\bullet\, (\mathit{refused\_task}(p,i) \vee C(p,i,\mathit{solved\_task}(p,i)))$$

means that a participant p cannot: (1) refuse the assigned task; nor (2) commit to solve it before ($\multimap\!\!\bullet$) the initiator i has taken a commitment to bring about assigning the task of interest to that participant. In LTL, this constraint rule can be expressed as follows:

$$\neg\big(\neg C(i,p,\mathit{assigned\_task}(i,p)\ U\ \big(\mathit{refused\_task}(p,i) \vee C(p,i,\mathit{solved\_task}(p,i))\big)\big)$$

Notice that the commitment of the participant and initiator are modeled as LTL atomic propositions. This semantics is declarative and meaningful. When the protocol specification is used to define the meaning of ACL messages, the semantics of this approach satisfies the formal criterion. However, the approach waives the formal semantics for commitments as commitments are simply modeled by propositions, which suffer from the same problem discussed in (Remark 1). Although the constraint rules can be expressed directly by LTL modalities which opens the way to use different model checkers to check, for example, whether or not agent behaviors comply with the protocol, the authors do not consider the verification issues. To this end, separating the regulative rules from commitments has been criticized recently by Marengo et al. [101]. In particular, Marengo et al. placed regulative rules in the antecedents and consequences of commitments to identify the "duties and rights" of the debtor and creditor.

3.3.5 Verdicchio and Colleagues

Spoletini and Verdicchio [134] developed an "automata-based monitoring module" to keep the interactions of a single agent safe by checking whether the evolution of its commitments on the basis of the events that have occurred in the system is compatible with some desirable properties defined in agent's specifications and expressed in LTL$^{\pm}$ (an extension of LTL with past and future modalities). A commitment is modeled as a predicate with four components: an event e that has created the commitment, a debtor i, a creditor j, and a content φ: $\mathit{Comm}(e,i,j,\varphi)$. The content φ of a commitment is represented by a first-order term and written $\lfloor\varphi\rfloor$ in order to refer to the relevant LTL$^{\pm}$ formula in which $\lfloor\varphi\rfloor$ is called a "truth-preserving translation" of φ. The commitment is fulfilled (resp. violated) when its content is true (resp. false):

$$\mathit{Fulf}(e,i,j,\varphi) \triangleq \mathit{Comm}(e,i,j,\varphi) \wedge \lfloor\varphi\rfloor$$
$$\mathit{Viol}(e,i,j,\varphi) \triangleq \mathit{Comm}(e,i,j,\varphi) \wedge \neg\lfloor\varphi\rfloor$$

In the verification part, the monitoring module comprises of two components [135]: (1) "Word Composer" responsible for collecting all data resulting from agent communication, keeping track of the relevant information, and then elaborating them to evaluate present and past time-stamped words selected from these data; and (2) "Word Analyzer" that translates such words into Alternating Modulo Counting Automata [133] functioning as a language acceptor. An unaccepted word means that the state of the system does not satisfy the property expressed by the monitored formula, and such violation is notified to the agent. This semantics meets formal (by using LTL$^{\pm}$-based framework to define the semantics for agent communication), declarative (where every communicative action is seen as the modification of commitments binding the agent to the others) meaningful and verifiable criteria. However, this approach lacks formal semantics for commitments because commitments are modeled as predicates, which are reduced into propositions as stated by the authors as follows: "to be able to write the properties to be monitored in the form of LTL$^{\pm}$ formulae, we need to translate the first-order logic sentences into propositions". The semantics of fulfillment and violation actions suffer from the over-specification problem. The main limitation of pure predicate logic (or first-order logic) [78] in modeling commitments is introduced in the following remark.

Remark 3 *Consider the following example, a customer Cus commits to pay* \$100 *to a merchant Mer. A naive attempt to translate this declarative sentence into predicate logic might result in the following:* $C(Cus, Mer, pay$ \$100$)$*. However, the commitment notion is* referentially opaque*—which set up* opaque contexts *in which the standard substitution rules usually used in predicate logic [78] cannot be applied. To illustrate this issue, let us assume that* \$100 *is equivalent to* €81 *(for a particular day), but we cannot simply say that the customer commits to the merchant to pay* €81*, i.e.,* $C(Cus, Mer, pay$ €81$)$ *because maybe for Cus and Mer,* €81 *does not have the same value as* \$100*. Recall that semantic value of each term in predicate logic is dependent only on the denotations of its arguments. The operators of predicate logic are said to be truth functional. On the other hand, commitments are not truth functional. Thus, what is meant by referentially opacity is "substituting equivalents into opaque context is not going to preserve meaning" as Wooldridge has eloquently argued in [153].*

By Remark 3, the 'sense' of the formula having opaque context is needed along with its truth value. In the literature of agent communication semantics, there are few proposals that have been introduced to address the limitation of pure predicate logic. For instance, Colombetti [35] proposed an *Extended First-Order Modal* (EFOM) language to represent and reason about commitments. In particular, the author introduced a new set of basic speech-act based ACL, which is named *Albatross* (agent language based on a treatment of social semantics). This ACL is based on the social notion of commitments. The semantics of the main elements of Albatross is based on the EFOM language.

Concretely, the semantics of commitment modality is defined using a certain type of accessibility relation: $f_c : D_{action} \times D_{agent} \times D_{agent} \times S \to 2^{2^S}$, which produces a family set of states for each state, and a typed domain of individual sorts of action and a pair of two agents. The semantics is defined by computing the set $||\varphi||$ of states satisfying the commitment content and testing if this set is among the set of states computed by f_c. Formally:

$$M, s \models C(e, i, j, \varphi) \textit{ iff } ||\varphi|| \in f_c(\delta(e, s), \delta(i, s), \delta(j, s), s)$$

where $C(e, i, j, \varphi)$ is called a *modal predicate* and means that the performance of action e commits i relative to j to bring about φ, $\delta(e, s)$ and $\delta(x, s)$ where $x \in \{i, j\}$ define respectively the denotation of action e and agent x at s. The author showed that the commitment is violated when its content is false. A violation of the commitment $C(e, i, j, \varphi)$ is defined by the following semantic rule:

$$V(e, i, j) \triangleq C(e, i, j, \varphi) \wedge \neg\varphi.$$

As claimed by the author, the logic of commitment is "indeed very weak" and there is no "counterpart of the D axiom". Consequently, an agent can make conflicting commitments. However, the author also claimed that inferential capacities are "sufficient to derive that at least one of two conflicting commitments is going to be violated". He also showed that reasoning on violations allows to deal with conditional commitments. This semantics is formal, declarative and meaningful, but verifiability has not been addressed. The weak aspect of this approach is that it suffers from the lack of intuition and computation problem as it has not been ascribed how in practice the accessibility relation f_c is computed. Also, it does not define: (1) axioms of commitment logic; and (2) the semantics of other commitment actions.

We conclude this section with Table 3, which summarizes the results of evaluating the contributed proposals that advocate LTL (or an extension of LTL) to define the semantics for ACL messages. In the table, we use **For.**, **Dec.**, **Mea.**, and **Ver.** to respectively refer to formal, declarative, meaningful, and verifiable Singh's criteria. We also indicate how the commitments are modeled (**Mod.**) and if a formal semantics (**Sem.**) is defined for them. The last column refers to the verification method. It should be understood that the verification method is not a new criterion. Also, when we classify a semantics of an approach as not verifiable, we mean that the verifiability issue is not considered but still there maybe a possibility to use a tool to verify it.

Table 3: Summary of LTL and commitment-based agent communication.

	Agent Communication Semantics				Commitment		Verification
	For.	**Dec.**	**Mea.**	**Ver.**	**Mod.**	**Sem.**	**Method**
Giordano et al. [66, 67]	√	√	√	√	fluent		model checking (informal translation to SPIN)
Pham et al. [107]	√	√	√		negative formula		
Desai et al. [39]	√	√	√	√	process		model checking (informal translation to SPIN)
Singh [127]	√	√	√		temporal modality	√	
Baldoni et al. [5, 6]	√	√	√		proposition		
Spoletini et al. [134, 135]	√	√	√	√	predicate		monitoring techniques

3.4 CTL and Commitment-based Agent Communication

In this section, we investigate the role of CTL enriched with modalities or predicates to represent and reason about commitments and associated actions. The resulting logical language can be then exploited to define the semantics of ACL messages, specify commitment patterns, define a richer temporal representation of the content of commitments, and specify commitment-based protocols. Such protocols are essential to the functioning of open systems, such as those that arise in most interesting Web applications, service engagements and business processes.

3.4.1 Singh and Colleagues

Venkatraman and Singh [143] introduced an approach for locally testing whether the behavior of an agent in Web-based MASs complies with a commitment-based protocol specified in CTL with the idea of *potential causality*, which is most applied in distributed systems. Their verification method concentrates on the conditions under which an individual agent (called an observer, who participates in the protocol) can check the satisfaction or violation of commitments made by each agent towards each other by making use of a model checking-like technique. This technique is similar to model checking, but it uses a different procedure and performs at run time. Technically, a model checking-like approach can "only falsify (but not verify) the correctness of the construction of the agents" in the MAS. That is, if the observer agent finds an inappropriate execution, this entails that the system does not satisfy the protocol. Each commitment in this approach is a meta-commitment. The content of commitment is expressed as a CTL formula and commitment itself is modeled as a simple variable in the form of abstract data type. In this modeling, there is no formal semantics for commitments. However, when it comes to agent communication, the semantics of this approach is formal (since the protocol is specified in CTL, the formal semantics of ACL messages is well defined), declarative, meaningful (as the meaning of every message is expressed in terms of commitments) and verifiable. Their verification technique is an effective one provided that the system under consideration has a small state space. On the other hand, the semantics of commitment actions is not considered in this work.

Based on Jürgen Habermas's theory of communicative acts [72], Singh [125] proposed three-level meanings for ACLs such that each communicative act is associated with three validity claims: (1) *objective claim* meaning that the debtor is committed to send something that is true; (2) *subjective claim* stating that the debtor is committed to sincerity, i.e., the debtor believes or intends what is communicated; and (3) *practical claim* regarding to the debtor's claim that it is justified in making the communication. Recall that defining the meaning of communicative acts is not different from the meaning of the messages specified in the protocols as argued in the approach proposed in [143]. To avert the shortcoming of [143] regarding the representation of commitments as simple variables, the author extended CTL modalities with three modalities for commitments, beliefs and intentions to model interactions among agents. He particularly presented three accessibility relations to define the semantics of those modalities. For instance, the semantics of the commitment modality is satisfied at s in a model M iff the content is true along every accessible path π defined using an accessible relation $\mathbb{C}(i, j, G, s)$ and emanating from the commitment state s, this would be expressed as follows:

$$M, s \models C(i, j, G, p) \textit{ iff } (\forall \pi : \pi \in \mathbb{C}(i, j, G, s) \Rightarrow M, \pi(s) \models p)$$

where $\Rightarrow$ read as "implies" and the accessibility relation $\mathbb{C}$: $\mathcal{A}\times\mathcal{A}\times\mathcal{A}\times S \rightarrow 2^{\Pi}$ where $\mathcal{A}$ is a set of agents and Π is a set of paths, produces the set of accessible paths along which the commitments made by the debtor i towards the creditor j in the social context G hold at a state $s \in S$. The author used the defined modalities to give the formal semantics of the ACL messages. For example, an informative message $\mathit{inform}(i, j, p)$ can be defined as one creating a commitment with its sender i as debtor, its receiver j as creditor and its consequent asserting the truth of the proposition p specified in the content of the message: $C(i, j, p)$ (objective semantics). The subjective and practical semantics of $\mathit{inform}(i, j, p)$ are respectively defined as $C(i, j, iBp)$ where iBp read as i beliefs p and $C(i, G, \mathit{inform}(i, j, p) \rightsquigarrow p)$ where $\rightsquigarrow$ read as "strict implication", which requires p to hold when $\mathit{inform}(i, j, p)$ holds. In fact, the strict implication is introduced to define the semantics of conditional commitment. Its semantics is given as follows:

$$M, s \models p \rightsquigarrow q \textit{ iff } M, s \models p \textit{ and } (\forall s' : M, s' \models p \Rightarrow (\forall s'' : s' \approx s'' \Rightarrow M, s'' \models q))$$

The formula $p \rightsquigarrow q$ is satisfied at s in a model M iff p holds in the current state s and for all states s' in M, if p holds, then we have that q holds in all states s'' similar to s', i.e., $s' \approx s''$. Given a commitment-based semantics for ACL primitives, the author used it to derive a specification language of commitment-based protocols. The new specification helps analyze the protocol by determining, for example, the compliance of an agent with respect to a given protocol.

Although Singh's approach satisfy formal, declarative and meaningful criteria, it does not clarify the intuition that the accessibility relation $\mathbb{C}$ captures and how accessible paths are computed (throughout the paper, we refer to this problem as *lack of intuition and computation*). More precisely, there is a lack of intuition if the meaning behind the fact that a state or a path is accessible from the state where the commitment holds is not defined. Particularly, an accessibility relation should specify the intuitive relation between a social commitment and accessible paths or states. Furthermore, an accessibility relation should clarify how in a computational model one can determine if a state or a path is accessible. To verify such semantics, the author suggested different levels of verifiability. For example, "Every commitment to a putative fact can be verified or falsified by challenging that putative fact. Every commitment to a mental state can be similarly verified or falsified, but only through the more arduous route of eliciting the agent's beliefs and intentions". Because created and modified commitments can be recorded publicly, the observer agent as in [143] can be used to test the compliance of other agents with a given protocol. However, when the semantics is given in terms of mental states, Woolridge [152] pointed out that it is very difficult to carry out the verification of this property since we do not understand how such states can be systematically attributed to programs. Such a problem stems from the subjective semantics, which refers to a mental component by claiming that communication should be sincere.

The first application of Singh's descriptive ontology of commitments [124] to design coordinated MAS was introduced by Xing and Singh [155]. They particularly proposed a set of commitment patterns inspired by design patterns to composedly model agent interactions. Each pattern captures an important scenario and can be specialized and applied to commitment actions. Different combinations of patterns can yield different kinds of agent interactions. In this approach, each pattern is expressed as a CTL formula. The commitments are simply modeled as abstract data types where the content is a predicate with a vector of domain arguments $\vec{v}$ to pass data values: $C(i, j, G, Pred(\vec{v}))$. Commitment actions are modeled as predicates (cf. Remark 3 to know why predicates are not suitable to represent commitments). The following formula expresses the relationship between the communication proposition *Inform* and action predicate *Create*:

$$\forall i, j, Pred, \vec{v}:\ A\Box[Inform(i, j, Pred(\vec{v})) \supset A\Diamond[Create(i, C(i, j, G, Pred(\vec{v})))]]$$

This formula means that along all paths in all states when agent i informs agent j about a predicate $Pred(\vec{v})$, then along all paths there is a possibility that i creates a commitment to bring about $Pred(\vec{v})$ towards j in the context G. The authors used a statechart to specify the behavior model of each agent. Indeed, the operational semantics provided by statecharts are used as a rigorous basis for coordinating the interactions of agents. To relate such operational semantics with temporal logic specifications, a CTL model is produced from agent's statechart. The soundness of this transformation is proved. The authors continued their approach in [156] by developing: (1) an algorithm to transform the statechart of behavior model of agent into CTL model; (2) a library of commitment patterns; and (3) a library of behavior models of agents along with theorem proving which behavior models comply with which patterns. Notably, the agent communication semantics of this approach is formal, declarative, meaningful and verifiable (using theorem proving). When commitments are considered, no formal semantics is given because they are simply modeled as abstract data types and treated as propositions in CTL model. Recall that a formal CTL-based framework is proposed to give only semantics for agent communication using commitments.

Mallya et al. [96] and [99] developed an extension of CTL with: (1) predicates to represent and reason about commitments and associated actions; and (2) two temporal quantifiers (existential and universal) to describe time points and intervals, to obtain a richer temporal representation for the content of commitments particulary to capture real-life scenarios in a natural way. Concretely, they introduced two predicates ($Breached(c)$ and $Satisfied(c)$) to respectively define the semantics of violation and fulfillment of the commitment c in question. For instance:

$$\begin{aligned} M, s \models Satisfied(c)\ iff\ \Big(&\exists s_3 : s_3 \leq s\ and\ M, s_3 \models Discharge(i, c)\ and,\\ &(\exists s_1 : s_1 < s_3\ and\ M, s_1 \models Create(i, c),\ and\\ &(\forall s_2 : s_1 \leq s_2 < s_3 \supset M, s_2 \models Active(c)))\Big)\end{aligned}$$

The semantics of the $Satisfied(c)$ predicate at s in a model M is defined in terms of whether or not the $Discharge(i, c)$ of the commitment c that has been created in the past and still active holds. A commitment is active if it is not canceled, delegated, assigned, released and discharged yet. The authors assumed that the discharge action "brings about p, and conversely, if p occurs, the discharge action is assumed to have happened". Thus, the performance of a discharge action is equivalent to the satisfaction of p. In fact, this assumption raises many issues. More specifically, Verdicchio and Colombetti [145], in the same proceedings, criticized such an assumption by showing that the above semantics of the *Satisfied*(c) predicate seems bypassing "the problem instead of solving it". As the scop of the discharge action focuses on checking whether or not the truth condition of the commitment content holds, it then suffers from the over-specification problem discussed in [127]. To conclude, such a semantics straightforwardly supports formal, declarative and meaningful criteria for agent communication but not the verifiable criterion. The authors also model commitments and associated actions as predicates, which suffer from the problem discussed in Remark 3.

Mallya and Singh [98] have tried to solve the key tradeoff between flexibility of executing protocols and verification in designing those protocols wherein protocol is defined as a set of allowed computation paths to achieve states that need to be reached in terms of commitments. They particularly proposed an approach for designing commitment-based protocols by extending refinement and aggregation notions of traditional software engineering, so that protocol designers should be able to create new protocols by either refining, reusing or composing existing protocols at design time whose properties are well-understood. Concretely, to compose protocols, the authors presented an algebra of protocol, which comprises of two operators: *merge*—to create a new refined protocol from existing ones—and *choice*—to choose between the computations belonging to different protocols. The authors argued that a protocol that allows many computations is better than the one which allows less computations, giving more choice and flexibility in protocol execution. To check compliance of refined protocols, Mallya and Singh presented a "sound theory" of comparing and then refining protocols using "subsumption of protocols" based on the notion of "state-similarity functions" that help designers verify the correctness of protocol properties in which longer computations subsume shorter ones, if they have similar occurring in the same order. The main idea of the state-similarity function is to say that two states are similar when they are labeled by the same set of propositions. As the protocol in this approach is used to define the meaning of ACL messages, this approach is formal, declarative, meaningful and verifiable using sound theory, which it can be proven. In this sense, the behavior of sound theory is similar to theorem proving, which in general cannot be fully automated [78]. There is no formal semantics for commitments and their actions because they are simply modeled as propositions (cf. Remark 1).

The approach proposed in [98] was further investigated in engineering cross-organizational business processes using protocols having social semantics in a research proposal by Gerard and Singh [65]. They introduced a formal model of protocols and their refinement by developing an analysis tool called "Proton". In particular, they specify the protocol in terms of its agents, guarded messages, and the meaning of each message as a set of actions. To verify refinement of protocols, they used model checking instead of Mallya and Singh's theorem proving to compute whether a protocol refines correctly another one under a given mapping that contains the essential elements for refining any two protocols. Their verification technique first reads protocols and specifications expressed as CTL formulae from files and then translates them into ISPL (the input language of MCMAS [92]). The proposed agent communication semantics is formal, declarative, meaningful and verifiable. Unfortunately, the formal semantics of commitments is entirely missing because the authors modeled commitments as objects that are mapped into domain variables in the ISPL language. These variables cannot represent the real and concrete meaning of commitments. Moreover, the operational semantics of discharge action—"which occurs implicitly when the consequent becomes true"—suffers from the over-specification problem.

Using the approach of Xing and Singh [155, 156], Telang and Singh [137, 139] introduced an interesting business model that uses social commitments and agent-oriented concepts such as goals and tasks inspired by Tropos software engineering methodology [20] to capture complex, long lived business scenarios amongst business partners involved in service engagements [137] or cross-organizational business processes that are the norms in today's economic [139]. As in [156], the authors developed a library of business patterns that model recurring business scenarios wherein each pattern is defined in a highly abstract-level based upon the notion of commitments with some attributes: *name, intent, motivation, implementation,* and *consequence* inspired by classical object-oriented design patterns. In order to verify agent interactions, Telang and Singh proposed two different methods to implement this process. In the former one, they [137] introduced a reasoning algorithm, which takes a business model populated by a set of business patterns and business interactions formalized using the UML sequence diagrams and returns a set of violated commitments. In the latter one, they [139] exploited the NuSMV model checker [30] to verify whether an operational model (a set of business interactions) correctly supports a business model. In such a case, a business model pattern is formalized as a set of CTL formulae. For example, when a commitment is inactive in the current state, then along all paths in the next state it could be inactive, active or detached. This would be expressed as follows:

$$A\Box(Inactive \supset A \bigcirc (Inactive \vee Active \vee Detached))$$

A commitment is simply defined as an isolated SMV module, which can be instantiated as a domain variable in the main module. To evaluate this semantics for agent communication against our criteria, it is formal (where a CTL-based framework is introduced to express business model patterns), declarative, meaningful and verifiable. Interestingly, there is no formal semantics for commitments as they are modeled simply as SMV modules. Also, formal semantics for commitment actions themselves are not considered. With respect to the flexibility of classical commitment machines [158], adopting the UML sequence diagram to model the behaviour of interacting agents forces the temporal ordering of action executions, which looses the flexibility supported by the adoption of commitments as shown in [5, 6].

3.4.2 Torroni and Colleagues

To enable a rich modeling of temporal aspects of commitments, Torroni et al. [140] extended Mallya et al.'s work [99] by using "variables with domains" inside commitments. The authors showed that without such an extension, Mallya et al.'s representation of commitments does not cover some practical situations. For example, a commitment of i towards j to bring about p is going to hold at a given moment in the interval beginning at t_1 and ending at t_2 would be represented in Mallya et al.'s model as follows: $C(i, j, [t_1, t_2]p)$. This modeling enables reasoning about the temporal aspect without considering the p's meaning, but it does not specify the time at which the commitment is satisfied. In Torroni et al.'s model, p is defined as a variable, which is bound to a domain interval: $[T]p, T \in [t_1, t_2]$. So, the commitment above can be written as follows:

$$C(i, j, [T]p), t_1 \leq T \leq t_2$$

When there exists a possible value of T in the range $[t_1, t_2]$, the commitment is satisfied and this value can be used for further inferences. This commitment is violated at time t ($viol(C(i, j, [T]p, t))$) "due to the elapsing at time t of a time interval in which p was supposed to be verified". The authors proposed a specification language called *Commitment Modeling Language* ($\mathcal{CML}$), which consists of a set of domain variables, constraints and rules. They used event calculus axioms not only for reasoning about the effects of commitment actions, but also for a static verification of properties and compliance checking, which tracks the evolution of commitment statuses at run time by making use of *Reactive Event Calculus* ($\mathcal{REC}$), which is implemented in $\mathcal{SCIFF}$, abductive logic programming proof-procedure [1]. In this approach, commitments are modeled as fluents and then there is no formal semantics for commitments (cf. Remark 1 for more information about the limitations of fluents). The proposed semantics for agent communication perspective is formal (where event calculus axioms are used to model commitment actions capturing the meaning of ACL messages), declarative, meaningful and verifiable.

3.4.3 Bentahar and Colleagues

El Menshawy et al. [50] presented a framework comprises of three parts aiming at excluding spurious aspects that plague some of the above approaches (e.g., lacking intuition and computation, modeling commitments as fluents, propositions and predicates, and transforming commitments into domain variables). In the first part, they introduced a new temporal logic, called CTLC, an extension of CTL with modality for commitments. They defined a social accessibility relation R_{sc} to interpret the semantics of commitment modality. This is the first approach that associates the formalism of interpreted systems introduced in [59] to model MASs with the Kripke model in order to compute the accessibility relation and define an intuitive semantics of commitments, which is missing in existing proposals of agent communication models. In the formalism of interpreted systems, each agent $i \in \mathcal{A}$ is characterized by a set of local states L_i, a set of local actions, a local protocol, and a local evaluation function. The set of all global states S is a subset of the Cartesian product of all local states of n agents at a given time: $S \subseteq L_1 \times L_2 \times \ldots \times L_n$. The standard CTL model $M = (S, R_t, V, I)$ is extended to $M' = (S, R_t, R_{sc}, V, I)$ where $R_{sc} : S \times \mathcal{A} \times \mathcal{A} \to 2^S$ is the social accessibility relation for commitments. It is defined as follows:

$$s' \in R_{sc}(s, i, j) \ \textit{iff} \ \ \exists \overline{s} \in S : l_i(s) = l_i(\overline{s}) \ \textit{and} \ l_j(\overline{s}) = l_j(s')$$

where $l_i(s)$ denotes the local state of agent i in the global state s. Intuitively, s' is accessible from s, i.e., $s' \in R_{sc}(s, i, j)$ iff there is an intermediate state $\overline{s}$, so that there is no difference for the debtor i between being in s and $\overline{s}$; but, for the creditor j there is no difference between being in the intermediate state $\overline{s}$ and accessible state s'. In the second part, the semantics of commitments is formally defined as follows:

$$M', s \models C(i, j, \varphi) \ \textit{iff} \ \forall s' \in S, \ \textit{if} \ s' \in R_{sc}(s, i, j), \ \textit{then} \ s' \models \varphi$$

which means that the content φ is true in every accessible state from the current state computed using $R_{sc}(s, i, j)$. In the last part, the authors shown how the problem of model checking CTLC can be formally reduced, using a transformation function $\mathscr{F}$, into the problems of model checking CTLK (an extension of CTL with knowledge modality [105]) and ARCTL (an extension of CTL with action formulae [104]) in order to be able to respectively use the MCMAS and extended NuSMV [90]. The more interesting point in this approach is that the commitment modality is reduced into the knowledge modality, not into domain variables, as follows:

$$\mathscr{F}(C(i, j, \varphi)) = \widehat{K}_i \mathscr{F}(\varphi) \ \wedge \ \mathrm{E}\bigcirc \widehat{K}_j \mathscr{F}(\varphi)$$

where $\widehat{K}_i\varphi$, read as "possible knowledge" of agent i [59], is the dual of agent knowledge $K_i\varphi$ (i.e., $\widehat{K}_i\varphi \triangleq \neg K_i\neg\varphi$) and $E \bigcirc \widehat{K}_j$ is read as there is a path such that in the next state agent j possibly knows φ. The authors also implemented the reduction technique on top of the MCMAS and extended NuSMV to automatically verify the Contract Net Protocol against some desirable properties, such as *reachability, safety* and *liveness*. It is clear that our six criteria for agent communication semantics using commitments are successfully met. However, the authors do not consider the semantics of conditional commitments and commitment actions and their verification.

We conclude this section with Table 4, which summarizes our results of evaluating current proposals that use CTL (or an extension of CTL) to define the semantics for ACL messages using commitments and related concepts.

Table 4: Summary of CTL and commitment-based agent communication.

	Agent Communication Semantics				**Commitment**		**Verification**
	For.	**Dec.**	**Mea.**	**Ver.**	**Mod.**	**Sem.**	**Method**
Venkatraman et al. [143]	$\surd$	$\surd$	$\surd$	$\surd$	abstract data type		model checking-like
Singh [125]	$\surd$	$\surd$	$\surd$	$\surd$	temporal modality	$\surd$	model checking-like
Xing et al. [155, 156]	$\surd$	$\surd$	$\surd$	$\surd$	abstract data type		theorem proving
Mallya et al. [96, 99]	$\surd$	$\surd$	$\surd$		predicate		
Mallya et al. [98]	$\surd$	$\surd$	$\surd$	$\surd$	proposition		theorem proving
Gerard et al. [65]	$\surd$	$\surd$	$\surd$	$\surd$	object		model checking
Telang et al. [139]	$\surd$	$\surd$	$\surd$	$\surd$	module		model checking
Torroni et al. [140]	$\surd$	$\surd$	$\surd$	$\surd$	fluent		$\mathcal{REC}$
El Menshawy et al. [50]	$\surd$	$\surd$	$\surd$	$\surd$	temporal modality	$\surd$	model checking (formal translation to MCMAS and NuSMV)
El Menshawy et al. [52]	$\surd$	$\surd$	$\surd$	$\surd$	temporal modality	$\surd$	dedicated a new model checking algorithm

3.5 CTL* and Commitment-based Agent Communication

In this section, CTL* is extended with modalities or predicates to mainly define semantics of speech-act based ACL with respect to social notions. The resulting logical model is exploited to derive a new specification language of commitment-based protocols.

3.5.1 Colombetti and Colleagues

Verdicchio and Colombetti [144, 145] introduced a logical framework for the definition of ACL semantics based on the concept of social commitments resulting from the performance of speech acts by presenting a branching-time temporal logic, called CTL$^{\pm}$, which enriches CTL* with: (1) "past-directed temporal operators"; and (2) meta-language predicates (or *many sorted first-order language*) containing terms, which denote events, actions, commitments and commitment actions. They also introduced a number of appropriate axioms to describe the constitutive effects of the action types for commitment manipulation. In other terms, each axiom describes the state of affairs that necessarily hold if a token of a given action type is successfully performed. There are basically two possibilities to represent the content of commitment: (1) it can be represented as a CTL$^{\pm}$ formula, in which case, a commitment can be modeled as a modal operator; and (2) it can be represented as a first-order term, in which case, a commitment can be modeled as a first-order formula. The authors forcefully argued that modeling a commitment as a first-order formula rather than a modal operator has the advantage that the "technicalities required by a predicative representation are simpler than the ones required by a modal representation". Such a first-order term can be viewed as the representation of a concrete *Content Language* (CL) statement. To achieve this goal, the syntax of CL statement can be represented as a first-order term in CTL$^{\pm}$. Technically, such a first-order term, say φ, is represented as a CTL$^{\pm}$ formula using a "truth-persevering translation" $\lfloor\varphi\rfloor$. As Wooldridge pointed out in [153], using meta-language predicates containing many sorted first-order terms to model notions that have opaque contexts solves the issues discussed in Remark 3. However, modeling commitments using many sorted first-order terms waives the formal semantics and the sense of formulae, which makes using model checking-based verification inappropriate. Inspired by Reichenbach's terminology [112], the semantics of fulfilling commitments is defined as follows:

$$Fulf(e, i, j, \varphi) \triangleq C(e, i, j, \varphi) \wedge A\Diamond^{-}(Happ(e) \wedge \lfloor\varphi\rfloor)$$

where $\Diamond^{-}$ is read as "sometimes in the past", the predicate $Happ(e)$ means that an event e has happened and $\lfloor\varphi\rfloor$ is a truth-preserving translation of the commitment content φ into a CTL$^{\pm}$ formula. The semantics of fulfillment means that the commitment is fulfilled at s' (Reichenbach's point of event), when the *commitment-inducing event* e has been made at s (Reichenbach's point of speech) where the truth value of $Happ(e)$ and $\lfloor\varphi\rfloor$ are true at some state in the past of s' on every path starting from s and going throughout s'. Notably, this agent communication semantics meets the following criteria: formal, declarative and meaningful. However, the semantics of fulfillment action suffers from the over-specification problem as it considers the truth value of the commitment content, which is the main component of the semantics of commitment itself. The authors also do not consider the verification issue. From our point of view, the verification of this logic by means of model checking is hard to perform as: (1) the proposed logical model uses superfluous translation of the commitment content and past operator that points to the state at which the commitment is made; and (2) the model checking of predicate logic is undecidable as we cannot write a procedure that works for all φ (the proof of this claim is introduced in [78]). Such a temporal language has been extended with two interval operators, dates, and times to express a rich assortment of temporal conditions such as deadlines in a natural way, which is missing in the CL expressions of FIPA-ACL [146]. This approach could be considered as a generalization of the work introduced in [99]. Colombetti et al. [36] and Verdicchio and Colombetti [147] shown that FIPA's communicative act library can be effectively redefined using a commitment-based semantics instead of FIPA's mental semantics with respect to their logical model introduced first in [144] and extended in [145].

3.5.2 Bentahar and Colleagues

The first work on combing social commitments as deontic notions and arguments into the paradigm of ACL was done by Bentahar et al. [13]. In this hybrid approach, the social and public aspects of conversational agents are captured by commitments and the reasoning aspects are defined by means of arguments. The authors claimed that existing approaches introduced to define ACL semantics are "not exclusive but rather complementary". However, this approach waives defining formal semantics of commitments, arguments and semantic link between them. In continuation of this work, Bentahar et al. [14, 16, 17] adopted a temporal logic to address such limitations. In particular, they extended CTL* with: (1) modalities for commitments and two-party actions; (2) argument modality; and (3) a dynamic logic (DL), which captures the actions that agents are committed to achieve. They then introduced two accessibility relations to define the semantics of commitment and argument modalities. For instance, the accessibility relation dedicated to commitments [17] is defined as : $R_{sc} : \mathcal{A}\times\mathcal{A}\times S \rightarrow 2^{\Pi}$, which associates with a state s a set of accessible paths along which an agent commits towards another agent. Thus, the semantics of the commitment modality is given as follows:

$$M, s \models C(i, j, \varphi) \textit{ iff } \forall\pi : \pi \in R_{sc}(s, i, j), M, \pi \models \varphi$$

The commitment formula is satisfied in a model M at s iff the content φ is true along every accessible path started at s and computed by R_{sc}. The semantics of the *Satisfy* (discharge) action is defined in terms of whether the commitment has been created in the past and still active and its content holds or not. This would be expressed as follows:

$$\begin{aligned} M, \pi(s) \models \mathit{Satisfy}(i, C(i,j,p)) \ \mathit{iff} \ \ & M, \pi(s) \models \mathit{Active}(C(i,j,p)) \ \mathit{and} \ \exists s' : T(s') \leq T(s) \\ & \mathit{and} \ M, s' \models \mathit{Create}(i, C(i,j,p)) \ \mathit{and} \ M, \pi(s) \models p \end{aligned}$$

where $T(s)$ gives the time point associated to the state s. A commitment is active iff this commitment was already created, and until the current moment, the commitment was not withdrawn.

$$M, \pi(s) \models \mathit{Active}(i, C(i,j,p)) \ \mathit{iff} \ M, \pi(s) \models \neg \mathit{Withdraw}(i, C(i,j,p)) \ U^{-} \mathit{Create}(i, C(i,j,p))$$

where U^{-} is interpreted as "since (in the past)". To define dynamic conversations, the authors interpreted a speech act not as an action performed on a commitment as usual, but also on a commitment content to enable agents to defend and justify their commitment content and to attack and challenge the commitment content of other agents. This semantics for agent communication meets formal, declarative and meaningful criteria. However, the accessibility relation suffers from the lack of intuition and computation problem. This semantics is also defined in a recursive manner (i.e., the semantics of one action depends recursively on the semantics of one or more other actions), which makes its model checking procedure hard. Also, the semantics of satisfy action suffers from the over-specification problem as it focuses upon checking the truth condition of the commitment content.

El Menshawy et al. [46, 48] addressed the limitations of [14, 16, 17] by developing a logical model based on a new temporal logic, called CTL^{*sc}, which extends CTL* with: (1) past-directed temporal modalities; and (2) modalities for commitments and all associated actions. The proposed logic is used to derive a new specification language of commitment-based protocols to particulary define the meaning of protocol messages in terms of commitments. In particular, the semantics of each action does not depend upon other actions using the notion of accessible and non-accessible paths. For instance, the semantics of fulfillment (discharge) action is satisfied in a model M along a path π_i starting at s_i iff: (1) the commitment was established in the past at s_j (after performing the creation action) through the prefix of π starting from s_j denoted by $\pi_i \downarrow s_j$; (2) all paths starting at state s_{i+1} (the state resulting from the fulfillment action) using R_{sc} at state s_j (where the commitment has been established) are accessible paths; and (3) at the current state s_i, there is still a possible choice of not satisfying the commitment since the prefix $\pi''_i \downarrow s_j$ of a non-accessible path π''_i starting at s_i exists. This would be expressed as follows:

$$
\begin{aligned}
(M, s_i, \pi_i) \models \mathit{Fulfill}(i, C(i,j,\varphi)) \ \mathit{iff}\ &(1)\ \exists j \leq i : (M, s_j, \pi_i \downarrow s_j) \models C(i,j,\varphi)\ \mathit{and}\\
&(2)\ (s_i, \mathit{Fulfill}, s_{i+1}) \in R_t\ \mathit{and}\\
&\quad \forall \pi'_{i+1} \in \Pi^{s_{i+1}} : \pi'_{i+1} \downarrow s_j \in R_{sc}(s_j, i, j)\ \mathit{and}\\
&(3)\ \exists \pi''_i \in \Pi^{s_i} : \pi''_i \downarrow s_j \notin R_{sc}(s_j, i, j)
\end{aligned}
$$

where Π^{s_i} is the set of paths starting at s_i. Thus, the semantics of fulfillment action is cured from the over-specification problem raised in [99, 144, 145, 14, 16, 17]. The idea is that "being accessible means that the content φ is true along all the paths $\pi'_{i+1} \downarrow s_j$ and fulfillment occurs automatically when the content holds by the deadline". They also proposed a new definition of assignment and delegation actions by considering the relationship between the original and new commitment contents, which is missing in the current approaches. The authors expressed in CTL^{*sc} a set of desirable properties called "functional correctness", such as *fairness constraint, safety,* and *liveness.* They also proposed a symbolic verification technique based on reducing the problem of model checking CTL^{*sc} into the problems of model checking LTL^{sc} and CTL^{sc} inspired by the notion introduced in [33] regarding the reduction of model checking CTL* into the problems of model checking LTL and CTL. Notice that LTL^{sc} and LTL^{sc} are the standard LTL and CTL extended with commitments and their actions. In this technique, the participating agents in the protocol are defined either as modules using SMV (the input language of NuSMV) or as a set of local states, local actions, local protocol, and local evaluation function using the ISPL language. However, commitments and their actions are simply defined as domain variables in the SMV and ISPL languages. To conclude this approach, its semantics is formal, declarative, meaningful and verifiable.

To address the verifiability criterion, which is the main limitation of the approaches [14, 16, 17], Bentahar et al. [12, 11] presented a verification technique based on translating ACTL* (an extension of CTL* with modalities for commitments, arguments and commitment actions) and protocols into Alternating Büchi Tableau Automata (ABTA) so that they made use of the CWB-NC model checker[2]. In this technique, commitments are defined as simple variables and agent actions as atomic propositions using CCS (the input language of the CWB-NC). Notably, the approaches of El Menshawy et al. [48] and Bentahar et al. [11] are still suffering from the following shortcomings: (1) the lack of intuition and computation problem; and (2) the consideration of commitments as variables in the CCS. Table 5 summarizes our results of investigating the proposals that adopt

[2]`http://www.cs.sunysb.edu/cwb/`.

CTL* (or an extension of CTL*) to: (1) represent and reason about commitments; (2) define the semantics for ACL messages using commitments; and (3) specify commitment-based protocols and express their properties.

Table 5: Summary of CTL* and commitment-based agent communication.

	Agent Communication Semantics				**Commitment**		**Verification Method**
	For.	**Dec.**	**Mea.**	**Ver.**	**Mod.**	**Sem.**	
Verdicchio et al. [144, 145]	✓	✓	✓		predicate		
Bentahar et al. [13, 14, 17]	✓	✓	✓		temporal modality	✓	
Bentahar et al. [12, 11]	✓	✓	✓	✓	temporal modality	✓	model checking (informal translation to MCMAS and CWB-NC)
El Menshawy et al. [46, 48]	✓	✓	✓	✓	temporal modality	✓	model checking (informal translation to MCMAS and NuSMV)

3.6 Other Logical Languages

As aforementioned in the introduction, commitment-based protocols are flexibly specified outside the agents and independently of their architecture in terms of creation, manipulation and satisfaction of commitments between those interacting agents. Yolum and Singh [158] developed a formalism called "commitment machines" to model, reason about and execute commitment-based protocols using a CTL-like semantics introduced by Singh [125]. The main idea of the commitment machine is to label states with commitments as well as some literals (if any) holding in those states, while transitions among states are labeled with actions applied to these commitments. Such a formalism is enabling agents to logically reason about their actions allowable in the protocol to compute their "legal computations". Fornara and Colombetti [62] pointed out that commitments lend themselves to operationalization in a more traditional manner. As we understood, this is the idea of compiling a commitment machine into a traditional representation such as a *finite state machine* (FSM) over finite computations, so that "the desired effect can be obtained without representing and reasoning about declarative meanings at run time" [158]. In other terms, such compilation deletes the opportunities for flexibility, which characterizes a commitment representation. Further, Yolum and Singh [158] shown that FSMs resulting from compiling commitment machines are still beneficiary for interacting agents who waive the ability to reason logically. Winikoff et al. [150] used the event calculus to revise some of Yolum and Singh's proposals but without considering the compilation of commitment machines into FSMs. In particular, they improved the methodology by which: (1) commitments are discharged in certain situations, such as nested conditional commitments and symmetrical; and (2) pre-conditions are specified. Furthermore, Winikoff [149] extended the framework of commitment machines to flexibly model interactions among autonomous agents in distributed systems. The new version is called "distributed commitment machines". The authors then proceed to explore the properties of distributed and centralized commitment machines and to show how the properties of distributed commitment machines can be used to provide "a simple implementation" of commitment machine-based interactions in distributed settings. Recently, Singh [126] generalized the formalism of commitment machines introduced in [150, 158] to include natural non-terminal protocols (or those that have cycles) analogous to those in real-life business applications. Singh's commitment machine is compiled into Büchi automaton over infinite computations (i.e., its acceptance condition is infinite). Hereafter, we briefly present two approaches that use an action logical language to specify commitment machines.

3.6.1 Action Logical Language: Event Calculus

Yolum and Singh [159, 160] specified commitment-based protocols using commitment machines with the use of a subset of Shanahan's event calculus [118]. Roughly speaking, event calculus ($\mathcal{EC}$) is a logical language for representing and reasoning about actions and their effects. The basic components of $\mathcal{EC}$ are "fluents" (properties or atomic propositions holding during time intervals), and "events" (or actions happening at time points) [118]. Fluents are initiated and terminated by occurring events (i.e., events manipulate fluents). In this approach, $\mathcal{EC}$ is akin to *many-sorted first-order predicate calculus* with eight predicates and a set of axioms to represent and reason about actions wherein commitments are modeled as predicates (cf. Remark 3 about the problem resulting from using predicates). For example, the discharge action, whose purpose is to successfully fulfill a commitment, is axiomatically defined as follows:

$$\mathit{Terminates}(e(i), C(i,j,p), t) \subset \mathit{Happens}(e(i), t) \wedge \mathit{Discharge}(e(i), C(i,j,p))$$

This axiom means that when an event $e(i)$ has been carried out by i at time t and the commitment is successfully discharged, the original commitment is terminated. Such a discharge axiom can be defined in another way (cf. Winikoff et al.'s framework, Fig. 7 [150]) as follows :

$$\mathit{Terminates}(e(i), C(i,j,p), t) \subset$$
$$\mathit{HoldsAt}(C(i,j,p), t) \wedge \mathit{Happens}(e(i), t) \wedge \mathit{Initiates}(e, p', t) \wedge \mathit{Subsumes}(p', p)$$

The new axiom focuses on checking the status of commitment in question at time t and the existence of a new fluent p' initiated by occurring an event $e(i)$ at the same time and subsumed the commitment content p, and if so the commitment is terminated. Notice that the predicate *Subsumes* checks implicity whether or not p holds (for other axioms that define other commitment actions, see [150]). Yolum and Singh's approach [159, 160] for defining the semantics of agent communication meets formal, declarative and meaningful criteria. Moreover, the authors used an abductive $\mathcal{EC}$ planner [119] to logically compute planning queries with respect to the $\mathcal{EC}$ axioms (in the form of reasoning rules) that specify protocols: Given the initial protocol state, final protocol state and protocol specification, the planner computes all possible protocol runs (i.e., the sequences of actions) that can be generated between the initial state and final (goal) one. Agents can use the abductive $\mathcal{EC}$ planner to logically calculate protocol runs leading to a desirable outcome. By keeping track of agent's commitments, we can check whether the agent behaviors comply with its commitments. This technique is called static verification that should ideally be carried out at run time. The technique seems promising, but computing all possible runs regarding the commitment in question is very hard to apply when we check open systems having a large state space [2]. Yolum and Singh also pointed out that their specifications achieve flexibility by means of enabling agents to adjust their actions by taking advantages of opportunities and accommodating exceptions that arise at run time by reconstructing plans as necessary. However, the flexibility resulting from reasoning capabilities can be expensive and may increase the code of the agents [158, 126]. Thus, the authors suggested that the specification of protocols can be compiled into FSMs [158] or Büchi automata [126]. Such produced automata can be more complete to capture the important scenarios and consequently be too large for designers to specify, analyze and verify manually. To address the limitation of [159, 160] regarding the verification issue, Yolum [157] presented the main generic properties that are required to help protocol designers analyze and correct the development of commitment-based protocols by signaling possible errors and inconsistencies that arise at run time and determining the applicability of protocols. Such properties are categorized into three classes: *effectiveness, consistency* and *robustness*. Yolum then presented algorithms to semi-automatically verify those properties using any available software tool at design time.

Based on Yolum and Singh's representation of commitment actions [160] in terms of the $\mathcal{EC}$ axioms, Chesani et al. [24] proposed a framework composed of a logical language and a verification procedure. The language defines commitments, deadlines, and "compensations actions" that arise when deadline is expired. The procedure is developed to monitor the commitment statuses at run time. Technically, they formalized the $\mathcal{EC}$ axioms using an extension of abductive logic programming, called $\mathcal{SCIFF}$ [1]. The main feature of $\mathcal{SCIFF}$ is that it is event-driven (reactive) and is based on reactive event calculus ($\mathcal{REC}$). In $\mathcal{REC}$ language, commitment and its associated action are modeled as fluents. For instance, when a commitment $C(i, j, p)$ has been established and the deadline has not expired yet, a fluent $\mathit{Waiting}(C(i, j, p))$ holds. Formally:

$$\mathit{Initiates}(e, \mathit{Waiting}(C(i, j, p)), T) \subset \mathit{Create}(e, i, C(i, j, p))$$

where e is an event, T is the absolute time at which the commitment has been established. By occurring an event regarding to discharging this commitment, the $\mathit{Waiting}(C(i, j, p)$ fluent is terminated and a new $\mathit{Satisfied}(C(i, j, p))$ fluent is instantiated, which means the commitment has been successfully discharged. This would be expressed as follows:

$$\begin{aligned}\mathit{Terminates}(e, \mathit{Waiting}(C(i, j, p)), T) \subset \mathit{HoldsAt}(\mathit{Waiting}(C(i, j, p)), T),\\ \mathit{Discharge}(e, i, C(i, j, p))\\ \mathit{Initiates}(e, \mathit{Satisfied}(C(i, j, p)), T) \subset \mathit{HoldsAt}(\mathit{Waiting}(C(i, j, p)), T),\\ \mathit{Discharge}(e, i, C(i, j, p))\end{aligned}$$

A fluent $\mathit{d\_check}(c, T_D)$ is introduced to check whether a commitment C is satisfied by a time T_D. By defining the values of deadline within the corresponding fluent, the monitor process within the verification procedure observes a set of events at run time reporting the violation of commitment in question (if any). The semantics of this approach is formal, declarative, meaningful and verifiable. However, the verification procedure is suitable for systems that have a small state space as each commitment fluent has many associated fluents. For instance, in the above example, a commitment fluent has two associated fluents and its discharge fluent has 8 associated fluents.

3.6.2 Action Logical Language: C^+

Chopra and Singh [26] specified commitment-based protocols using "nonmonotonic commitment machines" introduced first in [25] with the action logical description language C^+ developed by [68]. A nonmonotonic commitment machine is a modification of a commitment machine [158] to consider situations when agents must act with "incomplete information"; so they need nonmonotonic or defeasible reasoning. The authors adopted C^+ as it is easy to add or remove certain interactions to an existing specification. As in $\mathcal{EC}$, fluents and actions are the basic components in the C^+ language. In this approach, a protocol specification consists of a set of causal laws, which link messages specified as actions that occur at particular time points to their effect on commitments. More precisely, action description describes a transition system of a protocol, a graph with states and actions wherein commitments are modeled as "inertial fluents" (cf. Remark 1 about the problem of using fluents to represent commitments). The interpretation (i.e., semantic value) of such fluents persists from one state to the next state "unless changed by some other law". The authors presented a set of axioms to represent and define the intended meaning of commitment actions as follows [25]:

(1) $Create(i, j, p)$ *causes* $C(i, j, p)$ (2) $Discharge(i, j, p)$ *causes* $\neg C(i, j, p)$

(3) $Cancel(i, j, p)$ *causes* $\neg C(i, j, p)$ (4) $Delegate(i, j, p, k)$ *causes* $\neg C(i, j, p)$ & $C(k, j, p)$

(5) $Release(i, j, p)$ *causes* $\neg C(i, j, p)$ (6) $Assign(i, j, p, k)$ *causes* $\neg C(i, k, p)$ & $C(i, j, p)$

For instance, the axiom of the discharge action means that when the discharge action happens, then the commitment is disabled ($\neg C(i, j, p)$) in "the next state" according to the following rule [25]: *a causes f*, where a is an action and f is a fluent happening in the next state. Notably, this approach meets formal, declarative and meaningful criteria. As in [160], the authors used a static verification technique in the form of reasoning rules in order to verify the compliance of agent behaviors against a given protocol's state machine. As mentioned in the evaluation of [160], this technique is inapplicable in complex systems as the verification procedure needs to record all possible protocol runs from initial and final protocol states to search for commitment states in question.

Desai et al. [41] presented a modular action description geared towards protocols (MAD-P), an extension of the causal logic C^+ [68] to refine and compose protocols from existing ones. This approach enhances the approach of [26] for representing individual protocols. However, as in [26] Desai et al.'s commitments are modeled as inertial fluents and the operational semantics of commitment actions is defined as a set of axioms. Composition rules satisfying some requirements are also defined as a set of C^+ axioms without checking the correctness specification of a protocol composition. Such a challenge is addressed in [39] (cf. Section 3.3). To conclude this section, the advantages and disadvantages of $\mathcal{EC}$ and C^+ and the comparisons between them are discussed in [2]. Although promising, the above languages $\mathcal{EC}$, C^+ and MAD-P are not suitable for model checking that provides a full automatic verification and is effective in complex systems. Moreover, there is no formal semantics for commitments as they are modeled simply as fluents (cf. Remarke 1). Table 6 summarizes our results of evaluating the proposals that advocate logical languages of actions.

Table 6: Summary of logical languages of actions and commitment-based agent communication.

	Agent Communication Semantics				**Commitment**		**Verification**
	For.	**Dec.**	**Mea.**	**Ver.**	**Mod.**	**Sem.**	**Method**
Yolum et al. [159, 160]	$\surd$	$\surd$	$\surd$	$\surd$	predicate		abductive $\mathcal{EC}$ planner
Winikoff et al. [149, 150]	$\surd$	$\surd$	$\surd$		predicate		
Chesani et al. [24]	$\surd$	$\surd$	$\surd$	$\surd$	fluent		$\mathcal{REC}$
Chopra et al. [26]	$\surd$	$\surd$	$\surd$	$\surd$	fluent		static verification
Desai et al. [41]	$\surd$	$\surd$	$\surd$		fluent		

3.7 Summary and Discussion

In this chapter, we reviewed and evaluated most prominent proposals that have advocated computational logics to define semantics for ACL messages in terms of social commitments and related concepts and to use the resulting logical models to derive specification languages of commitment-based protocols. We also investigated different verification techniques that have been proposed to statically, semi-automatically and full-automatically verify these protocols. By highlighting the commonalities, advantages and disadvantages, we hoped that the designers can either make an informed decision when choosing to take the advantages of ACL semantics that satisfy our six crucial criteria or decide on an approach in terms of their own needs, including the availability of a verification tool. We also reviewed other logical languages of actions adopted to specify, model and execute commitment-based protocols. The overall conclusion is summarized in the following points, which particulary state the limitations of those proposals from our perspective:

1. Representing commitments and their actions as predicates, fluents or domain variables waives temporal reasoning, formal semantics and concrete meaning of commitments, which have opaque context.
2. Using informal translation-based techniques to be able to use existing model checkers raises the following issues:
   - Such techniques have the problem of preventing verifying the real semantics of commitments and associated actions as defined in the underlying logics.
   - They only provide partial solution to the problem of model checking commitments as they reduce commitment modalities into simple variables.
   - There are no tools supporting the informal translation-based techniques to carry out the translation process.

3. The $\mathcal{EC}$, C^{+} and MAD-P languages specifying commitment-based protocols are not suitable for model checking-based verification.

The idea of representing commitments and associated actions as modal operators is in fact not new. In the literature about agent communication, few research proposals have supported such idea [17, 45, 48, 50, 52, 125, 127]. On the one hand, although developing such modalities is far from being easy, it allows us to:

1. Combine the sense of formula along with its reference to qualify their truth values and to have a standard form for representing and reasoning about social notions having opaque contexts.
2. Define appropriate axioms incorporated with modal operators. Thus, the validity of those operators will be limited to models that only fulfill the constraints imposed by such axioms.
3. Develop dedicated and implementable model checking algorithms for commitments and their actions and thereby commitment-based protocols.

On the other hand, many of those proposals however substantially suffer from: (1) the intuition and computation problem, in fact, [50] addressed only the computation problem; (2) lacking formal translation method; and (3) the over-specification problem, except [50]. Furthermore, from Tables 3, 4, 5 and 6, we can observe that the number of proposals that use the computational logics LTL and CTL to define artificial languages that artificial agents can use in terms of social notions (commitments and commitment-based protocols) are more than those based on CTL*, although CTL* is more expressive than LTL and CTL.

Many formal languages for agent communication are presented in the literature and discussed in this paper. However, whether a universal language would be possible and whether it would even be desirable are pressing questions. Comparing the situation with human languages, computational programming and logical languages, we can understand why one unique language is not realistic even inside the agent communication community (see Bentahar recent manifesto included in [29]). Which formalism should be used for defining the language semantics and which model checking technique should be used for verifying the correctness of the language semantics are other unsolved problems preventing the definition of a universal language. The main motivations of the thesis are to answer all these questions. In Chapter 4, we use the expressiveness aspect to improve Singh's formal criterion to determine the formalism that we should use to define the semantics of ACL messages in terms of commitments. More precisely, we extend CTL*—which is more expressive and succinct than LTL and CTL—with temporal modalities for commitments and associated actions. In the discussion of Chapter 4, we will show how to answer the rest of those questions.

Chapter 4

An Integrated Approach for Commitment: Logics and Model Checking

In this chapter[1], we propose a logic-based approach for defining a semantics of ACL messages in terms of social commitments. This approach presents a new branching-time temporal logic, called ACTL^{*c}, which extends CTL* introduced by Emerson and Halpern [58] with modalities for social commitments and their actions. We then use ACTL^{*c} to formally derive a new specification language of commitment-based protocols. This specification is intended to be expressive and suitable for model checking. On the basis of this approach, we reduce the problem of model checking ACTL^{*c} into the problem of model checking GCTL* proposed by Bhat et al. [19], which makes the use of CWB-NC model checker possible.

4.1 Introduction

Conventionally, social commitments represent business contracts among autonomous agents with different competing objectives in communicating multi-agent systems. In particular, social commitments are used to define a semantics for ACL messages. Such social semantics supports flexible executions that enable agents to exercise their autonomy by reasoning about their actions and making choices [159]. This flexibility is also related to the accommodation of exceptions that arise at

[1]The results of this chapter have been published in the Journal of Expert Systems with Applications [55].

run time by offering more alternative computations to handle them [98]. Social commitments provide declarative representations of commitment-based protocols by focusing on what the message means and minimally constrain how the message is exchanged [159, 160]. Moreover, they provide a principle basis for checking if the agents are acting according to given protocols [143, 125].

Previous approaches have considered defining the semantics of ACL messages in terms of social commitments by means of computational logics [13, 16, 17, 127, 45, 48, 50, 125], which we called in Chapter 3 *logics of commitments*. Although promising, these approaches are not sufficiently general, rigorous, or clear in their assumptions. More precisely, as we showed in Chapter 3, they suffer from two problems: the intuition and computation problem and the over-specification problem. Moreover, the motivation is no longer just formally representing and reasoning about social commitments and commitment actions, but becomes the application of formal and automatic verification techniques, such as model checking, in order to verify commitments and commitment-based protocols. Specifying commitment-based protocols that ensure flexible interactions is only necessary, but not sufficient to automatically verify their conformance with given properties. Technically, in open environments, the designers and business process modelers of the system as a whole cannot guarantee that an agent complies with its commitments and protocols. A formal verification is beneficial to help them detect and eliminate errors so that such protocols comply with specifications. It also reduces the development cost and increases confidence on the safety, efficiency and robustness at design time.

In contrast to MASs, formal verification techniques for agent communication protocols (e.g., commitment-based protocols) are still in their infancy, due to the more complex nature of protocols and autonomy of agents. As we showed in Chapter 3, few proposals have been proposed to address the above challenge. Some proposals used: (1) local testing technique [143]; (2) static verification technique [24, 26, 140, 160]; and (3) semi-automatic verification technique [157] to detect the compliant and non-compliant agents at the end of the protocol. Although these approaches have made significant progress, they have been criticized by Artikis and Pitt [2] as they are inefficiently applicable in open systems, which have a large state space. Also, their specification languages, such as $\mathcal{EC}$, C^+ and MAD-P, cannot be directly model checked because those languages are not based on temporal logics. Other proposals have defined commitment-based protocols using existing computational logics to be more applicable in today's economy such as e-negotiation [11], cross-organizational business models [139] and business processes [67, 65, 47, 49]. Those protocols have been verified using different model checking techniques. However, they reduced commitments and their actions to simple abstract structures and types using informal translation-based approaches to be able to use existing model checking tools. Such informal translation-based approaches [11, 48, 47, 49] have the problem of preventing verifying the real and concrete semantics of commitments and related concepts as defined in the underlying logics. Those approaches only provide partial solution to the problem of model checking commitments because they reduce commitment modalities into domain variables. This stops distinguishing among various modes of truth such as necessarily true and true in the future. Also, informal translation-based approaches use simple variables [48, 139] and abstract processes [11, 39] that do not account for the meaning of commitments. Moreover, there are no tools supporting the informal translation-based approaches to perform the translation process before the actual verification process is undertaken. Model checking logics of commitments and their actions and commitment-based protocols is then still an open issue.

The motivation of this chapter is to address the above challenges by: (1) formally specifying commitment-based protocols using a new branching-time temporal logic dedicated to commitments and associated actions; (2) automatically verifying commitment-based protocols against given properties using a reduction method to an existing model checker where the semantics is presented; and (3) introducing a new symbolic model checking algorithm for the proposed logic. In fact, we believe that a formal and automatic translation-based approach is more suitable as it allows representing the commitment modality in other temporal modalities, which can still reflect its meaning.

Figure 7 gives an overview of our approach through the thesis, which consists of three parts. In the first part (logic), we develop a new branching-time temporal logic by extending CTL* [58] with modalities to reason about social commitments and associated actions. The introduction of a new logic is motivated by the fact that the needed modal operators for reasoning about social commitments and associated actions cannot be expressed using temporal operators of CTL [31, 56], LTL [109] or even CTL*. Furthermore, the election of CTL* is motivated by our objective to achieve more expressive, succinct and convenient specifications to define the semantics of all commitment actions as CTL* subsumes LTL and CTL. Called ACTL^{*c}, this new logic is particularly used to: (1) express well-formed formulae of commitments and their contents; and (2) formally specify agent's commitment actions. In the second part (specification and reduction), we use social commitments and associated actions to define a new specification language of commitment-based protocols and use the proposed logic to express some protocol properties. These properties aim either to eliminate unwanted and bad agents' behaviors or enforce desirable agents' behaviors at design time. Using commitments as a principle basis for defining the intrinsic meaning of communicative actions in ACLs and applying this idea to model communication protocols provide a general purpose to capture a variety number of communicative actions, which can model different individual communication protocols. Moreover, our approach aims at automatically verifying commitment-based protocols against properties expressed in this part (part 2). This verification step is implemented in the third part by reducing the problem of model checking ACTL^{*c} used to specify commitment-based protocols into the problem of model checking GCTL* (an extension of CTL* with action formulae) [19] in order to directly use the CWB-NC model checker. The implemented reduction method has been used to automatically verify the NetBill protocol, a motivated and specified example in the proposed specification language and the obtained results are reported. In addition to this reduction method, we also provide a new symbolic model checking algorithm dedicated to ACTL^{*c} logic. This algorithm provides a methodology to compute the set of states satisfying ACTL^{*c} formulae, which are encoded using Boolean functions that can be easily represented using OBDDs [21].

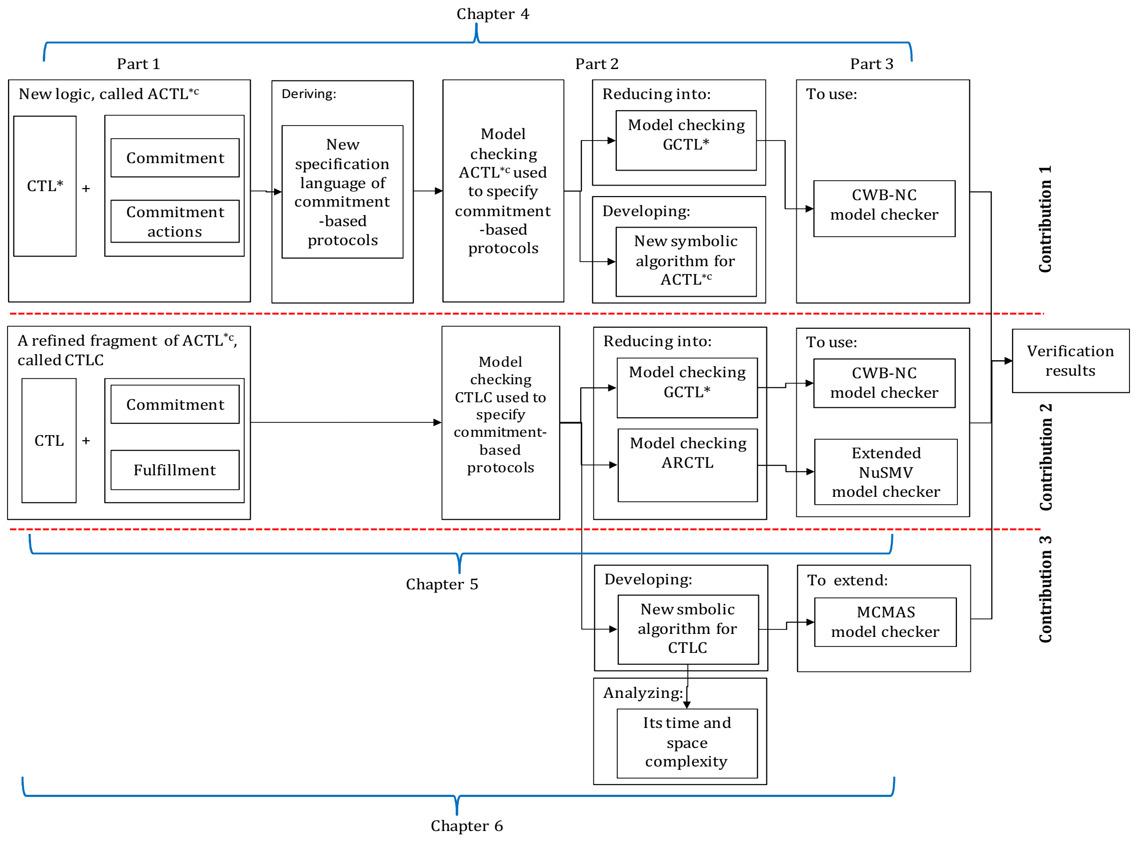


Figure 7: A schematic view of our approach

Based on the experiences and results that we will obtain through the chapter and with some technical reasons that we will present in Section 4.5, we only focus on a refined fragment of ACTL^{*c}, called CTLC, in Chapter 5. We also reduce the problem of model checking CTLC used to specify commitment-based protocols into the problems of model checking GCTL* and ARCTL (an extension of CTL with action formulae [104]) to use the CWB-NC and extended NuSMV model checkers respectively. The motivation behind the use of these two model checkers is to compare between symbolic OBDDs-based technique implemented in NuSMV and automata-based technique implemented in CWB-NC. In Chapter 6, we develop a new symbolic algorithm dedicated to CTLC in lieu of formal reduction method to directly model-check commitments and their actions and commitment-based protocols. We also aim to analyze the time and space complexity of the proposed algorithm. Our full implementation has been done on top of the MCMAS model checker. We call the new tool MCMASC (MCMAS for commitments).

The remainder of this chapter is organized as follows. Section 4.2 presents the life-cycle of commitments and the syntax and semantics of the proposed ACTL^{*c}. In Section 4.3, we use commitments and associated actions to define a new specification language of commitment-based protocols. The reduction of ACTL^{*c} model checking to the one of GCTL* is discussed in Section 4.4. The properties of interest to be checked and the verification of the NetBill protocol using the CWB-NC model checker along with experimental results are also reported in this section. Section 4.5 ends the chapter by comparing our approach with other relevant approaches, identifying the motivations of adopting a refined fragment of ACTL^{*c}, and linking the present chapter with the rest of the thesis.

4.2 Commitments and ACTL^{*c}

This section presents the first part of our approach, which is about describing the life-cycle of social commitments and defining a new branching-time temporal logic, called ACTL^{*c}. The commitment formalism defined by Singh and Huhns [130] has the following properties:

Multi-Agency: the agent who promises or commits to bring about some fact is called the debtor and the other agent who wants the fact to be true is called the creditor. Formally, in our approach social commitments are denoted by $C(Ag_1, Ag_2, \varphi)$ where Ag_1 is the debtor, Ag_2 the creditor and φ a wff in the proposed logic representing the commitment content. Intuitively, $C(Ag_1, Ag_2, \varphi)$ means that Ag_1 publicly commits to Ag_2 that φ holds. We can add a new argument to capture deadlines of social commitments as in [17, 140] or define deadlines using temporal quantifiers over time points or intervals as in [99]. In our approach, we can use the abstract timelines defined in temporal modalities of the underlying logic to define deadlines that can be used to manipulate commitment states.

Conditionally: there are some situations where an agent wants to only commit about some facts when a certain condition is satisfied. Formally, we denote conditional commitments by $\tau \supset C(Ag_1, Ag_2, \varphi)$ where Ag_1, Ag_2 and φ have the above meaning and τ is a wff in our logic representing the commitment condition. This formulation has pros and cons. However, if we regard unconditional commitment as the result of an assertion, we are forced to choose this formulation: if the condition τ of an assertion is true, then there is a way to end up with a commitment to bring about φ. Of course this is weak solution as it uses the logical implication operator (cf. Section 3.3 in Chapter 3). We use $CC(Ag_1, Ag_2, \tau, \varphi)$ as an abbreviation of $\tau \supset C(Ag_1, Ag_2, \varphi)$. In this case, we have $C(Ag_1, Ag_2, \varphi) \triangleq CC(Ag_1, Ag_2, \top, \varphi)$. Another solution would be defining a new accessibility relation for CC to give computationally grounded semantics to CC, but this is beyond the scope of this thesis and will be investigated as future work.

Manipulability: commitments can be modified in a principled manner using agent's actions, called commitment actions [124]. As we mentioned in Chapter 3, these actions are classified into two-party and three-party actions. Two-party actions are *Withdraw (or Cancel), Fulfill (or Discharge)* and *Release*. Three-party actions are *Delegate* and *Assign*. Several approaches assume that agents will respect their commitments. However, this assumption is not always guaranteed, especially in real-life business scenarios where a violation can occur if agents are malicious, deceptive, malfunctioning or unreliable. It is crucial to introduce *Violate* action [35, 99, 17, 45, 48] of social commitments along with their satisfaction and to use model checking to automatically verify agents' behaviors with regard to those actions.

As the notion of commitment is profound to our approach, we now describe commitments in greater detail. Figure 8 shows the life-cycle of a commitment by making use of a UML state diagram. The rounded rectangles represent the states and the directed edges represent the transitions. In particular, the label of a rectangle is the commitment state whilst the label of an edge is an action or an event that causes the transition. At run time, commitments arise between agents, but at design time we specify them between roles. A role helps specify a business relationship between interacting agents in an abstract way. Suppose that a commitment is established by performing a declarative or performative communicative action by an agent, given suitable conditions and conventions. The life-cycle of that commitment proceeds as follows: the commitment could be *Conditional* or *Unconditional*. This is represented by the selection operator. A conditional commitment can move either to *Negotiation* state to negotiate the condition of commitment among the participants or to *Condition* state to check the satisfaction of the condition. The conditional commitment could be negotiated many times until reaching either a mutual agreement about the condition, meaning that the negotiation's outcome is a conditional commitment where the condition is negotiated and agreed upon among the participants, or no agrement can be reached about the condition where the commitment moves to the *Final* state. If the condition is not satisfied, the commitment also moves to the *Final* state. When the unconditional commitment is established, then it may either move to the one of the following states: *Fulfilled*, *Violated*, *Withdrawn*, *Released*, *Delegated*, *Assigned* or move to *Negotiation* state again but this time to negotiate the commitment content. When the participant

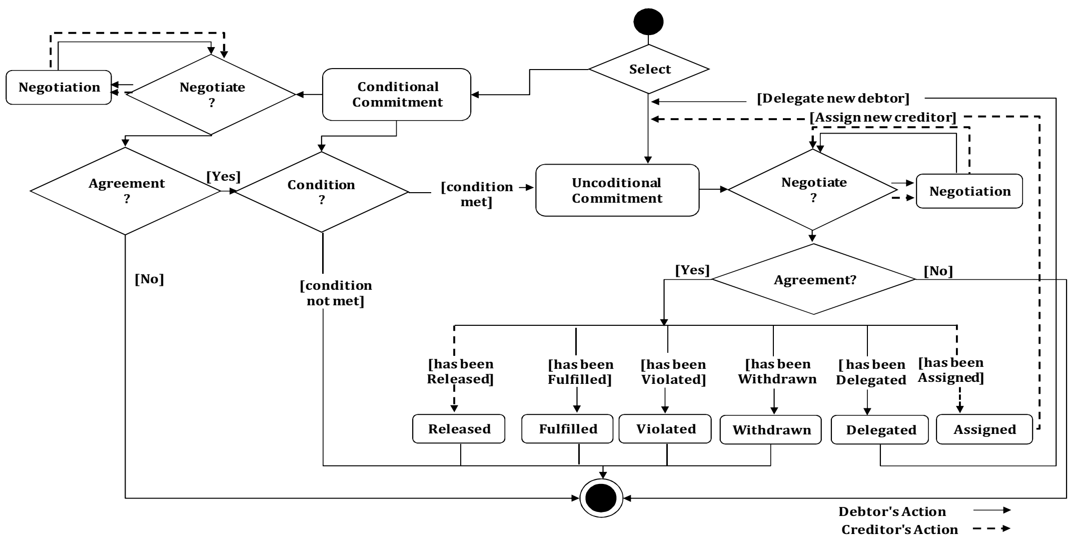


Figure 8: Life-cycle of social commitment

agents reach an agreement, then the commitment moves to the one of the aforementioned states or to the *Final* state if no agreement is reached. In this chapter, we consider all the actions except negotiation, which needs other techniques such as game theory or dialogue games that are beyond the scope of the thesis.

When the debtor performs the withdrawal action, the commitment is withdrawn. Specifically, only the debtor is able to perform this action without any intervention from the creditor. The commitment is fulfilled when the debtor reaches a state at which the content of commitment is satisfied. The social commitment is violated when there is no way to reach a state satisfying the content of commitment. The social commitment can be released by the creditor so that the debtor is no longer obliged to carry out its commitment. The social commitment can be assigned by the creditor, which results in releasing this creditor from the commitment and having a new unconditional commitment with a new creditor. The social commitment can be delegated by the debtor, which results in withdrawing this debtor from the commitment and delegating its role to another debtor within a new commitment. In our approach, some actions such as delegation and assignment can be applied on a commitment multiple times e.g., the delegated commitment can be delegated again or move to another state such as fulfillment and so on. In fact, these actions have the property of "leaving the agents sensitive to race conditions over commitments" [28]. Only release, fulfillment, violation and withdrawal can be applied one time.

4.2.1 Syntax of ACTL^{*c}

ACTL^{*c} extends CTL* with modalities to represent and reason about social commitments and associated actions. It conceptualizes time as a tree-like structure whose nodes correspond to the states (moments) of the system being considered. The branches or paths of the tree represent all choices in the future that agents have when they participate in conversations, while the past is linear. In what follows, we present the syntax according to which formulae in the proposed ACTL^{*c} can be constructed. Like for CTL*, these formulae are classified into state formulae $\mathcal{S}$ and path formulae $\mathcal{P}$. The state formulae are those that hold on a given state such as commitment formula, while the path formulae express temporal properties of paths and action formulae. The state formulae constitute the formulae of ACTL^{*c}.

Definition 3 (Syntax) *The syntax of ACTL^{*c} formulae is given by the following BNF grammar:*

$$\begin{aligned}
&\mathcal{S} ::= p \mid \neg\mathcal{S} \mid \mathcal{S} \vee \mathcal{S} \mid E\mathcal{P} \mid \mathcal{C}\\
&\mathcal{C} ::= C(Agt, Agt, \mathcal{P})\\
&\mathcal{P} ::= \theta \mid \mathcal{S} \mid \neg\mathcal{P} \mid \mathcal{P} \vee \mathcal{P} \mid \bigcirc\mathcal{P} \mid \mathcal{P}\ U\ \mathcal{P} \mid \alpha(Agt, Agt, \mathcal{C})\\
&\alpha ::= Wi \mid Fu \mid Vi \mid Re \mid De \mid As
\end{aligned}$$

- *$p \in \mathcal{PV}$ where $\mathcal{PV}$ is a set of atomic propositions and $\theta \in \Phi_\alpha$ where Φ_α is a set of atomic action propositions.*
- *We use $\mathcal{A} = \{Ag_1, Ag_2, Ag_3, \ldots\}$ as a set of agent names. Agt is nonterminal corresponding to the set $\mathcal{A}$.*
- *The Boolean and temporal operators and their readings are introduced in Chapter 3.*
- *The modal connective $C(Ag_1, Ag_2, \varphi)$ stands for social commitment. It is read as "agent Ag_1 commits towards agent Ag_2 that the path formula φ holds" or equivalently as "φ is committed to by Ag_1 towards Ag_2".*
- *The modal connective $\alpha(Ag_1, Ag_2, \mathcal{C})$ stands for commitment actions, in particular, modal operators Wi, Fu, Vi, Re, De and As stand for Withdraw, Fulfill, Violate, Release, Delegate, and Assign actions respectively. For instance, if α is a Delegate action, then $De(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi))$ is read as "agent Ag_1 delegates its commitment $C(Ag_1, Ag_2, \varphi)$ to agent Ag_3".*
- *Other modal connectives can be abbreviated in terms of the above as usual, for examples, $\Diamond\varphi \triangleq \top U \varphi$ (eventually) and $\Box\varphi \triangleq \neg\Diamond\neg\varphi$ (globally).*

Notice that committing to path formulae is more expressive than committing to state formulae as state formulae are path formulae and it is not the case that all path formulae are state formulae.

4.2.2 Semantics of ACTL^{*c}

We define the semantics of ACTL^{*c} formula φ with respect to the formal model M associated with commitment-based protocols using a Kripke structure as follows: $M = \langle \mathbb{S}, \mathcal{A}, ACT, R_t, \mathbb{V}_s, \mathbb{V}_\alpha, \mathbb{R}_c, \mathbb{L}, \{\approx_{x,y} \mid (Ag_x, Ag_y) \in \mathcal{A}^2\}, I\rangle$ where:

- $\mathbb{S}$ is a finite set of reachable[2] global states in which each global state is a tuple of local states for all agents in the MAS at a given time,
- $\mathcal{A}$ is a set of agent names where each agent is characterized by a set of possible local states representing its complete information about the system at different moments,
- ACT is a set of agent local actions including atomic and commitment actions,
- $R_t \subseteq \mathbb{S} \times \mathcal{A} \times ACT \times \mathbb{S}$ is a total labeled transition relation,
- $\mathbb{V}_s : \mathcal{PV} \to 2^{\mathbb{S}}$ is a function assigning to each atomic proposition a set of states satisfying this proposition,
- $\mathbb{V}_\alpha : ACT \to 2^{\Phi_\alpha}$ is a function assigning to each action a set of atomic action propositions to interpret this action,
- $\mathbb{R}_c : \mathbb{S} \times \mathcal{A} \times \mathcal{A} \to 2^{\Pi}$, where Π is the set of all paths, is a social accessibility relation, which associates with a state s a set of accessible paths along which an agent commits towards another agent,
- $\mathbb{L} : \mathbb{S} \to 2^{\mathcal{A}\times\mathcal{A}}$ is an agency function that associates to each state a set of pairs of two interacting agents in this state such that the first one is the conveyer and the second one is the addressee,
- $\approx_{x,y} \subseteq \mathbb{S}\times\mathbb{S}$ is an accessibility relation defined by $s_i \approx_{x,y} s_j$ for each pair of agents (Ag_x, Ag_y) iff the local states of Ag_x in the global states s_i and s_j are alternatives *à* la Hintikka's accessibility relation (i.e. indistinguishable) and the same thing for Ag_y such that $(Ag_x, Ag_y) \in \mathbb{L}(s_i)$, and
- $I \subseteq \mathbb{S}$ is a set of initial global states.

We assume that the set ACT of actions includes the special action ϵ for the "*null*" action. Thus, when an agent performs the null action, the local state of this agent remains the same. Furthermore, the underlying time domain in our model M is discrete, i.e., the present moment refers to the current state, the next moment corresponds to the immediate successor state in a given path and a transition

[2]This set contains only states that are reachable from I using the transition relation R_t.

corresponds to the advance of a single time-unit. As a modal logic, the temporal modalities of our logic capture the abstraction view of timelines. This meaning is clarified in the following example.

Example 2 *Consider $p =$ deliverGoods, in the context of the NetBill protocol [131]. There, we might want to define a commitment $C(Ag_1, Ag_2,\ E\Diamond^{\leq 7}$ deliverGoods), meaning that a delivery of goods is committed to agent Ag_2 within bounded time, namely 7 days (i.e., time unit is day) by agent Ag_1 where $\Diamond^{\leq 7}p \triangleq p \vee \bigcirc p \vee \bigcirc \bigcirc p \ldots \vee \underbrace{\bigcirc \ldots \bigcirc}_{6\ times} p$.*

The formulation in the above example is motivated by the fact that we are interested in model checking at the design time, so the model should be completely known as the one resulting, for example, from compiling commitment machines into FSMs [158].

Instead of $(s_i, Ag_x, \theta_i, s_{i+1})$, the labeled transitions will be written as $s_i \xrightarrow{Ag_x:\theta_i} s_{i+1}$, which means each transition is labeled with an agent and its action performed during this transition. The paths that path formulae are interpreted over have the form $\pi = \langle s_0 \xrightarrow{Ag_1:\theta_0} s_1 \xrightarrow{Ag_2:\theta_1} s_2 \ldots\rangle$ such that for all $i, x \geq 0, (s_i \xrightarrow{Ag_x:\theta_i} s_{i+1}) \in R_t$. A path in M is then an infinite sequence of reachable global states and labeled transitions with agents and their actions. $\pi(k)$ refers to the k-th state in this sequence. The set of all paths starting at s_i is denoted by Π^{s_i} while $\langle s_i, \pi\rangle$ refers to the path π starting at s_i. $\pi \uparrow s_i = \langle s_i \xrightarrow{Ag_x:\theta_i} s_{i+1} \xrightarrow{Ag_{x+1}:\theta_{i+1}} s_{i+2} \ldots\rangle$ is the suffix of the path π starting from the state s_i. $\pi \downarrow s_i$ is the prefix of π starting at s_i. When a state s_j is a part of a path π, we write $s_j \in \pi$. Also, when a transition $s_i \xrightarrow{Ag_x:\theta_i} s_{i+1}$ is part of a path π, we write $s_i \xrightarrow{Ag_x:\theta_i} s_{i+1} \in \pi$.

To address the spurious problem of intuition and computation regarding to the definition of the accessible path that plagues existing approaches, a path $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$ is an accessible path for the two interacting agents Ag_1 (the debtor) and Ag_2 (the creditor) iff all global states along this path are reachable and accessible states for the two interacting agents using the agency function, which formally means: $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$ iff $s_i = \pi(0)$ and for all $s_j \in \pi$ we have $(Ag_1, Ag_2) \in \mathbb{L}(s_j)$. Intuitively, an accessible path for Ag_1 and Ag_2 from the state s_i is a possible computation of the system for these two agents in the sense of reachability. $\mathbb{R}_c$ has the following properties:

1. $\mathbb{R}_c$ has a form of reflexivity as any accessible path should start from the state itself; i.e., if $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$ then $s_i = \pi(0)$ (*axiom T*).
2. If $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$ then $\forall s_j \in \pi$ if $\pi' \in \mathbb{R}_c(s_j, Ag_1, Ag_2)$ then $\pi' \downarrow s_j \in \mathbb{R}_c(s_j, Ag_1, Ag_2)$. This property is a form of transitivity (*axiom* 4).
3. If $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$ then $\forall s_j \in \pi$ we have $\pi \uparrow s_j \in \mathbb{R}_c(s_j, Ag_1, Ag_2)$.

Thus, the logic of commitment is at least $KT4$, which it is known as $S4$ (cf. Table 1 in Chapter 2 for the meaning of those axioms). An accessible state s_j for Ag_1 and Ag_2 from the state s_i (i.e., $s_i \approx_{1,2} s_j$) is an alternative state in the sense that the two local states of each agent in the global states s_i and s_j are indistinguishable and $(Ag_1, Ag_2) \in \mathbb{L}(s_i)$. It is easy to see that $\approx_{x,y}$ is transitive, and if we assume that it is serial (as in any normal modal logic), we obtain that $\approx_{x,y}$ is also reflexive, symmetric, and Euclidean. Thus, the logic of *Withdraw, Fulfill, Violate, Release, Delegate* or *Assign* action is at least $S5$.

Definition 4 (Satisfaction) *Satisfaction for an $ACTL^{*c}$ state (resp. path) formula φ in the model M at a global state s_i (resp. along the path π starting at global state s_i), denoted as $M, \langle s_i \rangle \models \varphi$ (resp. $M, \langle s_i, \pi \rangle \models \varphi$), is recursively defined as follows:*

- $M, \langle s_i \rangle \models p$ *iff* $s_i \in \mathbb{V}_s(p)$,
- $M, \langle s_i \rangle \models \neg\varphi$ *iff* $M, \langle s_i \rangle \not\models \varphi$,
- $M, \langle s_i \rangle \models \varphi \vee \psi$ *iff* $M, \langle s_i \rangle \models \varphi$ *or* $M, \langle s_i \rangle \models \psi$,
- $M, \langle s_i \rangle \models E\varphi$ *iff* $\exists \pi \in \Pi^{s_i}$ *s.t.* $M, \langle s_i, \pi \rangle \models \varphi$,
- $M, \langle s_i \rangle \models C(Ag_1, Ag_2, \varphi)$ *iff* $\forall \pi \in \Pi^{s_i}$ *s.t.* $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$ *we have* $M, \langle s_i, \pi \rangle \models \varphi$,
- $M, \langle \pi \rangle \models \theta$ *iff* $(\pi(0) \xrightarrow{Ag_x:\beta} \pi(1)) \in R_t$ *and* $\theta \in \mathbb{V}_\alpha(\beta)$ *for an agent* Ag_x,
- $M, \langle s_i, \pi \rangle \models \varphi$ *iff* $M, \langle s_i \rangle \models \varphi$,
- $M, \langle s_i, \pi \rangle \models \neg\varphi$ *iff* $M, \langle s_i, \pi \rangle \not\models \varphi$,
- $M, \langle s_i, \pi \rangle \models \varphi \vee \psi$ *iff* $M, \langle s_i, \pi \rangle \models \varphi$ *or* $M, \langle s_i, \pi \rangle \models \psi$,
- $M, \langle s_i, \pi \rangle \models \bigcirc\varphi$ *iff* $M, \langle s_{i+1}, \pi \uparrow s_{i+1} \rangle \models \varphi$ *on the suffix* $\pi \uparrow s_{i+1}$,
- $M, \langle s_i, \pi \rangle \models \varphi \; U \; \psi$ *iff* $\exists j \geq i$ *s.t.* $M, \langle s_j, \pi \uparrow s_j \rangle \models \psi$ *and* $M, \langle s_k, \pi \uparrow s_k \rangle \models \varphi \; \forall i \leq k < j$,
- $M, \langle s_i, \pi \rangle \models Wi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$ *iff* $M, \langle s_i \rangle \models \neg C(Ag_1, Ag_2, \varphi)$ *and* $\exists s_j$ *s.t.* $s_i \approx_{1,2} s_j$ *and* $M, \langle s_j \rangle \models C(Ag_1, Ag_2, \varphi)$ *and* $\pi \notin \mathbb{R}_c(s_i, Ag_1, Ag_2)$,
- $M, \langle s_i, \pi \rangle \models Fu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$ *iff* $M, \langle s_i \rangle \models C(Ag_1, Ag_2, \varphi)$ *and* $\forall s_j$ *s.t.* $s_i \approx_{1,2} s_j$ *we have* $M, \langle s_j \rangle \models C(Ag_1, Ag_2, \varphi)$ *and* $\pi \in \mathbb{R}_c(s_i, Ag_1, Ag_2)$,
- $M, \langle s_i, \pi \rangle \models Vi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$ *iff* $M, \langle s_i \rangle \models C(Ag_1, Ag_2, \varphi)$ *and* $\forall s_j$ *s.t.* $s_i \approx_{1,2} s_j$ *we have* $M, \langle s_j \rangle \models C(Ag_1, Ag_2, \varphi)$ *and* $M, \langle s_i, \pi \rangle \models \neg\varphi$,
- $M, \langle s_i, \pi \rangle \models Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$ *iff* $M, \langle s_i \rangle \models \neg C(Ag_1, Ag_2, \varphi)$ *and* $\exists s_j$ *s.t.* $s_i \approx_{2,1} s_j$ *and* $M, \langle s_j \rangle \models C(Ag_1, Ag_2, \varphi)$ *and* $\pi \notin \mathbb{R}_c(s_i, Ag_1, Ag_2)$,

– $M, \langle s_i, \pi \rangle \models De(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi))$ *iff* $M, \langle s_i, \pi \rangle \models Wi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$
and $M, \langle s_i \rangle \models C(Ag_3, Ag_2, \varphi))$,

– $M, \langle s_i, \pi \rangle \models As(Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi))$ *iff* $M, \langle s_i, \pi \rangle \models Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$
and $M, \langle s_i \rangle \models C(Ag_1, Ag_3, \varphi))$.

Excluding commitment modality and action formulae, the semantics of ACTL^{*c} state formulae is as usual (semantics of CTL*). The state formula $C(Ag_1, Ag_2, \varphi)$ is satisfied in the model M at s_i iff the content φ is true in every accessible path from this state using $\mathbb{R}_c(s_i, Ag_1, Ag_2)$. The intuition behind defining a commitment as a state formula is to reflect the fact that the debtor agent does not know what will be happening in the future along the path.

The formula $Wi(Ag_1, Ag_2, C(Ag_1,\ Ag_2, \varphi))$, which means *Withdraw* action, is satisfied in the model M at s_i through a path π iff the negation of the formula $C(Ag_1,\ Ag_2, \varphi)$ holds at the current state (i.e., s_i), but the commitment holds in a state s_j which can be seen from the current state s_i through the accessibility relation $\approx_{1,2}$, and π is not accessible. The intuition we capture by this semantics is that by withdrawing its commitment, the debtor agent selects a non-accessible path where the commitment is no more active, but still there is an accessible state where the commitment is active so it can be manipulated by fulfillment or violation. The semantics of the formula $Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$, which means *Release* action, is defined in the same way. The only difference is, however, in the accessibility relation $\approx_{2,1}$ instead of $\approx_{1,2}$ because in the case of release, the creditor who performs the action. Figure 9 depicts the motivation behind

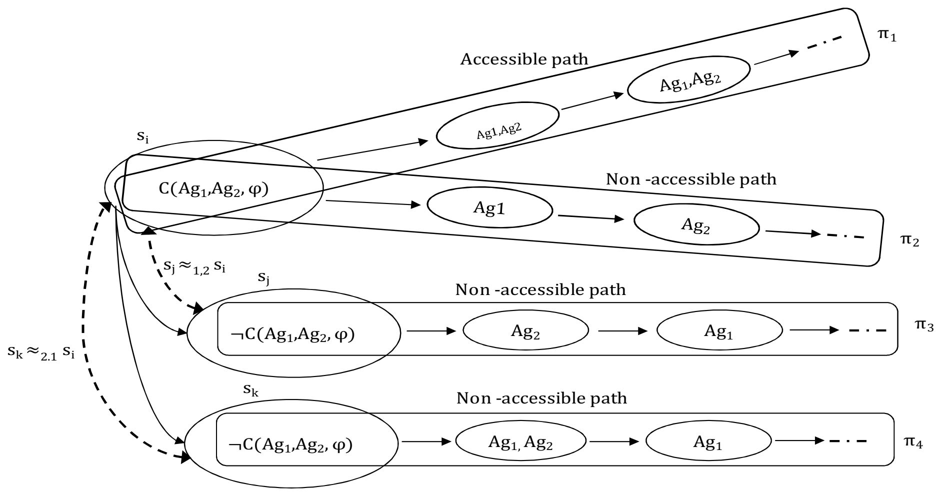


Figure 9: The intuition of the proposed semantics.

the semantics of *Withdraw* (resp. *Release*) action, which is performed by an agent Ag_1 (resp.

Ag_2) at s_j through the path π_3 (resp. π_4), i.e., $M, \langle s_j, \pi_3 \rangle \models Wi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$ (resp. $M, \langle s_k, \pi_4 \rangle \models Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$).

The formula $Fu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$, which means *Fulfill* action, is satisfied in the model M at s_i through π iff (1) the commitment $C(Ag_1, Ag_2, \varphi)$ holds both in the current state (i.e., s_i) and every accessible state from it using $\approx_{1,2}$; and (2) the path π starting at the current state is accessible using $\mathbb{R}_c(s_i, Ag_1, Ag_2)$. The intuition this semantics captures is that Ag_1 fulfills its commitment if the commitment is still active in all alternative states and Ag_1 selects an accessible path through which the content holds. The commitment is violated in a state along a path π iff the commitment is active in all alternative states and its content is false along the path. Intuitively, when an agent violates its commitment (i.e., the commitment content does not hold), this implies that the agent selects a non-accessible path, but the contrary is not always true (i.e., being on a non-accessible path does not imply that the content is false along the path). Figure 9 shows the intuition of the semantics of *Fulfill* (resp. *Violate*) action, which is performed by an agent Ag_1 at s_i through the path π_1 (resp. π_2), i.e., $M, \langle s_i, \pi_1 \rangle \models Fu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$ (resp. $M, \langle s_i, \pi_2 \rangle \models Vi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$). The semantics of the formulae *De* and *As*, which mean *Delegate* and *Assign* actions respectively, is simply defined in terms of the formulae Wi and Re and new commitments. A path π in the model M satisfies action proposition θ iff the label of the first transition on this path satisfies θ.

Notice that our semantics of commitment actions does not suffer from the over-specification problem as those actions are defined in terms of accessible and non-accessible paths, particulary the scope of each commitment action semantics does not check the truth condition of the commitment content, which is the key point of the logical model of commitment, except for the *Violate* action. Furthermore, the persistence of commitments is captured by giving the agent the chance to consider certain accessible states at which those commitments are still active and ready to be manipulated along accessible and non-accessible paths. Compared to CTL*, ACTL^{*c} defines seven new modalities: commitment and its action formulae. These seven modalities cannot be expressed in CTL* because their semantics are defined using the social accessibility relation $\mathbb{R}_c$ and accessibility relation $\approx_{i,j}$ which cannot be captured in any CTL* connector. For instance, $C(Ag_1, Ag_2, \varphi)$ cannot be expressed by $\bigcirc\varphi$ because the content φ is true along every accessible path starting from the current state; it cannot be expressed by $\Diamond\varphi$ because $C(Ag_1, Ag_2, \varphi)$ implies $\Diamond\varphi$, but the reverse is not always true. Thus, any formula containing $C(Ag_1, Ag_2, \varphi)$ cannot be expressed in CTL*.

The notation $\models \varphi$ refers to the validity of the formula φ meaning that φ holds for all models and for all states and paths. The following proposition is a direct result from the proposed semantics.

Proposition 1

1. $\models Fu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi)) \supset \neg Vi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$

2. $\models Fu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi)) \supset \neg Wi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$

3. $\models Fu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi)) \supset \neg Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$

4. $\models Wi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi)) \supset \neg Vi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))$

5. $\models Wi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi)) \supset \neg Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$

6. $\models Vi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi)) \supset \neg Re(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))$

4.3 Commitment-based Protocols

In this section, we proceed to the second part of our approach, which focusses on using the proposed logical model to derive a new specification language of commitment-based protocols. While modeling interactions among agents in terms of commitments provides a good basis for checking compliance, this compliance is only determined by specifying the notion of protocols.

4.3.1 Protocol Specification

Our specification language of commitment-based protocols is defined using: (1) a set of commitment actions without any constraint; (2) a set of autonomous agents (roles) that communicate by sending messages to each other; and (3) a set of propositions related to the application domain of the protocol. In addition to what and when messages can be exchanged, the proposed specification specifies the meaning of messages in terms of their effects on the commitments. Each message is denoted by $Message(snd, rcv, \psi)$ where snd is the sender, rcv the receiver and ψ (a wff in ACTL^{*c}) the message content representing the exchanged information. This message can be mapped into an action on a commitment in which ψ is mapped into the commitment content. We assume that exchanging messages among agents is reliable, which operationally means that messages do not get lost and communication channels are order-preserving. In this manner, a protocol is *public* meaning that it is published and stored in a public repository to be accessible by all participating agents. Notice that our proposed specification can be called a constitutive specification to be compatible with the literature about commitment-based protocol specifications.

The protocol specification begins by a message *MSG*, which can be followed by other messages (in a recursive way) or ϵ message. This message MSG can be directly mapped into either commitment actions or domain proposition actions. Specifically, *MSG* could either be *Withdraw*, *Fulfill*, *Violate*, *Release*, *Assign*, *Delegate*, *Dom_Pro* or ϵ. These messages are defined in our logic with action formula (α) and atomic action proposition θ respectively. The domain proposition such as *requestQuote*, *refund*, etc. is mainly related to the application domain of the protocol. Each domain application can be represented by a suitable ontology. The formal specification language of commitment-based protocols is defined in Table 7 using a BNF grammar with meta-symbols: "|" and ";" for respectively the choice, and message sequence that captures the dependence among some messages as for the delegate message that should follow with the new commitment.

Table 7: The formal specification of commitment-based protocols

Protocol	$::= MSG$
MSG	$::= \Big[Withdraw(Ag_1, Ag_2, \text{COM}) \mid Fulfill(Ag_1, Ag_2, \text{COM})$
	$\mid Violate(Ag_1, Ag_2, \text{COM}) \mid Release(Ag_2, Ag_1, \text{COM})$
	$\mid Assign(Ag_2, Ag_3, \text{COM}); \text{COM}$
	$\mid Delegate(Ag_1, Ag_3, \text{COM}); \text{COM} \mid Dom\_Pro \Big]; \Big[MSG \mid \epsilon \Big]$
COM	$::= CC(Ag_1, Ag_2, \text{Pro}, \text{Pro}) \mid C(Ag_1, Ag_2, \text{Pro})$
Pro	$::=$ A well-formed formula in ACTL^{*c}
Dom_Pro	$::=$ Identify domain propositions

The above formal specification provides both flexibility and rigor wherein a participant agent may use a protocol in any way it chooses as long as it fulfills all of its established commitments. Moreover, the regulative specifications that restrict a set of acceptable executions are also hold in two levels. In the first level, our formal specification uses the dependency operator ";" (it can be seen as a form of causality) to control the follow among messages. In the second level, it uses the abstract timelines incorporated in temporal modalities to place the regulative specifications within the conditions and contents of commitments instead of separating them from the constitutive specifications as done in [6]. From Example 2, the satisfaction of the proposition *deliverGoods* is constrained during an execution by the temporal modality $\Diamond^{\leq 7}$, which could be true either in the current moment or next moment or next next moment and so one till 6 moment. As in [6], the constitutive and regulative specifications of our formal language are mainly focused on commitments that hold in social states and not on commitment actions, which allow us to re-use actions in different scenarios. This is because the meaning of actions will be separated from the context of the protocol where it is used. Moreover, we believe that our formal specification language emphasizes the business meanings of the communications that are specified declaratively. It also includes temporal constraints, which are usually well-related to challenges of real-life distributed computing.

4.3.2 Protocol Compilation

The proposed specification can either be used to reason about the commitment actions at run time [159] or compiled into *finite state machines* (FSMs) at design time. At run time, the agents can logically compute their execution paths that respect the given protocol specifications using some reasoning rules as axioms. However, these rules are not enough to verify protocols against some properties when the system is large and complex (cf. Section 3.6 in Chapter 3 for more details). Moreover, "the flexibility comes at the price of reasoning with declarative representations at run time, which can be expensive and may increase the code footprint of the agents" [159, 126]. For these reasons, we adopt the second approach, which consists in compiling the commitment-based protocol into a FSM where commitments hold on states and actions are labeling transitions. This is compatible with our ACTL^{*c} where commitments are state formulae and actions are path formulae.

As in [126], the proposed protocol specifications are nonterminating analogous to those in real-life applications. In this sense, compiling these protocols requires to consider Büchi automata, which are automata over infinite words. The acceptance condition of Büchi automata handles nonterminating paths (or computations) by considering acceptance states, which are visited infinitely often. From Chapter 2, a Büchi automaton is a five tuple $\mathcal{B}= (Q, \Sigma, Q_0, F, \delta)$ where Q is a set of states; Σ is an alphabet; $Q_0 \in S$ is a set of initial states; $F \subseteq Q$ is a set of accepting states; and $\delta \subseteq Q \times \Sigma \times Q$ is the labeled transition relation. Let $\mathit{inf}(\pi)$ be the set of states that occur infinitely often in the computation π. The computation π is accepting by $\mathcal{B}$ iff $\mathit{inf}(\pi) \cap F$ is not equal to the empty set.

4.3.3 Motivating Example: NetBill Protocol

Let us consider the NetBill protocol (NB) [131] taken from e-business domain to clarify the specification language of commitment-based protocols. The NB protocol is a security and transaction protocol optimized for the selling and delivery of low-priced information goods over the Internet. The original wording from [131] is as follows:

"*The NetBill payment protocol is eight steps (cf. Figure 10). The first message requests a quote based on the customer's identity, to allow for customized per-user pricing, such volume discounts or support for subscriptions. If the quote (step two) is accepted (step three), the merchant sends the digital information to the customer (step four) but encrypts and withholds the key. The customer software constructs an electronic payment order (EPO) describing the transaction and including*

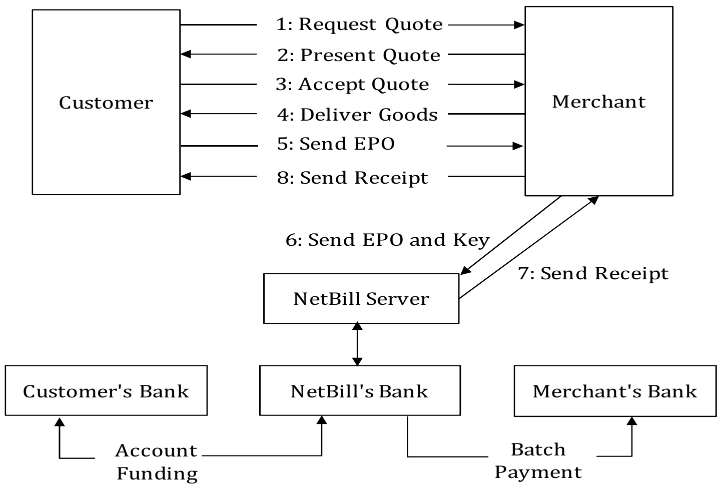


Figure 10: The NB payment protocol [131].

cryptographic checksum of the goods received. The order is signed with the customer's private key and sent to the merchant, who verifies its contents, appends the key for decrypting the goods, endorses the EPO with a digital signature and sends it on to the NetBill server. The NetBill server verifies funds in the customer's NetBill account, debiting the customer and crediting the merchant, and digitally signed receipt, including the key to decrypt the goods, is sent first to the merchant and then on to the customer. The customer software can now decrypt the purchased information and present it to the customer".

Figure 11 depicts an infinite Büchi automaton of the NB protocol where the dashed arrows show the accessible states using the accessibility relation $\approx_{Ag_x,Ag_y}$. In particular, the formal model M (cf. Section 4.2.2) of the NB protocol is compiled using the Büchi automaton $\mathcal{B}$ as follows: Q=$\mathbb{S}$ the set of states (nodes), Σ=$\mathcal{A} \times ACT \cup \{\approx_{Ag_x,Ag_y}\}$ the alphabet used to label transitions, Q_0=I, F=$\{s_6\}$, and δ=$R_t \cup \{\approx_{Ag_x,Ag_y} \mid (Ag_x, Ag_y) \in \mathcal{A}^2\}$ the union of the transition relation and the accessibility relation $\approx_{Ag_x,Ag_y}$. As in [48, 160], we omit the banking procedures by assuming that if a merchant gets an EPO, it can take care of EPO successfully.

Our compilation begins by a request quote message on a path starting at s_0 (the transition between s_0 and s_1 is labeled with the *requestQuote* action proposition: $s_0 \xrightarrow{Cus:requestQuote} s_1$). In this message, the customer (*Cus*) requests a price for some desired goods such as software programs. This request is followed by the merchant's (*Mer*) reply by presenting the price quote as an offer. This present quote message generates the *Mer* agent's commitment to deliver the requested goods to the *Cus* agent after receiving the payment ($CC(Mer, Cus, pay, deliver)$) at s_2

(the transition $s_1 \xrightarrow{Mer:presentQuote} s_2$ is labeled with the *presentQuote* action proposition). The *Cus* agent could either reject or accept this offer. Rejecting this offer means the *Cus* agent releases the offer along a path starting at s_2', which is an equivalent state to s_2 using $\approx_{Cus,Mer}$) (the transition between s_2' and s_7 is labeled with the *Release* action) and the protocol passes through the failure state s_7 as the computation that cycles through states s_0, s_1, s_2, s_2', s_7, namely, $(s_0 \xrightarrow{Cus:requestQuote} s_1 \xrightarrow{Mer:presentQuote} s_2 \xrightarrow{Cus:null} s_2' \xrightarrow{Cus:Release} s_7 \ldots)$ is not accepted by the Büchi automaton, because it does not visit the state s_6 infinitely often. Accepting this offer means the *Cus* agent commits to send the payment to the *Mer* agent along another path starting at the state s_2 (the transition between s_2 and s_3 is labeled with the *acceptQuote* action proposition).

Suppose that the *Cus* agent accepts the received offer, then it has three choices: (1) to withdraw its commitment through a path starting at s_3' (which is equivalent to s_3 using $\approx_{Cus,Mer}$) and passing through s_7, which is not accepted the Büchi automaton; (2) to violate it through a path starting

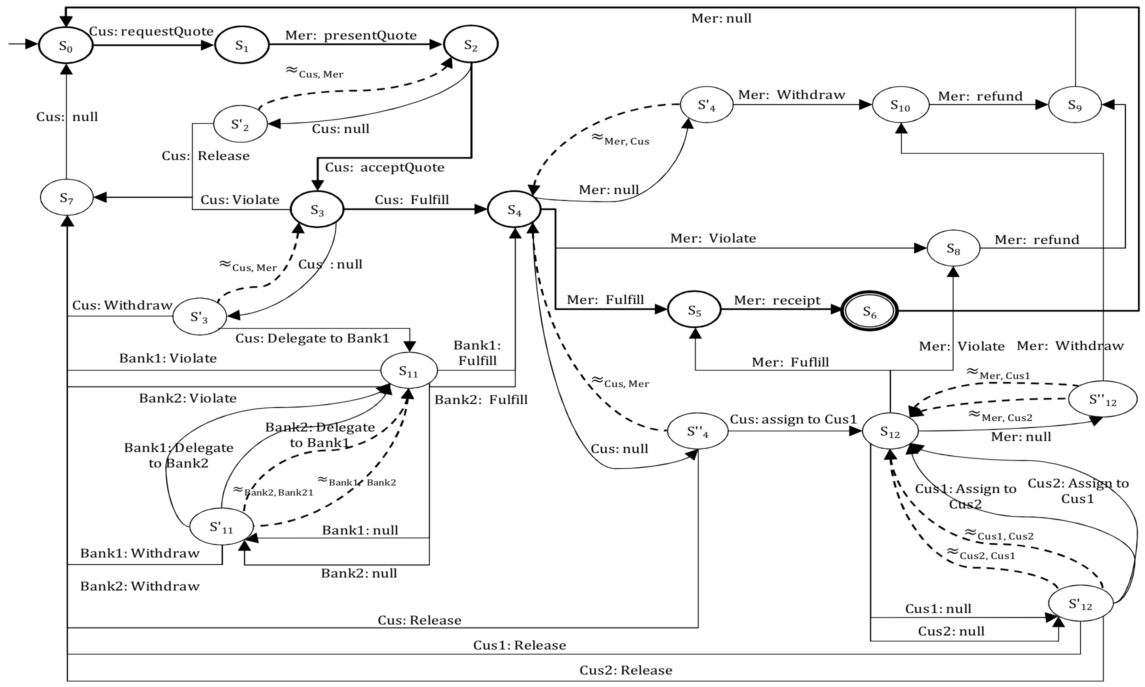


Figure 11: Extension of the NB protocol by considering commitments and actions.

at s_3 and passing through s_7; or (3) to fulfill it by sending the payment to the *Mer* agent along a fourth path starting at the state s_3 but passing through s_4. The computation $(s_3 \xrightarrow{Cus:Fulfill} s_4 \ldots)$ satisfies the formula $Fu(Cus, Mer, C(Cus, Mer, pay))$, i.e., the message *Fulfill* indicates the computation through which the formula is satisfied. In a similar way, the computations $(s_3 \xrightarrow{Cus:Violate}$

$s_7 \ldots$), and ($s'_3 \xrightarrow{Cus:Withdraw} s_7 \ldots$) satisfy the formulae $Vi(Cus, Mer, C(Cus, Mer, pay))$, and $Wi(Cus, Mer, C(Cus, Mer, pay))$ respectively.

According to the semantics of *Delegate* action, the *Cus* agent can delegate its commitment to a financial company (say *Bank₁*) to pay the *Mer* agent on its behalf along a path starting at the state s'_3 and passing through s_{11}. As in [157], the *Bank₁* agent can delegate this commitment to another bank (say *Bank₂*), which delegates the commitment back to the *Bank₁* agent. The bank agents (*Bank₁* and *Bank₂*) delegate the commitment back and forth infinitely often and this is presented by transitions that make a loop between states s_{11} and s'_{11}. In a sound protocol, this behavior should be avoided (in Section 4.4.3, we will show how to verify this issue). The *Mer* agent, before delivering the goods to the *Cus* agent, can withdraw its offer on a path starting at s'_4 (which is equivalent to s_4 using $\approx_{Mer,Cus}$) and passing through s_{10} and then move to the recovery state s_9 (which is not accepted by the Büchi automaton) after refunding the payment to the *Cus* agent, which means performing the *refund* action on a path starting at s_{10}. However, when the *Cus* agent pays for the requested goods and the *Mer* agent delivers them, the *Mer* agent fulfills its commitment along a path starting at the state s_4 and passing through s_5 and then moves to the acceptance state s_6 after sending the receipt to the *Cus* agent along a path starting at s_5. The computation that cycles through states $s_0, s_1, s_2, s_3, s_4, s_5, s_6$, namely, ($s_0 \xrightarrow{Cus:requestQuote} s_1 \xrightarrow{Mer:presentQuote} s_2 \xrightarrow{Cus:acceptQuote} s_3 \xrightarrow{Cus:Fulfill} s_4 \xrightarrow{Mer:Fulfill} s_5 \xrightarrow{Mer:receipt} s_6 \xrightarrow{Mer:null} s_0 \ldots$) is accepted by the Büchi automaton because it visits the state s_6 infinitely often.

Conversely, the *Cus* agent can pay for the requested goods without being delivered by the Mer agent. In this case, the *Mer* agent violates its commitment on another path starting at s_4 and passing through s_8 and then moves to the recovery state s_9 after refunding the payment to the *Cus* agent. As we mentioned, the state s_9 is not accepted by the Büchi automaton. Finally, the *Cus* agent, for some reasons, can assign the commitment of the *Mer* agent to deliver the goods to another customer (say *Cus₁*) along a fourth path starting at s''_4 (which is equivalent to s_4 using $\approx_{Cus,Mer}$) but passing through s_{12}. Specifically, the *Cus* agent releases the current commitment with the *Mer* agent and a new commitment between *Mer* and *Cus₁* is created to deliver the requested goods to the *Cus₁* agent. As for delegate scenario, the assign action can be repeated infinitely often among interacting agents and this scenario, presented by transitions that make a loop between states s_{12} and s'_{12}, is unwanted behavior in our protocol.

4.4 Implementation

There are two methods to verify the proposed commitment-based protocols: (1) by a direct method either by developing a new model checking algorithm from scratch (cf. Appendix A about our symbolic model checking algorithm dedicated to ACTL^{*c}) or extending an existing model checker (as we will do in Chapter 6 for CTLC); and (2) by a reduction (transformation) method into an existing model checker as we present hereafter. In fact, we adopt a reduction method as it is easy to implement. In particular, our implementation is performed in 4 steps as follows: (1) reduce the problem of model checking ACTL^{*c} into the problem of model checking GCTL* [19] in order to be able to use the CWB-NC model checker; (2) encode the NB protocol using CCS (the input language of CWB-NC); (3) express protocol properties; and (4) run the verification of the extended NB protocol against the expressed properties and report the experimental results.

4.4.1 Reducing ACTL^{*c} to GCTL*

GCTL* (read as "Generalized CTL*") extends CTL* [58] by allowing formulae to constrain actions as well as states. The syntax of GCTL* is defined by the following BNF grammar [19]:

$$\mathcal{S} ::= p \mid \neg\mathcal{S} \mid \mathcal{S} \vee \mathcal{S} \mid E\ \mathcal{P}$$

$$\mathcal{P} ::= \theta \mid \mathcal{S} \mid \neg\mathcal{P} \mid \mathcal{P} \vee \mathcal{P} \mid \bigcirc \mathcal{P} \mid \mathcal{P}\ U\mathcal{P}$$

where $p \in \mathcal{PV}$, $\mathcal{PV}$ is a set of atomic propositions and $\theta \in \Phi_\alpha$, Φ_α is a set of atomic action propositions. In fact, this syntax is similar to the syntax of the proposed ACTL^{*c} where the formulae generated by $\mathcal{S}$ and $\mathcal{P}$ are called state formulae and path formulae respectively. State formulae are those that hold on a given state, while path formulae express temporal properties of paths. State formulae constitute the formulae of GCTL*. Other temporal modalities can be defined as abbreviations as usual. To define the semantics of GCTL* formulae, we define the following model.

Definition 5 (Model of GCTL*) *A model $M_G = (S_G, Ac, l_S, l_{Ac}, \rightarrow, I_G)$ is a tuple where S_G is a nonempty set of states; Ac is a set of actions; $l_S : S_G \rightarrow 2^{\mathcal{PV}}$ is a state labeling function; $l_{Ac} : Ac \rightarrow 2^{\Phi_\alpha}$ is an action labeling function; $\rightarrow \subseteq S_G \times Ac \times S_G$ is a transition relation; and $I_G \subseteq S_G$ is a set of initial states.*

Intuitively, S_G contains the states that the system may enter, and Ac the atomic actions that the system may perform. In this sense, the labeling functions l_S and l_{Ac} indicate which atomic propositions hold on a given state and action respectively. The GCTL* semantics [19] follows a standard convention in temporal logic, such as CTL*. A state satisfies $A\varphi$ (resp. $E\varphi$) if every path (resp. some paths) emanating from the state satisfies φ. A path satisfies a state formula if the initial state in the path does, while a path satisfies θ if the path contains at least one transition and the label of the first transition on the path satisfies θ. $\bigcirc$ represents the "next-time operator" and has the usual semantics. $\varphi\ U\ \psi$ holds of a path if φ remains true until ψ becomes true.

The reduction process from the problem of model checking ACTL^{*c} to the problem of model checking GCTL* that allows us to a direct use of CWB-NC is defined as follows: given an ACTL^{*c} model $M = \langle \mathbb{S}, \mathcal{A}, ACT, R_t, \mathbb{V}_s, \mathbb{V}_\alpha, \mathbb{R}_c, \mathbb{L}, \{\approx_{x,y} \mid (Ag_x, Ag_y) \in \mathcal{A}^2\}, I\rangle$ and ACTL^{*c} formula φ, we have to define a GCTL* model $M_G = \mathscr{F}(M)$ and a GCTL* formula $\mathscr{F}(\varphi)$ using a transformation function $\mathscr{F}$ such that $M \models \varphi$ iff $\mathscr{F}(M) \models \mathscr{F}(\varphi)$. The model $\mathscr{F}(M)$ is defined as a GCTL* model $M_G = (S_G, Ac, l_S, l_{Ac}, \rightarrow, I_G)$ as follows:

- $S_G = \mathbb{S}$, $I_G = I$, and $l_S = \mathbb{V}_s$.
- The set Ac is defined as follows: $Ac = \mathcal{A} \times ACT\ \cup\ \{Ag_{x,y} |\ (Ag_x, Ag_y) \in \mathcal{A}^2\}\ \cup\ \{ACC_{x,y} |\ (Ag_x, Ag_y) \in \mathcal{A}^2\}$ where $\mathcal{A}$ and ACT are already used to label temporal transitions in R_t, $Ag_{x,y}$ and $ACC_{x,y}$ are the actions labeling the transitions defined from the accessibility relations $\mathbb{R}_c$ and $\approx_{x,y}$ respectively. Notice that for each element a of Ac, we have two cases:
  1. $a \in \mathcal{A} \times ACT$, in this case a can be written as $(a_{\mathcal{A}}, a_{ACT})$ such that $a_{\mathcal{A}} \in \mathcal{A}$ and $a_{ACT} \in ACT$.
  2. $a \in \{Ag_{x,y} |\ (Ag_x, Ag_y) \in \mathcal{A}^2\}\ \cup\ \{ACC_{x,y} |\ (Ag_x, Ag_y) \in \mathcal{A}^2\}$.
- The function l_{Ac} is then defined as follows:
  1. if $a \in \mathcal{A} \times ACT$, then $l_{Ac}(a) = l_{Ac}((a_{\mathcal{A}}, a_{ACT})) = \mathbb{V}_\alpha(a_{ACT})$,
  2. if not $l_{Ac}(a) = \emptyset$ (notice that $\emptyset \in 2^{\Phi_\alpha}$).
- The labeled transition relation $\rightarrow$ combines the temporal labeled transition R_t and the accessibility relations $\mathbb{R}_c$ and $\approx_{x,y}$ under the following conditions: for states $s_i, s_j \in \mathbb{S}$, $(Ag_x, Ag_y) \in \mathcal{A}^2$,
  1. $(s_i, Ag_x, \theta, s_j) \in\rightarrow$ iff $(s_i, Ag_x, \theta, s_j) \in R_t$,
  2. $(s_i, Ag_{x,y}, s_j) \in\rightarrow$ iff $s_i \approx_{x,y} s_j$,
  3. $(s_i, ACC_{x,y}, s_j) \in\rightarrow$ iff $\exists s, \pi \in \mathbb{R}_c(s, Ag_x, Ag_y)$ and $(s_i, -, -, s_j) \in \pi$ where the second and third arguments of $(s_i, -, -, s_j)$ can take whatever value from $\mathcal{A}$ and ACT respectively.

With respect to the proposed semantics, the transformation of an ACTL^{*c} formula into a GCTL* formula is defined as follows:

- $\mathscr{F}(p) = p$, if p is an atomic proposition,
- $\mathscr{F}(\neg\varphi) = \neg\mathscr{F}(\varphi)$, and $\mathscr{F}(\varphi \vee \psi) = \mathscr{F}(\varphi) \vee \mathscr{F}(\psi)$,
- $\mathscr{F}(E \bigcirc \varphi) = E \bigcirc \mathscr{F}(\varphi)$, and $\mathscr{F}(E(\varphi\ U\ \psi)) = E(\mathscr{F}(\varphi)\ U\ \mathscr{F}(\psi))$,
- $\mathscr{F}(E\Box\varphi) = E\Box\mathscr{F}(\varphi)$, and $\mathscr{F}(C(Ag_1, Ag_2, \varphi)) = A(\Box\ ACC_{1,2} \supset \mathscr{F}(\varphi))$,
- $\mathscr{F}(EWi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))) = \mathscr{F}(\neg C(Ag_1, Ag_2, \varphi)) \wedge E(Ag_{1,2} \wedge \bigcirc\mathscr{F}(C(Ag_1, Ag_2, \varphi))) \wedge E\Diamond\ \neg ACC_{1,2}$,
- $\mathscr{F}(EFu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))) = \mathscr{F}(C(Ag_1, Ag_2, \varphi)) \wedge A(Ag_{1,2} \supset \bigcirc\mathscr{F}(C(Ag_1, Ag_2, \varphi))) \wedge E\Diamond\ ACC_{1,2}$,
- $\mathscr{F}(EVi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))) = \mathscr{F}(C(Ag_1, Ag_2, \varphi)) \wedge A(Ag_{1,2} \supset \bigcirc\mathscr{F}(C(Ag_1, Ag_2, \varphi))) \wedge E(\mathscr{F}(\neg\varphi))$,
- $\mathscr{F}(ERe(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))) = \mathscr{F}(\neg C(Ag_1, Ag_2, \varphi)) \wedge E(Ag_{2,1} \wedge \bigcirc\mathscr{F}(C(Ag_1, Ag_2, \varphi))) \wedge E\Diamond\ \neg ACC_{1,2}$,
- $\mathscr{F}(EDe(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi))) = \mathscr{F}(EWi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))) \wedge \mathscr{F}(C(Ag_3, Ag_2, \varphi))$,
- $\mathscr{F}(EAs(Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi))) = \mathscr{F}(ERe(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))) \wedge \mathscr{F}(C(Ag_1, Ag_3, \varphi))$.

Notice that we formally reduced commitments and their actions into GCTL* formulae without losing their real and concrete meanings in contrast to when those commitments are represented as simple data structures [39] or domain variables [48, 11, 65, 139].

Remark 4 *Since we formally reduced ACTL^{*c} formulae into GCTL* formulae, whether GCTL* would be more expressive than ACTL^{*c} and whether GCTL* would even be used to directly represent and reason about commitments and their actions are pressing questions. As we have modal operators in ACTL^{*c} (e.g., a commitment modality) that cannot be directly expressed in GCTL*, our logic is then more expressive. It is worth noticing that our reduction is in fact a kind of transformation. This means that we transform a formula F_1 in ACTL^{*c} to a formula F_2 in GCTL*, but F_2 is not always transformable to F_1. Moreover, such a transformation is just a technical tool that allows us to model check the formula F_1 in ACTL^{*c} by model checking F_2 in GCTL*, but does not indicate that F_1 and F_2 are equivalent. Thus, transforming a logic to another one using a transformation function does not imply that the two logics have the same expressive power. Using GCTL* to represent commitments will end up by coding commitments as atomic propositions. Consequently, the intrinsic meaning of commitments as a way of communication will disappear. Moreover, no reasoning can be done on* *commitments so that fulfillment, violation and other commitment actions cannot be modeled. This is because those commitment actions can only be coded as atomic action propositions (or atomic propositions) in $GCTL^*$ without having any logical relation with the commitment itself.*

As our reduction is done formally, we can prove its soundness (correctness).

Theorem 1 (Correctness) *Let M and φ be respectively an $ACTL^{*c}$ model and formula and let $\mathscr{F}(M)$ and $\mathscr{F}(\varphi)$ be the corresponding model and formula in $GCTL^*$. We have $M \models \varphi$ iff $\mathscr{F}(M) \models \mathscr{F}(\varphi)$.*

Proof 1

We prove this theorem by induction on the structure of the formula φ:

If φ is a pure CTL^* formula, the correctness is straightforward from the fact that $GCTL^*$ is also an extension of CTL^*.

If φ is not a pure CTL^* formula, by induction over the structure of φ, all the cases are straightforward once the following cases are analyzed.

– $\varphi = C(Ag_1, Ag_2, \psi)$. *We have $M, \langle s\rangle \models C(Ag_1, Ag_2, \psi)$ iff $M, \langle s, \pi\rangle \models \psi$ for every $\pi \in \Pi^s$ such that $\pi \in \mathbb{R}_c(s, Ag_1, Ag_2)$. Consequently, $M, \langle s\rangle \models C(Ag_1, Ag_2, \psi)$ iff $\mathscr{F}(M), \langle s, \pi\rangle \models \mathscr{F}(\psi)$ for every $\pi \in \mathbb{R}_c(s, Ag_1, Ag_2)$, which means for every π where the transitions are labeled by $ACC_{1,2}$. By semantics of A and θ in $GCTL^*$, we obtain $\mathscr{F}(M), \langle s\rangle \models A(\Box\ ACC_{1,2} \supset \mathscr{F}(\psi))$.*

– $\varphi = EWi(Ag_1, Ag_2, C(Ag_1, Ag_2, \psi))$. *We have $M, \langle s_i, \pi\rangle \models EWi(Ag_1, Ag_2, C(Ag_1, Ag_2, \psi))$ iff $M, \langle s_i\rangle \models \neg C(Ag_1, Ag_2, \psi)$ and there exists an accessible state s_j using the accessibility $\approx_{1,2}$ (i.e., $s_i \approx_{1,2} s_j$) where $C(Ag_1, Ag_2, \psi)$ holds and π is not accessible path using $\mathbb{R}_c(s_i, Ag_1, Ag_2)$. By consequently, $M, \langle s_i, \pi\rangle \models EWi(Ag_1, Ag_2, C(Ag_1, Ag_2, \psi))$ iff $\mathscr{F}(M), \langle s_i\rangle \models \mathscr{F}\ (\neg C(Ag_1, Ag_2, \psi))$ and there exists a path such that $(s_i, Ag_{1,2}, s_j) \in \rightarrow$ and $\mathscr{F}(M), \langle s_j\rangle \models \mathscr{F}(C(Ag_1, Ag_2, \psi))$ and there exists a path where $\neg ACC_{1,2}$ holds in some future, which makes the path not accessible. By semantics of E, $\bigcirc$, and θ in $GCTL^*$, we obtain $\mathscr{F}(M), \langle s_i, \pi\rangle \models \mathscr{F}(\neg C(Ag_1, Ag_2, \psi)) \wedge E(Ag_{1,2} \wedge \bigcirc \mathscr{F}(C(Ag_1, Ag_2, \psi))) \wedge E\Diamond\ \neg ACC_{1,2}$.*

– *In a similar way, we can complete the correctness prove of our transformation with respect to the other cases: $\varphi = EFu(Ag_1, Ag_2, C(Ag_1, Ag_2, \psi))$, $\varphi = EVi(Ag_1, Ag_2, C(Ag_1, Ag_2, \psi))$, $\varphi = ERe(Ag_2, Ag_1, C(Ag_1, Ag_2, \psi))$, $\varphi = EDe(Ag_1, Ag_3, C(Ag_1, Ag_2, \psi))$, and $\varphi = EAs(Ag_2, Ag_3, C(Ag_1, Ag_2, \psi))$, so we are done.* ■

Remark 5 *With the result of Theorem 1, whether or not $ACTL^{*c}$ would be sound and complete is an important question. In the one hand, as the two logics (i.e., $ACTL^{*c}$ and $GCTL^*$) have different* *expressive powers, the correctness of our reduction tool cannot be used to infer the soundness and completeness of ACTL^{*c}. As forcefully argued in Remark 4, our reduction is just a model checking tool. On the other hand, there is no work done on proving the theoretical results of soundness and completeness of GCTL*.*

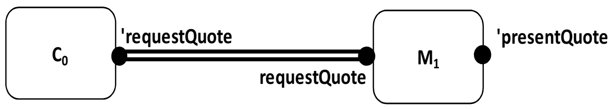
4.4.2 Encoding the NetBill Protocol using CCS

Having defined the reduction process of ACTL^{*c} model and ACTL^{*c} formula, hereafter we present our encoding of the NB protocol, formalized in the model M (cf. Section 4.3.3), by the CCS language. The syntax of the CCS language is given by the following BNF grammar [19]:

$$\mathtt{P} ::= \mathtt{nil} \mid \alpha.\mathtt{P} \mid (\mathtt{P} + \mathtt{P}) \mid (\mathtt{P}||\mathtt{P}) \mid \mathtt{proc}\ \mathcal{C}\mathtt{=P}$$

where `P` refers to the CCS process; the process `nil` means no action whatsoever; if `P` is a process and α is an action prefixing, then α.`P` is a process; if `P1` and `P2` are processes, then so is `P1 + P2` using the choice operator "+"; if `P1` and `P2` are processes, then so is `P1` || `P2` using the parallel composition operator "||"; and the keyword `proc` is used to assign the name $\mathcal{C}$ to the process `P`.

A given model of the NB protocol can be encoded using the language CCS by associating each agent, such as *Cus*, *Mer*, *Cus$_1$*, *Cus$_2$*, *Bank$_1$* and *Bank$_2$* to a set of processes in a recursive manner. Each process represents an agent's local state. Following standard conventions, a CCS process conceptually uses communication channels to receive messages from other processes using input channels and may send out messages after performing actions to other processes using output channels in a complementary fashion. This meaning is graphically illustrated as follows:



where the process C_0 is run in parallel with the process M_1 (i.e., $C_0|M_1$) so that they communicate by sharing the *requestQuote* message. This message is sent by the process C_0 in a complement form (*'requestQuote*) and received by the process M_1 over a communication channel. Recall that communication channels are reliable guaranteeing timely delivery of messages. Internally, commitment, acceptance, failure and recovery states are defined as variables in the `proc` statement. The local actions of each agent are explicitly represented using atomic action propositions in the `proc` statement in order to define the agent's behavior and capture the labeled transitions among commitment states. For example, the customer *Cus* agent can be specified using the CCS language as follows:

proc C_0 ='requestQuote.C_1

proc C_1 =request.presentQuote.C_2

proc C_2 =CC_MerCus_pay_deliver.('acceptQuote.C_3 + ('$Ag_{\mathsf{Cus,Mer}}$.'Release.C_7 + 'null.'Release.C_7))

...

which means the *Cus* agent initially produces the request quote message and evolves into the state C_1. The *Mer* agent replies by sending the present quote message, which makes the *Cus* agent enter into the state C_2. In the state C_2, the *Cus* agent is willing to produce accept quote message and enter into the state C_3 or the state C_7 after performing: (1) the atomic action $Ag_{Cus,Mer}$ and the action release; or (2) the null and release actions, but before doing this (i.e. choosing between C_3 and C_7), it needs to receive the commitment from the *Mer* agent as an offer.

The above encoding of the NB protocol is written in the `.ccs` suffixed file and the following protocol properties are first transformed into GCTL* formulae using our reduction tool and then stored in a separate file whose name includes the `.gctl` suffix.

4.4.3 Protocol Properties

To achieve the flexibility that gives each agent a great freedom and compliance within the same framework, we need to verify the commitment-based protocols against some properties that capture important requirements in MASs. Guerin et al. [71] proposed three types of verification of multi-agent interaction protocols depending on whether the verification process is done at either design time or run time: (1) verify that an agent will always comply; (2) verify compliance by observation; and (3) verify protocol properties. We adopt the third type of verification for three reasons:

1. The desirable properties play an important role in verifying multi-agent interaction protocols [11, 48, 47, 50], which reduces the cost of development process at design time and restricts agents' behaviors by removing bad behaviors without loosing the flexibility.
2. Verifying the compliance of multi-agent interaction protocols with specifications requires adding planner mechanisms equipped with reasoning rules in the code of each agent to reason about its actions to select appropriate ones that satisfy its goals at run time, which can be expensive and may increase the code of the agents [159, 126].
3. Protocol properties have a classification in both reactive and distributed systems to guide designers to check protocol specifications (cf. Section 2.4 in Chapter 2 for more details).

Some proposals have been put forward to classify properties that satisfy different requirements of commitment-based protocols. Yolum [157] verified the correctness of commitment-based protocols at design time with respect to three kinds of generic properties: *effectiveness*, *consistency* and *robustness*. Desai et al. [39] classified protocol properties into two classes: *general properties* such as deadlocks, livelocks and terminations and *protocol-specific properties* to verify commitment-based protocols and their composition. Bentahar et al. [11] classified these properties into: *safety*, *liveness* and *deadlock-freedom*. In this chapter, we specify a rich class of temporal properties: *fairness*, *safety*, *liveness*, and *reachability* using the proposed logic. These properties are similar inspirit to Lamport's properties [88] and could involve Manna and Pnueli's classes [100] widely recognized in reactive and distributed systems. However, the later case is beyond the scope of the thesis. They specifically include the properties introduced in [39] and satisfy the same functionalities of the properties presented in [157]. We will call those properties with functional correctness properties. The differences and similarities of our properties with the properties presented in [39, 157] are explained in Section 4.5. Recall that the formulation of these properties is mainly related to capture some specific properties in the NB protocol.

– Fairness constraint property. The motivation behind this property is to rule out unwanted behaviors of agents and remove any infinite loop in our protocol. An example of unconditional fairness constraint is given by φ_1, which states that along all paths it is not the case that in the future the $Bank_1$ agent globally delegates the commitment to the $Bank_2$ agent.

$$\varphi_1 = \neg E\Diamond\Box\ \neg De(Bank_1, Bank_2, C(Bank_1, Mer, pay))$$

This constraint allows us avoiding situations such as the bank agents delegate the commitment back and forth infinitely often. Thus, by considering fairness constraints, the protocol's fairness paths include only the paths that the interacting agents can follow to satisfy their desired states fairly.

– Safety property. This property means "something bad never happens". In general, it is expressed by $A\Box\ \neg p$ where p characterizes a "bad" situation, which should be avoided. For example, in our protocol a bad situation is: the Cus agent fulfills its commitment by sending the payment, but the Mer agent never commits to deliver the requested goods:

$$\varphi_2 = A\Box\neg\big(EFu(Cus, Mer, C(Cus, Mer, pay))\ \wedge A\Box\ \neg C(Mer, Cus, deliver)\big)$$

– Reachability property. A particular situation can be reached from the initial state via some computation sequences. For example, along a given path, it is eventually the case that there is a possibility for the *Mer* agent to fulfill its commitment by delivering the requested goods:

$$\varphi_3 = E\Diamond\ EFu(Mer, Cus, C(Mer, Cus, deliver))$$

– Liveness property. This property means "something good will eventually happen". For example, along all paths it is globally the case that if the *Cus* agent fulfills its commitment by sending the payment, then in all computations in the future the *Mer* agent will either (1) fulfill the commitment by delivering the goods; (2) withdraw this commitment; or (3) violate it:

$$\begin{aligned}\varphi_4 =& A\Box\big(EFu(Cus, Mer, C(Cus, Mer, pay)) \supset A\Diamond\big[EFu(Mer, Cus, C(Mer, Cus, deliver))\\ &\vee EWi(Mer, Cus, C(Mer, Cus, deliver))\\ &\vee EVi(Mer, Cus, C(Mer, Cus, deliver))\big]\big)\end{aligned}$$

The above temporal properties can be generalized to verify the conformance of other multi-agent interaction protocols as they have a corresponding classification in distributed systems to guide protocol designers to check protocol specifications. They are also defined from a general perspective, for example a formula having a future operator and "something good" with respect to the protocol specification can be used to define liveness property. Notice that our reachability property do the same function of reachability analysis in verifying the traditional protocol design errors in distributed systems [76]. The idea of reachability analysis is to generate a reachability graph (tree) by recursively exploring all possible transitions that lead to new state, given an initial state. Such a reachability graph is used by some tools—some of them are combined within model checking approach—for checking some design errors such as *deadlock* and *livelock*.

4.4.4 Experimental Results

We implemented our technique including a reduction tool (Com2Cwb)—that automatically transforms the problem of model checking ACTL^{*c} into the problem of model checking GCTL*—on top of the CWB-NC model checker and provided a thorough assessment of this reduction tool by using two case studies: (1) our extended version of the NB protocol; and (2) the Contract Net protocol (CN). The CN protocol is designed from online business point of view to reach agreements among interacting agents. These case studies were performed on a laptop equipped with the Intel(R) Core(TM) i5-2430M clocked at 2.4GHz processor and 6GB memory running under x86_64 Windows 7.

A) Verifying the Extended NB Protocol

Having respectively presented our extended version of the NB protocol and its encoding using the CCS process algebra language in Sections 4.3.3 and 4.4.2, thereafter we present its automatic verification using the CWB-NC model checker against the safety and reachability properties (cf. Section 4.4.3) as well as the deadlock property. The deadlock property mainly checks that all the fired transitions are labeled with agents and their actions. Formally, $E\Diamond\neg\{-\}$ [19]. In particular, we conducted 7 experiments, which are reported in Table 8 wherein the number of agents (n); reachable states (#States); transitions (#Trans) and the total time in seconds (i.e., the time for building the model (TBM) plus the time for checking deadlock (TDL), safety (TSF), and reachability (TRE) formulae) are given. These experiments start with a simple business scenario between two agents (*Cus* and *Mer*). This scenario initially begins by the *Cus* agent requests a quote for some goods and the *Mer* agent commits to deliver them after it received the payment where each one of them has possibilities to withdraw and violate its commitment. Thereafter, we start to give the *Cus* agent another possibility to delegate its commitment to the $Bank_1$ agent in experiment 2, which for some reasons delegates its commitment to the $Bank_2$ agent in experiment 3. In experiments 4 and 5, the *Cus* agent can assign its commitment to the Cus_1 agent, which for some reasons assigns its commitment to the Cus_2 agent. By experiment 5, the 6 participating agents in the extended NB protocol are completely modeled.

Table 8: Verification results of the NB protocol

n	#States	#Trans	TBM(sec)	TDL(sec)	TSF(sec)	TRE(sec)	Total Time(sec)
2	267	851	0.024	0.077	0.064	0.025	0.190
3	883	3878	0.123	0.336	0.254	0.033	0.746
4	5293	30206	1.160	3.154	2.232	0.089	6.635
5	18145	121965	6.031	15.227	10.569	1.033	32.860
6	108865	874916	56.194	138.253	92.316	0.171	286.932
7	254017	2302590	207.231	478.866	312.025	0.190	1684.409
8	428545	4288876	503.171	1119.284	687.520	0.235	5123.356

We underline that when the number of reachable states (which reflect the state space of the model) is small, the total time is also small (cf. Exp.1 and Exp.2). However, when the number of reachable states increases (as in Exp.4 and Exp.5), the total time is going to be much higher. In experiment 6, we turned toward showing the effectiveness of our technique in terms of the total time and number of reachable states by making more than one customer request a quote for some goods. For instance, in experiment 6, we have two customers and each one of them can asynchronously communicate with five agents (2 Bank agents and 3 Merchant agents as we mentioned), so we have 7 participating agents. Also, in experiment 7, we have 3 customers, 2 Bank agents, and 3 Merchant agents, so the total number of participating agents is 8.

B) Verifying the CN Protocol

The CN protocol is already used to show how commitments and their actions can specify protocols in business settings [157, 50]. It starts with a manager requesting proposals for a particular task. Each participant either sends a proposal or a reject message. The manager accepts only one proposal among the received proposals and explicitly rejects the rest. The participant with the accepted proposal informs the manager with the proposal result or the failure of the proposal. As in our modeling of the extended NB protocol, we associate the formal model M with the CN protocol and then use the Büchi automaton $\mathcal{B}$ defined above to compile this protocol. The acceptance state is when the participant agent fulfills its commitment by sending result to the manager.

Table 9 reports the verification results of the CN protocol against the three properties (safety, reachability and deadlock) using our transformation method and the CWB-NC model checker on four experiments wherein the number of agents (n), the number of reachable states (#States), the number of transitions (#Trans) and the total time in seconds (as in Table 8) are given. In the first experiment, we have three agents: the manager agent and two participating agents wherein the first participant can delegate its commitment to the second participant. In experiments 2, 3 and 4, we have one manager agent and respectively 5, 7 and 9 participant agents. It is obvious that when the state space of the model increases, the total time is also going to be much higher.

Table 9: Verification results of the CN protocol

n	#States	#Trans	TBM(sec)	TDL(sec)	TSF(sec)	TRE(sec)	Total Time(sec)
3	721	3040	0.093	0.030	0.198	0.015	0.336
5	1711	8308	0.325	0.027	0.569	0.018	0.939
7	2983	15812	0.784	0.029	1.268	0.021	2.102
9	4537	25552	1.627	0.035	2.411	0.030	4.103

Notice that the three tested formulae are valid (cf. Tables 8 and 9) and the reachability and deadlock formulae show that the CWB-NC automata-based model checking is very powerful in such cases. However, since the safety formula contains a universal path quantifier, it must be performed on the whole model, thereby invalidating the on-the-fly technique implemented in the CWB-NC model checker, which builds the intersection of the two *Alternating Büchi Automata Tableau Automata*

(ABTA) of the model and the formula to be checked on demand without exploring the whole model [19]. From our perspective, this is the main cause of a longer time of verification for the given properties having this form (cf. Tables 8 and 9). We can conclude with the proposed technique is efficient and effective in terms of the number of agents and reachable states. These results prove the effectiveness of our reduction tool that allows checking the satisfiability of commitments and their actions as temporal modal connectives and not as domain variables.

4.5 Discussion and Relevant Work

The novelty of our approach lies in presenting: (1) a new semantics for commitments and all associated actions; (2) a new specification language of commitment-based protocols; (3) a new symbolic model checking algorithm for ACTL^{*c} (cf. Appendix A); and (4) a new technique for reducing the above language without missing or restricting real semantics and verifying commitment-based protocols. Using two business protocols, we have experimentally evaluated the effectiveness of our reduction tool and technique along with our verification approach implemented using the CWB-NC model checker. These experiments also paint the following picture: the model checker was able to verify a variety of complex formulae correctly and efficiently. Our approach is clearly not exhaustive but helps protocol designers eliminate bad behaviors. This approach establishes the usability of the approach by applying it to a large real-life business processes (approximately $4.2e + 06$ states). The overall conclusion coincides with the usual considerations in that automatic verification methods complement other static verification methods very well. When comparing our approach to other approaches in the literature, we find that our approach considerably simplifies the specifications to be checked and maintains the feasibility of the model checking approach.

The protocol properties introduced by Desai et al. [39] are specified using LTL and classified into *general properties* such as deadlocks and livelocks and *protocol-specific properties* to check a specific behavior of interacting agents. The SPIN model checker checks deadlocks and livelocks, given end states, which is unlike our deadlock property. Technically, their deadlock properties can result from the contradiction among axioms used to compose protocols. An example of protocol specific properties is introduced in Chapter 3. In our approach, the protocol specific properties can be successfully defined using safety and liveness properties, which have an original and standard classification in distributed systems. However, the authors do not consider fairness and reachability properties. Gerard and Singh [65] used CTL to define a set of properties to verify protocols refinement using the MCMAS model checker. Compared to these approaches [39, 65], our specification language

ACTL^{*c} is more expressive and compact than LTL and CTL adopted in [39, 65].

Yolum [157] presented three generic properties (*effectiveness, consistency* and *robustness*) that are required to develop commitment-based protocols at design time. The effectiveness property is mainly related to check if a given protocol effectively progresses. The consistency property checks that the execution of protocol does not produce conflicting computations. A protocol is robust when the intended tasks can be satisfied by different alternative paths. Our properties meet these requirements in the sense that the reachability and deadlock-freedom can be used to satisfy the same objective of the effectiveness property. The consistency property is achieved in our protocol by satisfying the safety property. Moreover, the robustness property is satisfied by considering the liveness property and fairness paths accompanied with recovery states that capture the protocol failures, such as if the *Mer* agent withdraws or violates its commitment, then it must refund the payment to the *Cus* agent. Finally, the proposed approach enhances Yolum's semi-automatic verification with full-automatic verification using model checking.

The tool CWB-NC is used in [11] to verify the NB protocol without delegation and assignment actions, but exploiting a different encoding of the example and considering only two agents. This encoding models each agent by describing its possible actions and each action is described by a set of states, e.g., *Withdraw* action needs 2 states. There is also a separate process that models the communication channels between two agents. Such an encoding is less efficient than the one presented here, in that the internal states increase the agent model and separated communication channel repeats various parameters for the protocol. While the encoding proposed in [11] allowed for the verification of 2 agents only, the experimental results shown in Table 8 report scenarios with up to 8 agents. The differences with the other proposals can be also discussed at the level of the chapter introduction and motivation. We also presented some comparisons against some proposals through the chapter. While [39, 65] and [11, 47] show that the model checking of commitment-based protocols is feasible, in this chapter we focused on formal reduction technique, correctness, efficiency and expressiveness considerations.

In a previous work [50], we presented an extension of CTL with commitment modality to formally specify commitment-based protocols. We reduced the problem of model checking the underlying logic into the problems of model checking CTLK and ARCTL in order to use the MCMAS and extended version of NuSMV model checkers respectively. Our previous specification language is less expressive than our current language. Furthermore, we used a symbolic model checking-based technique, which is different from the technique we introduced in this chapter. In terms of reduction method, we only reduced commitment modality without considering commitment actions.

To evaluate our semantics, it meets all Singh's crucial criteria (formal based on an expressive logic, meaningful, declarative and verifiable using full and automatic model checking approach). It is also property-based in terms of formally defined protocol properties. Thus, we have a clear, intuitive and computational semantics for ACL messages. However, whether the advocated model checking technique would be efficient and whether the language semantics would even be computationally grounded are the rasing questions. As the proved space complexity of the developed model checking algorithm for ACTL^{*c} is PSPASE-complete (cf. Appendix A) and the proved time complexity of model checking its CTL* fragment is exponential in the length of the formula and linear in the size of the model [31, 33] (cf. Table 2 in Chapter 2), the proposed model checking technique is computationally complex, due to the fact that being expressive probably means being computationally complex. To develop an efficient model checking technique, a balance between expressiveness and verification efficiency aspects should be considered. For these reasons, we adopt a refined fragment of ACTL^{*c}, called CTLC in Chapter 5 because the time complexity of model checking CTL is linear in both the length of the formula and model [32, 33], whereas its space complexity is PSPACE-complete [86] with respect to concurrent programs (cf. Table 2 in Chapter 2).

As in modal logic, the proposed semantics is given with respect to Kripke-like models, which allow us to remain silent with regard to the internal structure of interacting agents. Woorldridge [151] and Jamroga and Ågotnes [79] pointed out that Kripke semantics are not *computationally grounded*. Also, general Kripke structures do not always give enough help at design and implementation stages [79]. Such a computationally grounded semantics means formulae should be interpreted directly with respect to a computational model. The semantics of social commitment logic is given in terms of accessible paths, but these accessible paths do not have a direct and concrete interpretation in terms of the states of interacting agents; instead, we used an agency function, which checks for the existence of two interacting agents at all states along the accessible path. However, this makes the semantics semi-computationally grounded, which is not enough to say that social commitment formula crucially represents the properties of those agents. A property of agents means that agents' computations can be understood as models for social commitment logic and in this sense, social commitment logic is computationally grounded. In Chapter 5, we make the semantics full-computationally grounded by considering local states of the two interacting agents in the definition of social accessibility relation that we use to define the semantics of commitments. Moreover, we adopt the formalism of interpreted systems introduced by Fagin et al. [59] to model a system of interacting agents.

Chapter 5

On the Reduction of Model Checking Commitments

In this chapter[1], we focus on a refined fragment of ACTL^{*c}, called CTLC, to balance between expressiveness and verification efficiency. Our ultimate aim is to define a fully computationally grounded semantics for commitments and their fulfillment. We extend the interpreted system formalism introduced by Fagin et al. [59] to model MASs with shared and unshared variables to account for agent communication. We present axioms of commitment and fulfillment modalities. With those theoretical foundations, we advocate a formal reduction method to transform the problem of model checking CTLC into the problem of model checking ARCTL [104] and the problem of model checking GCTL* [18, 19] in order to be able to respectively use the extended NuSMV symbolic model checker and the CWB-NC automata model checker as a benchmark. We also prove that our reduction methods are sound. To illustrate the effectiveness of the proposed reduction methods, we provide two business case studies along with their respective implementations and experimental results

5.1 Introduction

In Chapter 4, by extending CTL* with commitment and action modalities, we successfully painted the following picture: the existing model checkers are feasible to verify the modalities of commitments and their actions correctly without losing the intrinsic meaning. With this result, we go forward to consider other interesting issues such as efficiency of model checking, computational grounding

[1]The results of this chapter have been published in the Journal of Autonomous Agent Multi-Agent Systems [51].

definition, and determining effective model checking techniques. However, an argument against the use of CTL * is that its model checking algorithm is exponential in the length of the formula, which is computationally intractable to automate, especially when the length of the formula is very large, in opposite to that, CTL model checking algorithm is linear in the length of the formula (cf. Table 2 in Chapter 2). On the other hand, it is argued that in practice, space efficiency is more important than time efficiency [19, 32, 86, 132]: an agent is quite prepared to wait 10 minutes for a model checker to finish executing, but an agent learns nothing if after 10 seconds a model checker aborts due to a lack of memory. Also, when the space complexity of two model checking algorithms is computationally equal, but one of them performs faster than the other, then one can advocate the faster one. Besides CTL model checking being fast, the standard classes of safety properties can be expressed in CTL, which are used frequently and regularly. CTL formulae can be used to show the absence of deadlock or livelock in reactive and distributed systems. Interestingly, most model checkers available today are tailored for CTL, such as NuSMV and MCMAS. It must be noted here that CTL model checkers, specifically those based on OBDDs, can perform LTL model checking too, but it is known that model checking LTL is like CTL* exponential in the length of the formula. The model checking community tends to favor CTL model checkers [32, 33, 102]. For these reasons, we adopt a branching fragment of ACTL^{*c}, called CTLC, i.e., we only consider the formulae that can be expressed both in ACTL^{*c} and CTLC with some refinements. More precisely, CTLC is an extension of CTL [31] with modalities for reasoning about commitments and their fulfillment. Thus, CTLC is expressive enough to represent and reason about the deontic notion of commitments. The objective of this fragment is to provide a balance between expressiveness and verification efficiency.

Following the definition of computational grounding theory introduced in [151], computationally grounded semantics for commitments and their fulfillment is not simply a desirable aspect that we can choose to not consider. As when a logic of commitment is not computationally grounded, then this open doubt on claiming that such a logic can be useful and fruitful for reasoning about computational MASs. Moreover, when we are treating agents' commitments as formal specifications for computational MASs, their computationally grounded semantics is an issue that must be addressed.

With respect to model checking, we advocate in the present chapter a reduction method because it is not only easy to implement, but also allows us to compare different verification techniques with respect to the same logic. Our reduction method preserves the truth value of all formulae, and we prove that it is sound (correct). The validity of the reduction method is then tested automatically.

The motivation of this chapter is to overcome the above challenges by addressing the three parts discussed in Figure 7 in Chapter 4: logic (part 1), reduction (part 2) and implementation (part 3).

In the first part, we introduce CTLC, which is a new branching-time temporal logic of commitments including CTL temporal modalities and modalities for commitments and their fulfillment using the formalism of interpreted systems. This formalism was introduced by Fagin et al. [59] to model MASs and applied by many as a prime example of computationally grounded models of distributed systems. We here extend this formalism with shared and unshared variables as part of agents' local states to account for the intuition that social commitments are conveyed through communication between agents. Concretely, these variables and the local components of interacting agents are used to define the social accessibility relation, which defines a full-computationally grounded semantics for commitments and their fulfillment. We notice that this communication aspect has not been considered in Chapter 4 wherein the semantics is only semi-computationally grounded. In fact, communication between agents is considered as the first-class entity in our approach, which means commitments are made through communication and for communication purposes and not via blackboard or implicitly. So, social commitments in our approach can be called *Communicative Commitments*. In this part, we: (1) consider the relationship between commitments and their fulfillment, which is still missing in all the existing approaches; (2) remove the violation modality as used in Chapter 4; instead we show how to express violation as property in the proposed logic; and (3) present the valid axioms of commitment and fulfillment modalities.

In the second part, we reduce the problem of model checking CTLC into: (1) the problem of model checking ARCTL, the combination of CTL with action formulae [104]; and (2) the problem of model checking GCTL*. The first reduction allows a direct use of the extended version of the NuSMV model checker introduced in [90]. The second reduction makes using the CWB-NC model checker possible. Two reasons have motivated the election of the extended NuSMV and CWB-NC tools: (1) their models are characterized by labeled transitions with actions and during the transformation of our model, these labeled transitions are used to capture the accessibility relations; and (2) they use different model checking techniques namely, *Ordered Binary Decision Diagrams* (OBDDs) implemented in extended NuSMV and *Alternating Büchi Tableau Automata* (ABTA) implemented in CWB-NC, which gives us the possibility to compare these two techniques with respect to the verification of commitments and their fulfillment.

To check the effectiveness of our approach—in the third part—we implement our reduction tools on top of the model checkers (extended NuSMV and CWB-NC) and then report the experimental results of verifying two case studies—widely used to clarify commitment-based protocols: (1) the NetBill protocol; and (2) the Contract Net protocol against some desirable properties expressed in our logic.

The remainder of this chapter is organized as follows. In Section 5.2, we briefly summarize the formalism of interpreted systems [59] and define the syntax and semantics of CTLC. We also present the valid axioms of commitment and fulfillment modalities. Reducing the problem of model checking CTLC into the problem of model checking ARCTL and GCTL* and the theorems that prove the soundness of our reduction techniques are presented in Section 5.3. In Section 5.4, we present two case studies widely studied in the literature of agent communication, the NetBill protocol and the Contract Net protocol along with the implementation of our reduction techniques on top of extended NuSMV and CWB-NC. Section 5.5 ends the chapter by a discussion about comparing our approach with other approaches and identifying the motivations of developing a dedicated symbolic model checking algorithm for CTLC in Chapter 6.

5.2 Interpreted Systems and CTLC

In this section, we present the logic part of our approach, which is mainly related to extend the formalism of interpreted systems and present the branching-time temporal logic, CTLC, and axioms of commitment and fulfillment modalities.

5.2.1 Interpreted Systems

The formalism of interpreted systems was potentially investigated by Fagin et al. [59]. This formalism provides a mainstream framework for modeling, reasoning and systematically exploring fundamental classes of MASs, such as synchronous and asynchronous. We advocate this formalism for many reasons summarized as follows:

1. It allows us to abstract from the details of the components and to focus only on modeling key characteristics of the agents and the evolution of their social commitments, which is missing in existing agent communication models.
2. It is a useful tool for ascribing autonomous [91, 92] and social behaviors of interacting agents within open MASs [52].
3. It supports the interoperability between global (system) and local (agent) models [52, 59].

The formalism of interpreted systems provides a useful framework to locally model autonomous and heterogeneous agents who interoperate within a global system via sending and receiving messages. The concept of local states offers a flexible abstraction for the agents. Technically, local states can be singletons, corresponding to a very high-level description of the agents. However, local states are allowed to have a more complex structure. For instance, local states could be a combination of singletons and a set of variables as we will show later in this chapter.

Interpreted systems can be defined as follows [59]. Consider a set $\mathcal{A} = \{1, \dots, n\}$ of n agents in which at any given time each agent in the system is in a particular local state. Intuitively, each local state of an agent represents the complete information about the system that the agent has at its disposal at a given moment. We associate a nonempty, countable and private set L_i of local states for each agent $i \in \mathcal{A}$. To account for the temporal evolution of the system, the formalism of interpreted systems associates with each agent i the finite set Act_i of possible local actions that the agent is allowed to perform. Differently from L_i, Act_i is "public". As in interleaved interpreted systems [91], to model synchronous communication among interacting agents, it is assumed that $null \in Act_i$ for each agent i where $null$ refers to the silence action (i.e., the fact of doing nothing). The action selection mechanism is given by the notion of local protocol $\mathcal{P}_i : L_i \to 2^{Act_i}$ for each $i \in \mathcal{A}$. That is $\mathcal{P}_i$ is a function giving the set of enabled actions that may be performed by i in a given local state. Notice that this definition may enable more than one action to be performed for a given local state. Also, the local protocol enables an agent to consider its preferable policies or strategies at its local states, which is functionally similar to the conversation policies in ACLs. The agents act within an "environment" (e), which can be also modeled with a set of local states L_e, a set of local actions Act_e, and a local protocol $\mathcal{P}_e$. This can be seen as a special agent that can capture any information that may not pertain to a specific agent. The local states for e are "public", i.e., all the remaining agents may access. For the sake of simplicity purposes, we remove the environment agent from the interpreted system formalism as done in [90]. As in [59], we represent the instantaneous configuration of all agents in the system at a given time via the notion of global state.

Definition 6 ([59]) *Let G be the set of all global states and let $g = (l_1, \dots, l_n)$ be a global state, i.e., $g \in G$ where each element $l_i \in L_i$ represents a local state of agent i, thus the set of all global states $G = L_1 \times \ldots \times L_n$ is the Cartesian product of all local states of n agents.*

We use the notation $l_i(g)$ to represent the local state of agent i in the global state g. $I \subseteq G$ is a set of initial global states for the system. As in [59], the global evolution (or transition) function is defined as follows: $\tau : G \times ACT \to G$ where $ACT = Act_1 \times \ldots \times Act_n$ and each component $a \in ACT$ is a "joint action", which is a tuple of actions (one for each agent). In addition, the local evolution function τ_i determines the transitions for an individual agent i between its local states. It is defined as follows: $\tau_i : L_i \times Act_i \to L_i$ where if $\tau_i(l_i(g), null) = l_i(g')$ for two given global states g and g', then $l_i(g) = l_i(g')$. It easy to see that such a formalism provides a modular representation [79], in the sense that components can be replaced, removed or added, with little modifications to the whole representation.

In this chapter, we extend the interpreted system formalism to account for communication that occurs during the execution of MAS. This extension allows us to provide an intuitive and full-computationally grounded semantics for social commitments that are established through communication between interacting agents. Thus, associated with each agent $i \in \mathcal{A}$ is a set Var_i of n local Boolean variables, i.e., $|Var_i| = n$, which are used to represent communication channels (each agent has one communication channel with each agent) through which messages are sent and received. Each local state $l_i \in L_i$ of agent i is associated with different values obtained from different assignments to variables in Var_i. We denote the value of a variable x in the set Var_i at local state $l_i(g)$ by $l_i^x(g)$. Thus, if $l_i(g) = l_i(g')$, then $l_i^x(g) = l_i^x(g')$ for all $x \in Var_i$. The idea is that, as in distributed systems, for two agents i and j to communicate, they should share a communication channel, which is represented by a shared variable between i and j. In this perspective, a communication channel between i and j does exist iff $\exists! x \in Var_i \cap Var_j$, which means $|Var_i \cap Var_j| = 1$ (i.e., at least one variable is shared between the two interacting agents). For the Boolean variable $x \in Var_i \cap Var_j$, $l_i^x(g) = l_j^x(g')$ means the values of x in $l_i(g)$ for i and in $l_j(g')$ for j are the same. This intuitively represents the existence of a communication channel between i (in g) and j (in g') through which the value of Boolean variable x has been sent by one of the two agents to the other, and as a consequence of this communication, i and j will have the same value for this variable (cf. Figure 12). It is worth noticing that shared variables only motivate the existence of communication channels, not the establishment of communication. This aspect is addressed next.

The semantics of our modal language is interpreted using a model generated from the interpreted system formalism. In fact, this model, as in [59], moves away from Kripke models while still benefiting from most of its technical apparatus.

Definition 7 (Models) *A model $\mathcal{M}_C = (S, I, R_t, \{\sim_{i \to j} \mid (i,j) \in \mathcal{A}^2\}, \mathcal{V})$ that belongs to the set of all models $\mathbb{M}$ is a tuple where*

- *$S \subseteq L_1 \times \ldots \times L_n$ is a set of global states for the system in which it contains only states that are reachable from I by means of R_t,*
- *$I \subseteq S$ is a set of initial global states for the system,*
- *$R_t \subseteq S \times S$ is a transition relation defined by $(s, s') \in R_t$ iff there exists a joint action $(a_1, \ldots, a_n) \in ACT$ such that $\tau(s, a_1, \ldots, a_n) = s'$,*

– *For each pair $(i,j) \in \mathcal{A}^2$, $\sim_{i\to j} \subseteq S \times S$ is a social accessibility relation defined by $s \sim_{i\to j} s'$ iff (1) $l_i(s) = l_i(s')$ and (2) $\exists! x \in Var_i \cap Var_j$ such that $l_i^x(s) = l_j^x(s')$ and $\forall y \in Var_j - Var_i$ we have $l_j^y(s) = l_j^y(s')$. We also assume that for any pair $i,j \in \mathcal{A}$, we have that for any $s \in S$, the set $\sim_{i\to j}(s)$ is not equal to the empty set wherein $\sim_{i\to j}(s)$ is the set of accessible states from s, i.e., $\sim_{i\to j}(s) = \{s' \in S \mid s \sim_{i\to j} s'\}$, and*

– *$\mathcal{V} : S \to 2^{\mathcal{PV}}$ is a valuation function where $\mathcal{PV}$ is a set of atomic propositions.*

The social accessibility relation $\sim_{i\to j}$ from a global state s to another global state s' ($s \sim_{i\to j} s'$) captures the intuition that there is a communication channel between i and j ($\exists! x \in Var_i \cap Var_j$) and s' is a resulting state from this communication initiated by i at s. The channel is thus filled in by i in s, and in s' j receives the information (i.e., the channel's content), which makes the value of the shared variable the same for i and j ($l_i^x(s) = l_j^x(s')$). As i is the agent who intentionally initiates the communication, the source and target (or resulting) states s and s' are indistinguishable for i ($l_i(s) = l_i(s')$) as i is not learning any new information. And as j is the agent who receives the communication, s and s' are indistinguishable with regard to the variables that have not been communicated by i, i.e. unshared variables ($l_j^y(s) = l_j^y(s')$ $\forall y \in Var_j - Var_i$). Finally, as our focus in this thesis is agent communication, we assume the existence of communication channels between any two agents in any state, which motivates the assumption that $\sim_{i\to j}(s)$ has at least one accessible state. This social accessibility relation formalized in terms of relations over the states of a model differs from the ones presented in [52, 50] in terms of considering shared and unshared variables. Figure 12 illustrates an example of establishing a communication channel among two global states. In this example, two agents are communicating and the shared and unshared variables for those agents are as follows. Agent i: $Var_i = \{x_1, x_2\}$. Agent j: $Var_j = \{x_1, x_3\}$. In the figure x_1 is the shared variable (i.e., it represents the communication channel between i and j), and x_3 is a j's

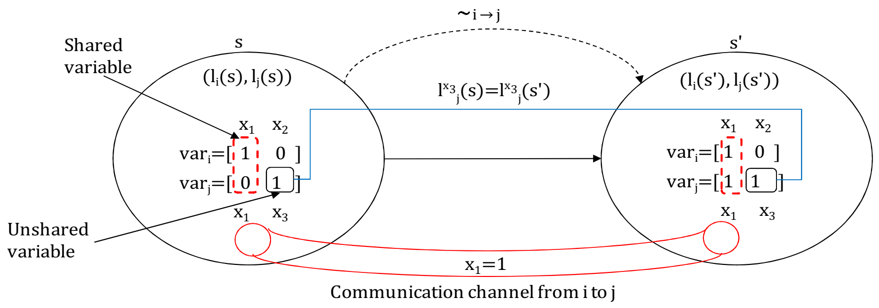


Figure 12: An example of social accessibility relation $\sim_{i\to j}$.

variable unshared with i. Notice that the value of the variable x_1 for j in the state s' is changed to be equal to the value of this variable for agent i, which illustrates the message passing through the channel. However, the value of the unshared variable x_3 is unchanged.

It is worth to mention that shared variables are used solely to model communication channels to define the social accessibility relation, which in turn will be used to define the semantics of commitment, meaning that shared variables are entirely independent or unrelated to commitment content. In fact, our modeling can be seen as an abstraction of *message-passing systems* described in [59]. Specifically, in message-passing systems, process's local state contains information including internal actions that can change the value of a variable and the messages that it has sent and received. So each agent can directly control and manage communication channels by means of its actions. Furthermore, our extension of the interpreted system formalism by using variables for communication is, to some extent, related to the *modular interpreted systems* approach proposed in [79] where variables are also used for communication purposes. Specifically, modular interpreted systems include a component $\mathcal{I}n$, which represents interaction alphabet and two interaction functions: (1) out_i that illustrates the influence that an action of an agent i may have on the external world; and (2) in_i that illustrates the influences of other agents on the agent i depending on its local state. However, unlike modular interpreted systems, our extension is not meant to focus on the influences of agents through interaction, but on the existence of a communication channel through which two agents can communicate. While communication is the first-class citizen in our perspective, the direct interactions are modeled by means of social commitments.

Modeling complex and open systems such as MASs using the formalism of interpreted systems is typically conducted by using logic-based formalisms as formal tools to express desirable properties [59, 91, 92].

5.2.2 CTLC

The syntax of CTLC, which is a combination of branching time CTL introduced by Clarke et al. [31] with modalities for social commitments and their fulfillment, is defined as follows:

Definition 8 (Syntax of CTLC)

$$\varphi ::= \ p \mid \neg\varphi \mid \varphi \vee \varphi \mid E \bigcirc \varphi \mid E\Box\varphi \mid E(\varphi\ U\ \varphi) \mid C_{i\to j}\varphi \mid Fu(C_{i\to j}\varphi)$$

where:

– $p \in \Phi_p$ *is an atomic proposition;*

– *E is the existential quantifier on paths;*

– *$\bigcirc$, $\Box$, and U are CTL path modal connectives standing for "next time", "globally", and "until" respectively;*

– *The Boolean connectives $\neg$ and $\vee$ are defined in the usual way; and*

– *The modal connectives $C_{i\to j}$ and Fu stand for "commitment" and "fulfillment of commitment" respectively.*

In this logic, $C_{i\to j}\varphi$ is read as "agent i commits towards agent j that φ", or equivalently from communication perspective as "i is conveying information φ to j", or simply as "φ is committed to" when i and j are understood from the context. $Fu(C_{i\to j}\varphi)$ is read as "the commitment $C_{i\to j}\varphi$ is fulfilled". Other temporal modalities can be defined in terms of the above as usual. For example, $E\Diamond\varphi \triangleq E(\top\ U\varphi)$, $A\bigcirc\varphi \triangleq \neg E\bigcirc\neg\varphi$, and $A\Box\varphi \triangleq \neg E\Box\neg\varphi$ where $\Diamond$ and $\top$ stand for eventually (future) and propositional constant true respectively. We use $\widehat{C}_{i\to j}$ as a modality dual to $C_{i\to j}$. It stands for "possibility of committing" and abbreviates as $\widehat{C}_{i\to j}\varphi \triangleq \neg C_{i\to j}\neg\varphi$. We read $\widehat{C}_{i\to j}\varphi$ as "agent i considers it possible to commit towards agent j that φ" or also "φ is compatible with what i commits to towards j".

Computation paths. A computation path $\pi = (s_0, s_1, \ldots)$ in $\mathcal{M}_C$ is an infinite sequence of global states in S such that for all $i \geq 0, (s_i, s_{i+1}) \in R_t$. $\pi(k)$ is the k-th global state of the path π. The set of all paths is denoted by Π, whilst $\Pi(s_i)$ is the set of all paths starting at the given state $s_i \in S$.

Definition 9 (Satisfaction) *Given the model $\mathcal{M}_C$, the satisfaction of a CTLC formula φ in a global state s, denoted by $(\mathcal{M}_C, s) \models \varphi$ is recursively defined as follows:*

– $(\mathcal{M}_C, s) \models p$ *iff* $p \in \mathcal{V}(s)$,

– $(\mathcal{M}_C, s) \models \neg\varphi$ *iff* $(\mathcal{M}_C, s) \nvDash \varphi$,

– $(\mathcal{M}_C, s) \models \varphi \vee \psi$ *iff* $(\mathcal{M}_C, s) \models \varphi$ *or* $(\mathcal{M}_C, s) \models \psi$,

– $(\mathcal{M}_C, s) \models E\bigcirc\varphi$ *iff there exists a path π starting at s such that* $(\mathcal{M}_C, \pi(1)) \models \varphi$,

– $(\mathcal{M}_C, s) \models E\Box\varphi$ *iff there exists a path π starting at s such that* $(\mathcal{M}_C, \pi(k)) \models \varphi$ *for all* $k \geq 0$,

– $(\mathcal{M}_C, s) \models E(\varphi\ U\ \psi)$ *iff there exists a path π starting at s such that for some* $k \geq 0$, $(\mathcal{M}_C, \pi(k)) \models \psi$ *and* $(\mathcal{M}_C, \pi(j)) \models \varphi$ *for all* $0 \leq j < k$,

– $(\mathcal{M}_C, s) \models C_{i\to j}\varphi$ *iff for all global states $s' \in S$ such that $s \sim_{i\to j} s'$, we have* $(\mathcal{M}_C, s') \models \varphi$,

– $(\mathcal{M}_C, s) \models Fu(C_{i\to j}\varphi)$ *iff there exists* $s' \in S$ *such that* $(\mathcal{M}_C, s') \models C_{i\to j}\varphi$ *and* $s' \sim_{i\to j} s$.

For the propositions, Boolean connectives, and temporal modalities, the relation $\models$ is defined in the standard manner.

The state formula $C_{i\to j}\varphi$ is satisfied in the model $\mathcal{M}_C$ at s iff the content φ holds in every accessible state obtained by the accessibility relation $\sim_{i\to j}$. In the specific context of agent communication, which is the focus of this thesis, when i commits towards j that φ, $\sim_{i\to j}$ captures the intuition that for a commitment to take place, a communication channel should exist between the communicating agents (shared variables), and the accessible state s' (where φ holds) is indistinguishable from the current state s for i (which reflects the persistence of i towards its commitment) as i is the agent who is committing; however, for j who is receiving the commitment, the two states are different as new information is obtained from i through the communication channel and this is why in the accessible state, j has the same value as i has for the shared variable (i.e., the content of the communication channel). Furthermore, the accessible state is not completely different from the current state for j as some information is still the same, and this is why for the unshared variables, the current and accessible states for j are indistinguishable (cf. Figure 12). As mentioned earlier, there is no relation between the content φ and the local Boolean variables (shared and unshared), which are only used to define the accessibility relation. Notably, this semantics is full-computationally grounded because it has a direct interpretation in terms of the local states of interacting agents and represents the property of interacting agents (cf. Section 4.5 in Chapter 4).

The state formula $Fu(C_{i\to j}\varphi)$ is satisfied in the model $\mathcal{M}_C$ at s iff there exists a state s' satisfying the commitment (called here the *commitment state*) from which the current state (i.e., s) is "seen" via the accessability relation $\sim_{i\to j}$. The idea behind this semantics is to say that a commitment is fulfilled when we reach an accessible state from the commitment state. The commitment is fulfilled because its content holds in this accessible state. Although the semantics of fulfillment proposed in Chapter 4 uses not only accessible states but also accessible paths, the two semantics have the same intuition. However, the new semantics differs from the one proposed in [11, 48, 50, 52] in which the current state s should not only be accessible but also reachable from the commitment state s'. In our semantics, the reachability condition is omitted. In fact, being reachable from the commitment state is not a part of the meaning of fulfilling a commitment, but a condition that can be checked as a property as follows:

$$A\Box(\neg E(\neg C_{i\to j}\varphi \; U \; (\neg C_{i\to j}\varphi \wedge Fu(C_{i\to j}\varphi))))$$

The property is a condition saying that in all states of every computation, it is not the case that there is a computation where fulfilling a commitment occurs in its future without such a commitment has been established before. What is more interesting is that this property, which guarantees that a commitment cannot be fulfilled without being active (i.e., established or created), is satisfied in every model, so valid (the validity proof is given in the next section). The main advantage of having a property that can be checked (or proved as validity) instead of adding it as a part of the semantics is to simplify such a semantics, which means making its model checking simpler. This is extremely important as far as time and space complexity of model checking is an issue.

Example 3 *Let us assume the models depicted in the Figure 13. The state s_1 in $\mathcal{M}_{C1}$ is labeled with the commitment from i to j to bring about $E\bigcirc p$ because: (1) there is only one accessible state s_2; and (2) the formula $E\bigcirc p$ is true at s_2. The other models are the same but with different commitment contents. According to the semantics of fulfillment, the state s_2 in $\mathcal{M}_{C1}$ is also labeled with the fulfillment of the commitment because: (1) s_2 is accessible from s_1; and (2) such a commitment has been established at s_1.*

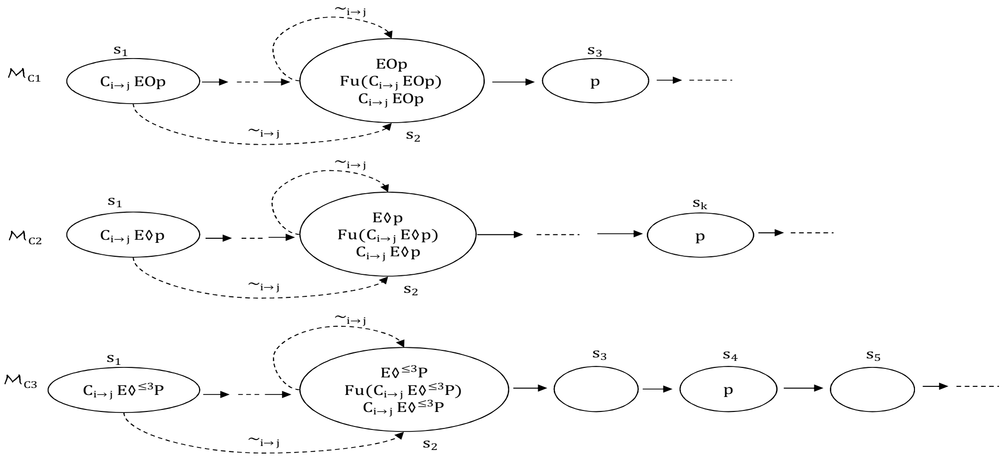


Figure 13: Illustration of the semantics of commitment and its fulfilment

Furthermore, our logic does not include an additional operator for violation as we did in the previous chapter and as in [11, 48]; instead weak and strong violations can be expressed as follows:

$$\neg A\Box(C_{i\to j}\varphi \ \rightarrow \ A\Diamond(Fu(C_{i\to j}\varphi))) \equiv E\Diamond(C_{i\to j}\varphi \wedge E\Box(\neg Fu(C_{i\to j}\varphi)))$$

and

$$\neg A\Box(C_{i\to j}\varphi \ \rightarrow \ E\Diamond(Fu(C_{i\to j}\varphi))) \equiv E\Diamond(C_{i\to j}\varphi \wedge A\Box(\neg Fu(C_{i\to j}\varphi)))$$

The weak violation takes place when there is a computation so that in its future a commitment is established but from the moment where the commitment is active there is a possible computation where globally the fulfillment never happens. The strong violation comes out when after having the commitment, the fulfillment does not occur in all states of every possible computation.

Figure 14 shows the relationship between a commitment and its fulfillment and its violation, from semantics perspective. Consider an example of a commitment made by a merchant (Mer) towards a customer (Cus) about φ, for example about delivering a given item. At the commitment state s_1, there are three choices available to the merchant, two of them satisfy its commitment and the other does not. Specifically, when the merchant reaches one of the accessible states s_2 or s_3 from the commitment state s_1, the commitment is fulfilled (the commitment content holds at s_2 and s_3), which means two different ways to fulfill the commitment are possible. However, the commitment is not fulfilled at s_4 where the merchant fails to deliver the item. In fact, the computation (s_1, s_4^*), where * means infinite repetition, corresponds to a violation of the commitment as in the future of this path, fulfillment will never take place. If we change the example so that s_2 and s_3 are not reachable from s_1, then the commitment is strongly violated.

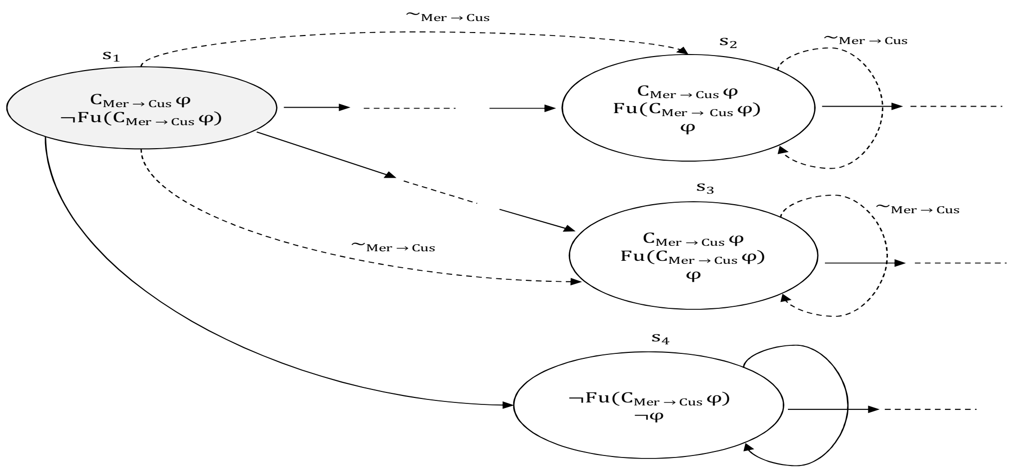


Figure 14: Illustrative example of commitment, its fulfilment, and its violation

The following proposition is a direct result from the semantics.

Proposition 2 *When the commitment is fulfilled, then there is no way to violate it in the future and vice versa.*

As we showed in Chapter 4 that ACTL^{*c} is semantically different from CTL*, by the same arguments, we can show that CTLC is also semantically different from CTL. For instance, $C_{i\to j}\varphi$ cannot be expressed by $E \bigcirc \varphi$ because the accessible state is not always the next state in a given path; it cannot be expressed by $\Diamond\varphi$ either because $C_{i\to j}\varphi \supset E\Diamond\varphi$, but $E\Diamond\varphi \supset C_{i\to j}\varphi$ is not always true.

Thus, any formula containing $C_{i\to j}\varphi$ cannot be expressed in CTL.

Definition 10 (Validity) *A formula φ is valid (we write $\models \varphi$) iff $(\mathcal{M}_C, s) \models \varphi$ for all models $\mathcal{M}_C$ of $\mathbb{M}$ and for all s of S.*

5.2.3 Axiomatization

We here investigate the logical properties that the models $\mathcal{M}_C$ of $\mathbb{M}$ inherit from the axiomatic point of view. The axiomatic method is the standard way to usually define a class of well-formed formulae (axioms) without any reference to their meanings. These axioms are plausible in some way which lead us to explore their consequences. As these axioms correspond the properties of our social accessibility relation, an immediate consideration comes from the following lemma.

Lemma 1 *The social accessibility relation $\sim_{i\to j}$ is serial, transitive and Euclidean.*

Proof 2

- *$\sim_{i\to j}$ is serial: this follows from the assumption that for any pair of agents $(i,j) \in \mathcal{A}$, we have that for any $s \in S$, the set $\sim_{i\to j}(s)$ is different from the empty set.*
- *$\sim_{i\to j}$ is transitive: assume $s \sim_{i\to j} s'$ and $s' \sim_{i\to j} s''$, for any pair $i,j \in \mathcal{A}$, , according to the definition of $\sim_{i\to j}$, it is the case that $s \sim_{i\to j} s''$ as $l_i(s) = l_i(s') = l_i(s'')$, $\exists! x \in Var_i \cap Var_j$ such that $l_i^x(s) = l_i^x(s') = l_j^x(s'')$, $l_j^y(s) = l_j^y(s') = l_j^y(s'')$ $\forall y \in Var_j - Var_i$.*
- *$\sim_{i\to j}$ is Euclidean: assume $s \sim_{i\to j} s'$ and $s \sim_{i\to j} s''$, for any pair of agents $(i,j) \in \mathcal{A}$, according to the definition of $\sim_{i\to j}$, we have $s' \sim_{i\to j} s''$ as $l_i(s') = l_i(s) = l_i(s'')$, $\exists! x \in Var_i \cap Var_j$ such that $l_i^x(s') = l_i^x(s) = l_j^x(s'')$, $l_j^y(s') = l_j^y(s) = l_j^y(s'')$ $\forall y \in Var_j - Var_i$.* ■

According to this observation, we can immediately conclude that the logic of commitments that deals with agent communication via social commitments is at least as strong as $KD45n$ (where n is the number of agents) which is to be expected. This logic is different from the one introduced in our previous work [52], which is $KB5n$ (for more details about the standard systems of modal logic, see for example [77, 103]). From Lemma 1, we obtain the following straightforward corollary.

Corollary 1 *The social accessibility relation $\sim_{i\to j}$ is shift reflexive, i.e. if $s \sim_{i\to j} s'$ then $s' \sim_{i\to j} s'$.*

As our objective in this thesis is to investigate the practical problem of model checking commitments for communicating agents, the completeness issue of the proposed logic will not be considered, so that the thesis is more focused on the implementation (cf. Section 5.3), algorithmic and computational complexity (cf. Chapter 6) aspects of the verification problem. In the following, we only study the individual validities (axioms) of commitment and fulfillment modalities with respect to Definition 10. The validities of commitment modality are:

Proposition 3 (Axioms of commitments)

1. $\models C_{i \to j}(\varphi \supset \psi) \supset (C_{i \to j}\varphi \supset C_{i \to j}\psi)$ *(axiom K)*
2. $\models C_{i \to j}\varphi \supset \widehat{C}_{i \to j}\varphi$ *(axiom D)*
3. $\models C_{i \to j}\varphi \supset C_{i \to j}(C_{i \to j}\varphi)$ *(axiom* 4*)*
4. $\models \widehat{C}_{i \to j}\varphi \supset C_{i \to j}(\widehat{C}_{i \to j}\varphi)$ *(axiom* 5*)*

Using the definition of the accessibility relation $\sim_{i \to j}$ and the semantics of $C_{i \to j}\varphi$ and $\widehat{C}_{i \to j}\varphi$, it is easy to check that the above validities hold. In light of the literature in this area (see for instance [103]), the proposed logic should be seen as supporting a *bird's eye view* of the properties of MASs. Therefore, the first validity of axiom K should seem relevant and reasonable. It means the commitment is closed under implication, that is if both the implication $\varphi \supset \psi$ and antecedent φ are committed to, then also the conclusion ψ is committed to.

The second validity, which corresponds to serial frames, expresses that if agent i commits towards agent j that φ, then φ is compatible with what i commits to towards j. Consequently, $\models \neg C_{i \to j}\bot$, which means that one cannot honestly, truthfully and justifiably state to commit to something that is false, which guarantees that the commitments of each agent are consistent.

The third and fourth validities are called the *positive* and *negative introspection axioms.* Introspection is the process of examining agent's own commitments. The positive introspection axiom corresponds to transitive frames. It is perhaps not as strong as a first reading might suggest. It should be read and understood that when an agent commits to φ, it should consider the action of committing as a commitment itself. In fact, it would be unreasonable to allow an agent to commit to φ without indicating that it is committing. Let us consider the example of an agent i conveying an information φ to another agent j, it is reasonable to say that i is conveying the information that it is conveying φ. More importantly, this will allow to account for the intentional commitment as the agent is committing that it is committing to φ. The commitment about commitment can be considered as a meta data (or meta information) prescribing a fact of committing as a commitment.

The negative introspection axiom, which corresponds to Euclidean frames, expresses that if the agent considers it possible to commit that φ, then it commits to what is possible to commit to. Another way of reading axiom 5 is to note that if the agent does not commit to not φ, then it commits to the fact that it does not commit to not φ, which again reflects the intentionality of

commitments. It is worth noticing that positive and negative introspection axioms are desirable and relevant as they encapsulate the fact that agents are aware of their commitments and have full control on what they do commit and do not commit to.
The validities of fulfillment modality are:

Proposition 4 (Axioms of fulfillment)

1. $\models Fu(C_{i\to j}\varphi) \supset \varphi$ *(axiom T)*
2. $\models C_{i\to j}\varphi \supset C_{i\to j}(Fu(C_{i\to j}\varphi))$
3. $\models C_{i\to j}(Fu(C_{i\to j}\varphi)) \supset C_{i\to j}\varphi$
4. $\models Fu(C_{i\to j}\varphi) \supset C_{i\to j}\varphi$

It is easy to check that the above validities hold using the definition of the accessibility relation $\sim_{i\to j}$ and semantics of $C_{i\to j}\varphi$ and $Fu(C_{i\to j}\varphi)$. The first validity, which corresponds to axiom T, expresses that if the commitment is fulfilled, then its content holds, which is very reasonable. The second validity says that if the agent commits to then it commits to fulfill its commitment, which is also a reasonable axiom. The third validity says that if the agent commits to fulfill a commitment about φ, then it is committing about φ, which, in other words, means the commitment is active when the agent commits to satisfy it. This is very reasonable and desirable as an agent cannot commit to satisfy a nonactive (nonexisting) commitment. In fact, putting the second and third validities together shows that committing to something is equivalent to committing to satisfy this commitment, which is very logical. From the semantics of $Fu(C_{i\to j}\varphi)$ and axiom 4 of commitments (where $\sim_{i\to j}$ is transitive, Proposition 3), we can entail the fourth validity. This validity says that if the commitment is fulfilled, the commitment appears in the same state as its fulfillment and is important for the logic of communicative commitments. This is because it technically allows us to have the following proposition (Proposition 5) saying "the commitment should be active when it comes time to its fulfillment", which is very important as it implies the impossibility of fulfilling a nonexisting commitment (cf. Figure 15).

Proposition 5 *The following validity holds:*

$$\models A\Box(\neg E(\neg C_{i\to j}\varphi\ U\ (\neg C_{i\to j}\varphi \wedge Fu(C_{i\to j}\varphi))))$$

Proof 3

Let us assume the opposite is true, which means: $E\Diamond(E(\neg C_{i\to j}\varphi\ U\ \neg C_{i\to j}\varphi \wedge Fu(C_{i\to j}\varphi)))$. *According to the semantics of until,* $\neg C_{i\to j}\varphi\ U\ (\neg C_{i\to j}\varphi \wedge Fu(C_{i\to j}\varphi))$ *is satisfied in a path if it has*

$\neg C_{i\to j}\varphi \wedge Fu(C_{i\to j}\varphi)$ in its future. However, from the fourth validity of Proposition 4, this cannot be the case; so the proposition. ■

It is important to mention that the commitment $C_{i\to j}\varphi$ in the fourth validity is not a new commitment that is being activated in the current state s but an existing commitment that has been activated in another state s' from which the state s emanates through $\sim_{i\to j}$. Figure 15 depicts the connection between commitment and its fulfillment in which s' is the state of activating the commitment (i.e., the commitment state) and s is the state of fulfilling the commitment (the current state). Notice that the commitment $C_{i\to j}\varphi$ activated in s' is still active in s as the accessibility relation is shift reflexive.

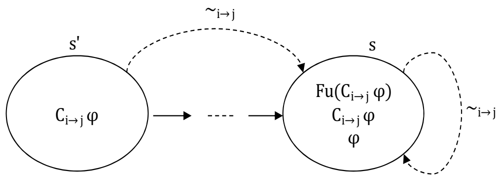


Figure 15: Link between commitment and its fulfillment.

In some approaches, for instance [127], instead of the fourth validity, the following axiom (called here (DA)) holds: $Fu(C_{i\to j}\varphi) \supset \neg C_{i\to j}\varphi$. However, DA has two problems. The first one is that it forces the fulfillment of commitments to be in the absolute future, so no commitment can be fulfilled in the same state at the moment of its activation. From our perspective, for communicative commitments such as those we are considering in this thesis, they can be fulfilled in the present, for example when an agent i conveys an information that already holds to an agent j. Thus, with DA axiom, committing about tautologies, atomic propositions and formulae without temporal operators is not supported. The second problem is that commitments can be fulfilled without being activated, which means another axiom (probably with past operators as in [144, 145]) is needed to prevent fulfilling a commitment which has not been activated or does not exist, or forcing the model to have the property discussed in the Proposition 5.

To summarize, interesting properties are then captured with our logic in terms of connection between commitments and their fulfillment, which are not addressed in the existing commitment logics and cannot be captured with a less strong logic, for instance a normal modal logic.

5.3 Model Checking CTLC using Reduction

In this section, we proceed to present our reduction techniques to model checking commitments and commitment-based protocols (the second part in our approach). In these techniques, we reduce the problem of model checking CTLC into the problem of model checking ARCTL and into the problem of model checking GCTL*. Before that, we define the problem of model checking CTLC: in a nutshell, given a MAS represented as an interpreted system $\mathcal{M}_C$ and a CTLC-formula φ describing a property, the problem of model checking CTLC can be defined as establishing whether or not $\mathcal{M}_C \models \varphi$ (i.e., $\forall s \in I : (\mathcal{M}_C, s) \models \varphi$).

5.3.1 Reducing CTLC into ARCTL

We very briefly review ARCTL logic (an extension of CTL with action formulae) [104]. We then show how the problem of model checking CTLC can be reduced to the problem of ARCTL model checking. An ARCTL mixes state formulae and action formulae by restricting path formulae to paths whose actions satisfy a given action formula. The syntax of ARCTL is defined by the following BNF grammar [104]:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid E_\alpha \bigcirc \varphi \mid E_\alpha(\varphi \; U \; \varphi) \mid E_\alpha \Box \varphi$$

$$\alpha ::= b \mid \neg\alpha \mid \alpha \vee \alpha$$

where φ is a state formula, α is an action formula, $p \in \mathcal{PV}$ is an atomic proposition, and $b \in \Phi_b$ is an atomic action proposition.

Definition 11 (Model of ARCTL) *A model $\mathcal{M}_A = (S_A, I_A, Act, TR, V_S, V_{Act})$ is a tuple where S_A is a nonempty set of states; $I_A \subseteq S_A$ is a set of initial states; Act is a set of actions; $TR \subseteq S_A \times Act \times S_A$ is a labeled transition relation; $V_S : S_A \to 2^{\mathcal{PV}}$ is a function assigning to each state a set of atomic propositions to interpret this state; and $V_{Act} : Act \to 2^{\Phi_b}$ is a function assigning to each action a set of atomic action propositions to interpret this action.*

The semantics of this logic [104] is given by defining the α-restriction of $\mathcal{M}_A = (S_A, I_A, Act, TR, V_S, V_{Act})$ as follows $\mathcal{M}_A^\alpha = (S_A, I_A, Act, TR^\alpha, V_S, V_{Act})$ where TR^α is a transition relation such that $(s, a, s') \in TR^\alpha$ iff $(s, a, s') \in TR$ and $a \models \alpha$ wherein $\models$ is defined as follows:

- $a \models b$ *iff* $b \in V_{Act}(a)$;
- $a \models \neg\alpha$ *iff not* $(a \models \alpha)$ *and*;
- $a \models \alpha \vee \alpha'$ *iff* $a \models \alpha$ *or* $a \models \alpha'$.

The motivation behind the α-restriction is to focus each time on specific transitions whose labels satisfy a given action formula, so all the other transitions are disregarded. This is useful when a formula has to be checked because only relevant transitions should be considered. Pecheur and Raimondi [104] have considered finite and infinite paths to define the semantics of ARCTL. However, we only consider the general case of infinite paths. $\Pi^{\alpha}(s)$ is the set of paths (called α-paths) whose actions satisfy a given action formula α and starting at s. Now, we define the satisfaction relation $(\mathcal{M}_A^{\alpha}, s) \models \varphi$, or concisely $s \models \varphi$, as follows (we omit the semantics of Boolean connectives and propositional atoms):

- $s \models E_{\alpha} \bigcirc \varphi$ *iff there exists a path* $\pi \in \mathit{\Pi}^{\alpha}(s)$ *and* $\pi(1) \models \varphi$,
- $s \models E_{\alpha}(\varphi\ U\ \psi)$ *iff there exists a path* $\pi \in \mathit{\Pi}^{\alpha}(s)$ *such that for some* $k \geq 0, \pi(k) \models \psi$ *and* $\pi(j) \models \varphi$ *for all* $0 \leq j \leq k$-1,
- $s \models E_{\alpha}\Box\varphi$ *iff there exists a path* $\pi \in \mathit{\Pi}^{\alpha}(s)$ *such that* $\pi(k) \models \varphi$ *for all* $k \geq 0$.

The reduction process is defined as follows: given a CTLC model $\mathcal{M}_C = (S, I, R_t, \{\sim_{i\to j} \mid (i,j) \in \mathcal{A}^2\}, \mathcal{V})$ and a CTLC formula φ, we have to define an ARCTL model $\mathcal{M}_A^{\alpha} = \mathscr{F}(\mathcal{M}_C)$ and an ARCTL formula $\mathscr{F}(\varphi)$ using a transformation function $\mathscr{F}$ such that $\mathcal{M}_C \models \varphi$ iff $\mathscr{F}(\mathcal{M}_C) \models \mathscr{F}(\varphi)$. The model $\mathscr{F}(\mathcal{M}_C)$ is defined as an ARCTL model $\mathcal{M}_A^{\alpha} = (S_A, I_A, Act, TR^{\alpha}, V_S, V_{Act})$ as follows:

- $S_A = S$; $I_A = I$; $V_S = \mathcal{V}$,
- The set Φ_b is defined as follows: $\Phi_b = \{\epsilon, \alpha_{1\to 1}, \alpha_{1\to 2}, \ldots, \alpha_{n\to n}\} \cup \{\beta_{1\to 1}, \beta_{1\to 2}, \ldots, \beta_{n\to n}\}$, and then $Act = \{\alpha^o, \alpha^{11}, \alpha^{12}, \ldots, \alpha^{nn}\} \cup \{\beta^{11}, \beta^{12}, \ldots, \beta^{nn}\}$ where α^o and α^{ij} are the actions labeling transitions respectively defined from the transition relation R_t and the accessibility relation $\sim_{i\to j}$, while β^{ij} is the action labeling transitions added when there exists a transition labeled with α^{ij} and needed to define the transformation of the formula $Fu(C_{i\to j}\varphi)$,
- The function V_{Act} is then defined as follows:

  1. $V_{Act}(\alpha^o) = \{\epsilon\}$, i.e., ϵ is the atomic action proposition forming α^o,
  2. $V_{Act}(\alpha^{ij}) = \{\alpha_{i\to j}\}$ for $1 \leq i \leq n$ and $1 \leq j \leq n$, i.e., $\alpha_{i\to j}$ is the atomic action proposition forming α^{ij},
  3. $V_{Act}(\beta^{ij}) = \{\beta_{i\to j}\}$ for $1 \leq i \leq n$ and $1 \leq j \leq n$, i.e., $\beta_{i\to j}$ is the atomic action proposition forming β^{ij}.

- The labeled transition relation TR^{α} combines the temporal labeled transition R_t and the accessibility relation $\sim_{i\to j}$ under the following conditions: for states $s, s' \in S$,

1. $(s, \alpha^0, s') \in TR^{\epsilon}$ if $(s, s') \in R_t$,
2. $(s, \alpha^{ij}, s') \in TR^{\alpha_{i\to j}}$ if $s \sim_{i\to j} s'$,
3. $(s, \beta^{ij}, s') \in TR^{\beta_{i\to j}}$ if $(s', \alpha^{ij}, s) \in TR^{\alpha_{i\to j}}$.

From the definition of $\mathscr{F}(\mathcal{M}_C)$, it is clear that $\mathscr{F}(S) = S_A$. Let us now define the transformation of a CTLC formula φ (i.e., $\mathscr{F}(\varphi)$) by induction on the form of φ.

– $\mathscr{F}(p) = p$, if p is an atomic proposition,

– $\mathscr{F}(\neg\varphi) = \neg\mathscr{F}(\varphi)$, and $\mathscr{F}(\varphi \vee \psi) = \mathscr{F}(\varphi) \vee \mathscr{F}(\psi)$,

– $\mathscr{F}(E \bigcirc \varphi) = E_\epsilon \bigcirc \mathscr{F}(\varphi)$, and $\mathscr{F}(E(\varphi\ U\ \psi)) = E_\epsilon(\mathscr{F}(\varphi)\ U\ \mathscr{F}(\psi))$,

– $\mathscr{F}(E\Box\varphi) = E_\epsilon\Box\mathscr{F}(\varphi)$, and $\mathscr{F}(C_{i\to j}\varphi) = A_{\alpha_{i\to j}} \bigcirc \mathscr{F}(\varphi)$,

– $\mathscr{F}(Fu(C_{i\to j}\varphi)) = E_{\beta_{i\to j}} \bigcirc \mathscr{F}(C_{i\to j}\varphi)$.

Thus, this reduction allows us to model-check CTLC formulae by model checking their reductions in ARCTL using the extended NuSMV tool introduced in [90]. Figure 16 illustrates the reduction process of the fulfillment formula. The following theorem proves the soundness of our reduction from CTLC to ARCTL.

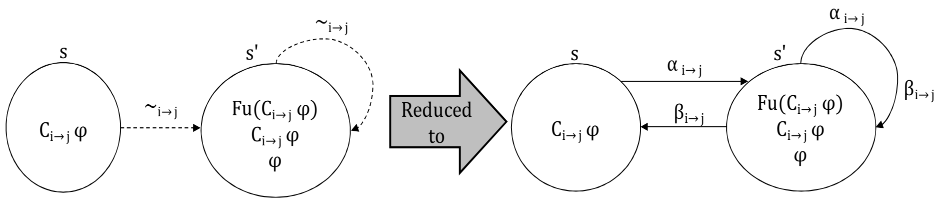


Figure 16: An example of the reduction process from CTLC to ARCTL.

Theorem 2 (Soundness of $\mathscr{F}$) *Let $\mathcal{M}_C$ and φ be respectively a CTLC model and formula and let $\mathscr{F}(\mathcal{M}_C)$ and $\mathscr{F}(\varphi)$ be the corresponding model and formula in ARCTL. We have $\mathcal{M}_C \models \varphi$ iff $\mathscr{F}(\mathcal{M}_C) \models \mathscr{F}(\varphi)$.*

Proof 4

We prove this theorem by induction on the structure of the formula φ. All the cases are straightforward once the following two cases are analyzed:

– *$\varphi = C_{i\to j}\psi$. We have $(\mathcal{M}_C, s) \models C_{i\to j}\psi$ iff $(\mathcal{M}_C, s') \models \psi$ for every $s' \in S$ such that $s \sim_{i\to j} s'$. Consequently, $(\mathcal{M}_C, s) \models C_{i\to j}\psi$ iff $(\mathcal{M}_A^{\alpha_{i\to j}}, s') \models \mathscr{F}(\psi)$ for every $s' \in S_A$ such that $(s, \alpha^{ij}, s') \in TR^{\alpha_{i\to j}}$. As $\sim_{i\to j}$ is shift reflexive, we obtain an infinite path $\pi \in \Pi^{\alpha_{i\to j}}(s)$ such that $\pi(1) = s'$ and $(\mathcal{M}_A^{\alpha_{i\to j}}, \pi(1)) \models \mathscr{F}(\psi)$ (cf. Figure 16). By semantics of $A_{\alpha_{i\to j}}\bigcirc$ in ARCTL, we obtain $(\mathcal{M}_A^{\alpha_{i\to j}}, s) \models A_{\alpha_{i\to j}} \bigcirc \mathscr{F}(\psi)$.*

– $\varphi = Fu(C_{i\to j}\psi)$. *We have* $(\mathcal{M}_C, s') \models Fu(C_{i\to j}\psi)$ *iff* $(\mathcal{M}_C, s) \models C_{i\to j}\psi$ *for a state* $s \in S$ *such that* $s \sim_{i\to j} s'$. *Consequently,* $(\mathcal{M}_C, s') \models Fu(C_{i\to j}\psi)$ *iff* $(\mathcal{M}_A^{\alpha_{i\to j}}, s) \models \mathscr{F}(C_{i\to j}\psi)$ *for a state* $s \in S_A$ *such that* $(s, \alpha^{ij}, s') \in TR^{\alpha_{i\to j}}$. *As* $\sim_{i\to j}$ *is shift reflexive, we obtain* $s' \sim_{i\to j} s'$ *and so* $(s', \alpha^{ij}, s') \in TR^{\alpha_{i\to j}}$. *Consequently,* $(s', \beta^{ij}, s') \in TR^{\beta_{i\to j}}$. *There is then an infinite path* $\pi \in \Pi^{\beta_{i\to j}}(s')$ *such that* $\pi(1) = s'$ *and* $(\mathcal{M}_A^{\beta_{i\to j}}, \pi(1)) \models \mathscr{F}(C_{i\to j}\psi)$ *(cf. Figure 16). By semantics of* $E_{\beta_{i\to j}}\bigcirc$ *in ARCTL, we obtain* $(\mathcal{M}_A^{\beta_{i\to j}}, s') \models E_{\beta_{i\to j}}\bigcirc\, \mathscr{F}(C_{i\to j}\psi)$*, so we are done.* ■

Remark 6 *By using the same arguments introduced in Remark 5 in Chapter 4, we cannot claim that CTLC is sound and complete as a corollary of Theorem 2. However, we can extend the standard decision procedure of CTL proposed by Emerson and Halpern [57] to prove the soundness and completeness of an axiomatic system for CTLC, but such issues are beyond the scope of the thesis and will be explored as future work.*

5.3.2 Reducing CTLC into GCTL*

As in the reduction of CTLC into ARCTL, we start with introducing the syntax of GCTL* and the model of GCTL*, which are given and discussed in Chapter 4. Here, we recall the syntax of GCTL*:

$$\mathcal{S} ::= p \mid \neg\mathcal{S} \mid \mathcal{S} \vee \mathcal{S} \mid E\ \mathcal{P}$$
$$\mathcal{P} ::= \theta \mid \neg\mathcal{P} \mid \mathcal{S} \mid \mathcal{P} \vee \mathcal{P} \mid \bigcirc\mathcal{P} \mid \mathcal{P}\ U\ \mathcal{P}$$

where $p \in \mathcal{PV}$ is an atomic proposition and $\theta \in \Phi_b$ is an atomic action proposition. The GCTL* model is a six-tuple $\mathcal{M}_G = (S_G, Ac, l_S, l_{Ac}, \to, I_G)$ where S_G is a nonempty set of states; Ac is a set of actions; $l_S : S_G \to 2^{\mathcal{PV}}$ is a state labeling function; $l_{Ac} : Ac \to 2^{\Phi_b}$ is an action labeling function; $\to \subseteq S_G \times Ac \times S_G$ is a transition relation; and $I_G \subseteq S_G$ is a set of initial states. The semantics of GCTL* formulae is defined in Chapter 4.

The reduction process from the problem of model checking CTLC to the problem of model checking GCTL* that allows us to a direct use of CWB-NC is defined as follows: given a CTLC model $\mathcal{M}_C = (S, I, R_t, \{\sim_{i\to j} \mid (i,j) \in \mathcal{A}^2\}, \mathcal{V})$ and a CTLC-formula φ, we have to define a GCTL* model $\mathcal{M}_G = \mathscr{H}(\mathcal{M}_C)$ and a GCTL* formula $\mathscr{H}(\varphi)$ using a transformation function $\mathscr{H}$ such that $\mathcal{M}_C \models \varphi$ iff $\mathscr{H}(\mathcal{M}_C) \models \mathscr{H}(\varphi)$. The model $\mathscr{H}(\mathcal{M}_C)$ is defined as a GCTL* model $\mathcal{M}_G = (S_G, Ac, l_S, l_{Ac}, \to, I_G)$ as follows:

– $S_G = S$; $I_G = I$; $l_S = \mathcal{V}$,

– To define the set Ac, let us first define the set of atomic action propositions $\Phi_b = \{\epsilon, \alpha_{1\to 1}, \alpha_{1\to 2}, \ldots, \alpha_{n\to n}\} \cup \{\beta_{1\to 1}, \beta_{1\to 2}, \ldots, \beta_{n\to n}\}$, then $Ac = \{\alpha^o, \alpha^{11}, \alpha^{12}, \ldots, \alpha^{nn}\} \cup \{\beta^{11}, \beta^{12}, \ldots, \beta^{nn}\}$ where α^o and α^{ij} are the actions labeling transitions respectively defined from the transition relation R_t and the accessibility relation $\sim_{i\to j}$, while β^{ij} is the action labeling transitions added when there exists a transition labeled with α^{ij} and needed to define the transformation of the formula $Fu(C_{i\to j}\varphi)$,

– The function l_{Ac} is then defined as follows:

1. $\alpha^o \in Ac$, then $l_{Ac}(\alpha^o) = \{\epsilon\}$,
2. $l_{Ac}(\alpha^{ij}) = \{\alpha_{i\to j}\}$ for $1 \le i \le n$ and $1 \le j \le n$,
3. $l_{Ac}(\beta^{ij}) = \{\beta_{i\to j}\}$ for $1 \le i \le n$ and $1 \le j \le n$.

– The labeled transition relation $\to$ combines the temporal labeled transition R_t and the accessibility relation $\sim_{i\to j}$ under the following conditions: for states $s, s' \in S$,

1. $(s, \alpha^0, s') \in\to$ if $(s, s') \in R_t$,
2. $(s, \alpha^{ij}, s') \in\to$ if $s \sim_{i\to j} s'$,
3. $(s, \beta^{ij}, s') \in\to$ if $(s', \alpha^{ij} s) \in\to$.

From the definition of $\mathscr{H}(\mathcal{M}_C)$, it is clear that $\mathscr{H}(S) = S_G$. Let us now define $\mathscr{H}(\varphi)$ by induction on the form of φ.

– $\mathscr{H}(p)$=p, if p is an atomic proposition,

– $\mathscr{H}(\neg\varphi) = \neg\mathscr{H}(\varphi)$, and $\mathscr{H}(\varphi \vee \psi) = \mathscr{H}(\varphi) \vee \mathscr{H}(\psi)$,

– $\mathscr{H}(E \bigcirc \varphi) = E \bigcirc \mathscr{H}(\varphi)$, and $\mathscr{H}(E(\varphi\ U\ \psi)) = E(\mathscr{H}(\varphi)\ U\ \mathscr{H}(\psi))$,

– $\mathscr{H}(EG\varphi) = E\Box\mathscr{H}(\varphi)$, and $\mathscr{H}(C_{i\to j}\varphi) = A(\alpha_{i\to j} \supset \bigcirc\mathscr{H}(\varphi))$,

– $\mathscr{H}(Fu(C_{i\to j}\varphi)) = E(\beta_{i\to j} \wedge \bigcirc\mathscr{H}(C_{i\to j}\varphi))$.

Theorem 3 (Soundness of $\mathscr{H}$) *Let $\mathcal{M}_C$ and φ be respectively a CTLC model and formula and let $\mathscr{H}(\mathcal{M}_C)$ and $\mathscr{H}(\varphi)$ be the corresponding model and formula in GCTL*. We have $\mathcal{M}_C \models \varphi$ iff $\mathscr{H}(\mathcal{M}_C) \models \mathscr{H}(\varphi)$.*

Proof 5

We can prove this theorem by induction on the structure of φ in a way similar to the proof of Theorem 2. ■

Remark 7 *GCTL* and ARCTL logics can both express properties of state-based and action-based models and a careful analysis of the semantics of these two logics reveals that GCTL* subsumes ARCTL. In fact, in GCTL*, a path satisfies θ iff the path contains at least one transition and the label of the first transition on the path satisfies θ where θ is an atomic action proposition, and an action a satisfies θ iff $\theta \in l_{Ac}(a)$. In ARCTL, the actions of the whole path, not only of the first transition, should satisfy an action formula α as the transitions in the model of ARCTL are α-restricted. In fact, ARCTL allows quantification over action-labelled paths, so a path formula is evaluated on α-paths, which means paths where all transitions are α-restricted, which can also be done in GCTL*. For example, for a path formula γ, $E_\alpha\gamma$ in ARCTL means there is an α-path where γ holds, which can be expressed in GCTL* as: $E(\Box\alpha \wedge \gamma)$. However, although GCTL* subsumes ARCTL, their model checking techniques are different and our motivation behind using these two logics is to be able to use their respective model checkers, which are based on two different model checking techniques: automata-based technique for GCTL* and symbolic, OBDD-based technique for ARCTL.*

5.4 Case Studies

One of the main motivations of this chapter is to check the effectiveness of our reduction techniques and experimentally compare between automata-based and symbolic OBDDs-based techniques.

We implemented the reduction techniques presented in Section 5.3 on top of the extended NuSMV and CWB-NC. Such tools have been used to check action formulae as well as state formulae. Specifically, NuSMV has been successfully adopted to model checking Web service composition [84], multi-agent interaction protocols [48], and business models [139]. One limitation is that it does not support model checking epistemic properties in a system of agents. The extended NuSMV is used to overcome such a limitation [90, 104]. It has also been used to verify commitment-based protocols [50]. CWB-NC is used for model checking large-scale protocols in agent communication [11] and security pattern composition [44]. Concretely, the reduction from CTLC to ARCTL using the transformation function $\mathscr{F}$ is performed by Com2Arctl tool we have implemented using a library of M4 macros. M4 is a general-purpose macro processor available on most UNIX platforms. The reduction from CTLC to GCTL* using the transformation function $\mathscr{H}$ is performed by Comm2Cwb tool we have implemented too. As shown in Figure 17, the solely manual intervention is the provision of the input file describing the problem to be verified.

On the one hand, the extended NuSMV is a symbolic model checker based on OBDDs, where the

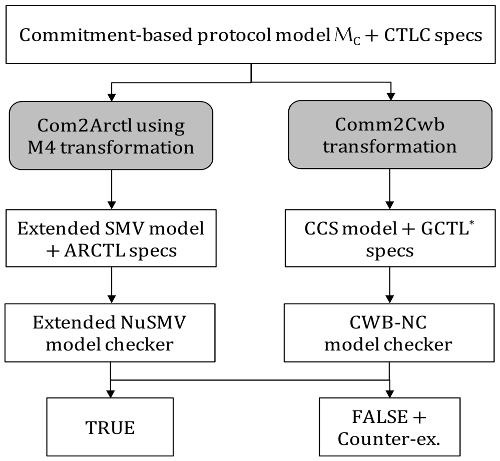

Commitment-based protocol model $\mathcal{M}_c$ + CTLC specs
Com2Arctl using M4 transformation
Comm2Cwb transformation
Extended SMV model + ARCTL specs
CCS model + GCTL* specs
Extended NuSMV model checker
CWB-NC model checker
TRUE
FALSE + Counter-ex.

Figure 17: Verification work flow for commitment-based protocols.

states of the model and formula to be checked are represented by means of Boolean functions, which can be easily represented using OBDDs and the set of states of the model satisfying an ARCTL formula is also represented as a Boolean function, which is encoded as an OBDD too. By comparing the later set with the set of initial states encoded also as an OBDD, it is possible to establish whether or not a formula holds in a given model. As a result, the problem of model checking ARCTL is reduced to the comparison of OBDDs. In the extended NuSMV, models are described by means of a modular language called *extended symbolic model verifier* (extended SMV) with respect to a finite state machine formalism.

On the other hand, the CWB-NC is an automata-based model checker based on ABTAs, which are a variation of alternating tree automata to support efficient model checking wherein the algorithm searches only the part of the state space that needs to be explored to prove or disprove a certain formula [19]. That is, the state space is never constructed *a priori* because the CWB-NC model checker implements an on-the-fly technique. Specifically, in CWB-NC, instead of building the automata for both the formula and the model first, it only builds the ABTA of the formula. It then uses the formula's ABTA to guide the construction of the model's ABTA while computing the intersection. As a result, the problem of model checking GCTL* is reduced to check the emptiness condition of the ABTA intersection of the ABTA accepting the model and the ABTA accepting the formula. In CWB-NC, the models are described into a process algebra language called CCS.

Two case studies for which we have been able to carry out the above motivations are the NetBill protocol (NB) and Contract Net protocol(CN), already applied to show how commitments can specify protocols in business settings as we pointed out in Chapter 4, for example.

5.4.1 Verifying the NetBill Protocol

Having shown in Chapter 4 that our formal specification language is employed successfully to specify the NB protocol that orchestrates and regulates interactions between the merchant (*Mer*) agent and the customer (*Cus*) agent. We also used our model to formalize the protocol and compiled it using a Büchi automaton. As our specification language provides modularity aspect (as it is defined in terms of a set of patterns where each pattern involves one or more commitment actions), then it is easy to remove or add patterns regarding to the current business scenarios in order to only focus on commitments and their fulfillment and violation. In particular, here we remove patterns that define *Release, Withdraw, Delegate* and *Assign* actions. Figure 18 depicts the compilation of the NB protocol formalized in our model $\mathcal{M}_C$=$(S, I, R_t, \{\sim_{i\to j} \mid (i,j) \in \mathcal{A}^2\}, \mathcal{V})$ using the Büchi automaton $\mathcal{B}$= $(Q, \Sigma, Q_0, F, \delta)$ in which Q=S the set of global states, Σ=$\mathcal{A} \times ACT \ \cup \ \{\sim_{i\to j} \mid (i,j) \in \mathcal{A}^2\}$ the alphabet used to label transitions, Q_0=I, F=$\{s_9\}$, and δ=$R_t \ \cup \ \{\sim_{i\to j} \mid (i,j) \in \mathcal{A}^2\}$ the union of the transition relation and the accessibility relation $\sim_{i\to j}$. In the figure, each transition connecting

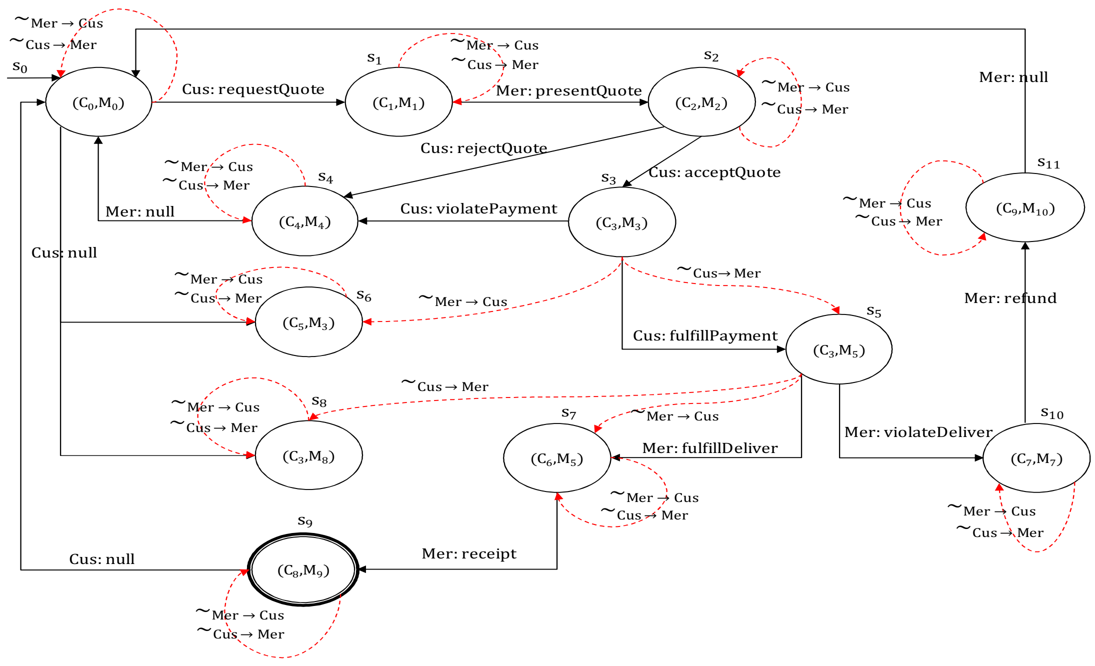


Figure 18: The NB protocol representation using our model $\mathcal{M}_C$.

global states is labeled by an agent and its local action which implicity means the local action of other agent is *null*. The global states are numbered and also identified with their local states of the two interacting agents. Also, the dashed arrows refer to the social accessibility relation using $\sim_{i\to j}$. The transition loops are added at some global states to make the two accessibility relations for the two interacting agents serial, so each state should have a minimum of two accessibility relations.

This compilation can be easily extended to any number n of agents greater than two as we show next in our experiments. To automatically verify the NB protocol using the extended NuSMV model checker, we have to encode it using its input language namely, the extended SMV. In particular, the participating agents in the NB protocol are encoded as a set of isolated modules `MODULE anAgent<name>` in which each agent module is instantiated in the main module at run time using the `VAR` keyword, which generally defines the SMV variables. The main module also includes the `SPEC` keyword to specify formulae that need to be checked using the ARCTL syntax. For each agent, we can use the `VAR` keyword to define its local states including the commitment states (s_3 and s_5), fulfillment states (s_5 and s_7), violation states (s_4 and s_{10}), acceptance state (s_9), recovery state (s_{11}), other reachable global states in the form of enumeration type. The local actions of each agent are represented as input variables using the `IVAR` keyword. Agent's protocol is defined as a relation between its local state and local action variables through the `TRANS` statement. The labeled transitions among states are encoded using the `TRANS` statement with the `next` and `case` expressions that represent agents's choices in a sequential manner, and initial conditions using the `INIT` statement. The main components of agent definition as an SMV module are defined as follows:

```
MODULE main                                 -- main module
VAR Cus : Customer(args1,args2);            -- customer agent
    Mer : Merchant(args1,agrs2);            -- merchant agent
SPEC <formulae_list>;                       -- list of formulae
MODULE Customer(args1,agrs2)                -- customer agent module
VAR    local_state: {...};                  -- customer's local states
IVAR local_action: {...};                   -- customer's local actions
INIT(...);                                  -- initial condition
  TRANS(local_action = case ... esac);      -- customer's protocol
INIT(...);                                  -- initial condition
  TRANS(next(local_state)=  case ... esac); -- customer's evolution function
```

Notice that "`--`" defines comments in the SMV program. Many properties can be checked in the NB protocol as we showed in Chapter 4, such as *fairness constraints, liveness, safety, reachability, deadlock freedom, livelock freedom.* Two examples of these properties formalized using CTLC are listed in Table 10. The first formula is given in the universal form and the second one uses the existential form. The first formula expresses an example of the standard *safety property.* As we mentioned in Chapter 4, the safety property can be expressed by $A\Box\neg p$ where p characterizes a "bad" situation that we must be avoided. In our protocol, a bad situation happens when the customer fulfills its commitment by sending the payment, but the merchant never commits to deliver the requested goods. The motivation behind this property is to check if the protocol is consistent, i.e., the NB protocol should not yield conflicting computations.

Table 10: Examples of tested formulae

$\varphi_1 = A\Box\neg\big(Fu(C_{Cus\to Mer}pay) \ \wedge A\Box\neg\ C_{Mer\to Cus}\ deliver\big)$
$\varphi_2 = E\Diamond Fu(C_{Mer\to Cus}\ deliver)$

The formula φ_2 is an example of the standard *reachability property*, i.e., given the initial state, we can reach a certain state via some computation sequences. It states that along a given path, it is eventually the case that there is a possibility for the merchant to fulfill its commitment by delivering the requested goods. This property checks if the NB protocol is effective, i.e., the protocol's transitions should be enough to confirm that it can be executed and ended successfully.

Our experimental results were performed on a laptop equipped with the Intel(R) Core(TM) i5-2430M clocked at 2.4GHz processor and 6GB memory running under Ubuntu Linux 8.04 with a vanilla 2.6.24-28-generic Kernel. We reported 5 experiments in Table 11 wherein the number of agents (#Agent), number of reachable states (#States), execution time (Time) in seconds (sec), which is the summation of the time required for building all OBDDs parameters and the actual execution time for the verification, and memory in use (Memory) in MB are given. From Table 11, we notice that the number of reachable states (which reflects the state space) and execution time—especially from experiment 3—increase exponentially when the number of agents increases. In contrast, the memory usage does not increase exponentially, but only polynomially when augmenting the number of agents.

Table 11: Verification results of the NB protocol using extended NuSMV

#Agents	**#States**	**Time(sec)**	**Memory(MB)**
2	12	0.020	4.241
3	446	0.184	5.507
4	4224	2.736	12.957
5	33454	63.687	15.432
6	238787	630.914	83.839

Notice that from experiment 2 we rewrite the defined formulae in a parameterized form, for example in experiment 5:

$$\varphi_2' = E\Diamond\ (\bigwedge_{i=1}^{5} Fu(C_{Mer \to Cus_i}\ deliver_i))$$

This formula captures the reachability property in the NB protocol that intuitively means the merchant along a given path has a possibility to eventually fulfill its commitment by delivering the requested goods to the five customers paid for these goods.

The encoding of the NB protocol formalized and compiled respectively using our model and a Büchi automaton with the CCS language to use CWB-NC as a benchmark is similar to the one explained in Chapter 4. Experimental results for the verification of the NB protocol are reported in Table 12. These experiments were performed on a laptop equipped with the Intel(R) Core(TM) i5-2430M clocked at 2.4GHz processor and 6GB memory running under x86_64 Windows 7. The execution time, in seconds, (i.e., the summation of the time required for building all ABTAs parameters and the actual execution time for the verification) and memory usage (in MB) are reported in Table 12. This table shows only the results of checking φ_2 and its parameterized form because as φ_2 is satisfied and includes the existential path quantifier, it only performs on a fragment of the model, whilst φ_1, as it is satisfied and has the universal form, needs to be performed on the whole model. Notice that unlike the case with the automata-based CWB-NC, the verification results for model checking using extended NuSMV, which is based on OBDD techniques, are not affected by the structure of the formula being verified [90]. Moreover, we put "N/A" in Table 12, which means that CWB-NC is not applicable regarding to the state explosion problem (cf. Subsection 2.3.2 in Chapter 2 for more details about this problem). Here again the number of reachable states and execution time increase exponentially when the number of agents increases while the memory usage increases only polynomially.

Table 12: Verification results of the NB protocol using CWB-NC

#Agents	**#States**	**Time(sec)**	**Memory(MB)**
2	325	0.035	4.020
3	6501	1.785	10.340
4	128327	58.872	25.470
5	N/A	N/A	N/A

5.4.2 Verifying the Contract Net Protocol

The CN protocol is designed from online business point of view. It is enhanced with commitments and their actions, which specify business meanings of the protocol's messages in high-level abstractions [157, 50]. In this case study, we have n agents, which represent the managers (Mgr) and participants (Prt) that interact with each other to send results about a particular task. As we mentioned in Chapter 4, the CN protocol is started when the manager calls for proposals about a particular task of interest. Each participant has two choices: (1) sending a proposal message; or (2) sending a reject message. At this stage, the manager accepts only one proposal among the received proposals and explicitly rejects the rest. The complete description of the protocol is introduced in Chapter 4. The protocol is self-described in Figure 19 using our model $\mathcal{M}_C$=$(S, I, R_t, \{\sim_{i\to j} \mid (i,j) \in \mathcal{A}^2\}, \mathcal{V})$. As we did in the previous case study, the protocol is compiled using the Büchi automaton $\mathcal{B}$= $(Q, \Sigma, Q_0, F, \delta)$ as follows: Q=S; Σ=$\mathcal{A}\times ACT \cup \{\sim_{i\to j} \mid (i,j) \in \mathcal{A}^2\}$; Q_0=I; F=$\{s_7\}$; and δ=$R_t \cup \{\sim_{i\to j} \mid (i,j) \in \mathcal{A}^2\}$. Notice that legal choices (polices) of agents at local states are enabled by using the local protocol function in the formalism of interpreted systems. For instance, in Figure 19, the local protocol $\mathcal{P}$ of the participant Prt enables *reject* and *proposal* actions at the local state P_1, i.e., $\mathcal{P}_{Prt}(P_1) = \{reject, proposal\}$. An example of the safety property

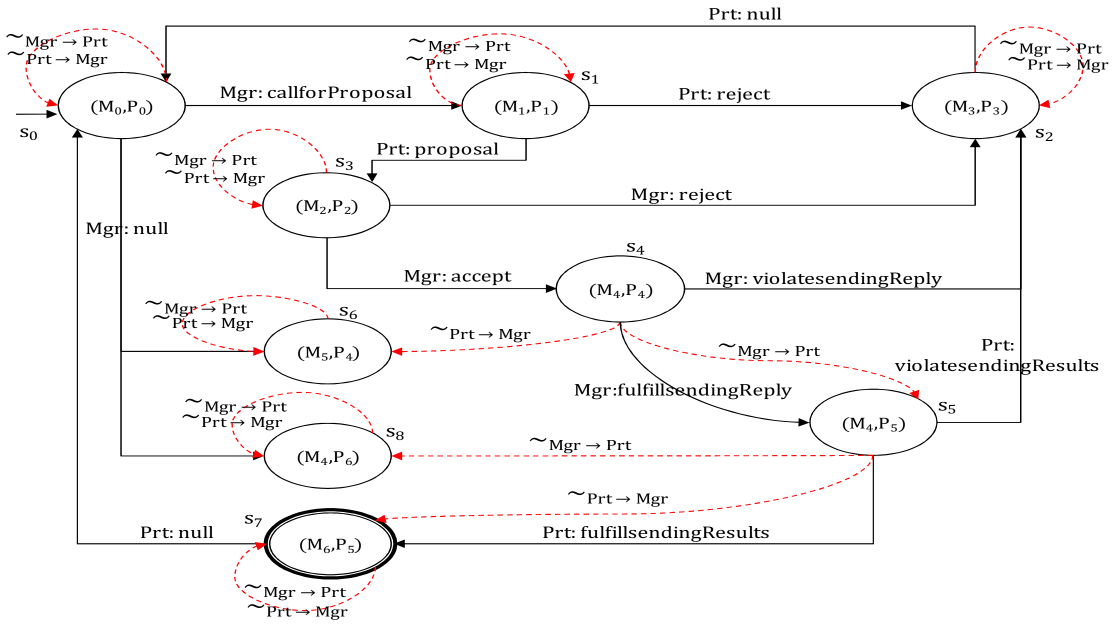


Figure 19: The CN protocol representation using our model $\mathcal{M}_C$.

in this protocol can be expressed by the formula φ_3 stating a bad situation: the manager fulfills its commitment by sending reply message, but the participant never commitments to deliver the results of the proposal. The formula φ_4 is an example of the standard reachability property in the CN protocol, which means along a given path, there is a possibility for the participant to fulfill its commitment by sending the results performing its proposal.

$$\varphi_3 = A\Box\neg(Fu(C_{Mgr\to Prt}\ reply) \land A\Box\neg\ C_{Prt\to Mgr}\ results)$$

$$\varphi_4 = E\Diamond\ Fu(C_{Prt\to Mgr}\ results)$$

Table 13 shows the CN protocol verification results with the extended NuSMV using the same machine as in the previous case study. Because the number of reachable states that are effectively considered in the CN protocol is much more smaller, its execution time is shorter than in the NB protocol with the extended NuSMV. Table 14 reports the verification results of the CN protocol using CWB-NC. In both cases, the memory usage increases polynomially when augmenting the number of agents as we expected.

Table 13: Verification results of the CN protocol using extended NuSMV

#Agents	**#States**	**Time(sec)**	**Memory(MB)**
2	9	0.044	4.226
3	528	0.1	4.734
4	4121	0.908	14.414
5	30346	11.285	14.835
6	217631	145.913	31.610

Table 14: Verification results of the CN protocol using CWB-NC

#Agents	**#States**	**Time(sec)**	**Memory(MB)**
2	257	0.038	3.013
3	5655	1.219	7.112
4	114717	50.754	18.022
5	N/A	N/A	N/A

We should underline that CWB-NC is efficiently applied to check satisfiability of existential formulae (i.e., for checking that a universal formula does not hold), which validate the main ABTA idea of finding counter-examples without exploring the whole model. It is also efficiently applicable when the model of MAS formalized using the interpreted systems is getting larger. To that end, we do not compare the results of the present chapter with the results of Chapter 4 regarding the two adopted protocols because these results are based on two different models and encodings. For instance, the local actions of each agent in Chapter 4 are more than the ones used here; so agent's model has more reachable states than the agent's model in this chapter (cf. for example the first experiment). The encoding in Chapter 4 does not consider the local protocol function as it is not defined in the underlying model.

5.5 Discussion and Related Work

The first contribution of this chapter lies in presenting a new full-computationally grounded semantics for communicative commitments and their fulfillment using CTLC, an extension of CTL with modalities for commitments and their fulfillment. This semantics excludes spurious aspects that plague existing approaches by balancing between expressiveness and verification efficiency. It also establishes a new link between commitments and their fulfillment, which is entirely missing in the literature about commitment representations. The valid axioms of commitment and fulfilling commitment modal formulae and a new formal reduction tools that transform the problem of model checking the above language into the problems of model checking ARCTL and GCTL* without missing or restricting real and concrete semantics are also introduced. Moreover, we proved the soundness of the proposed reduction techniques. Regarding to the business protocols used, we have experimentally evaluated the effectiveness and efficiency of our reduction techniques and our verification approach implemented using two different model checkers (extended NuSMV and CWB-NC). These experiments paint the following picture: the model checkers were able to verify a variety of complex formulae correctly and efficiently. This chapter establishes the practical usability of the approach by applying it to a large business protocol having approximately 2.3e+06 states, thanks to the OBDDs-based symbolic encodings used in extended NuSMV. The final conclusion confirms the the observation that automatic verification using symbolic techniques is moderately better than automata-based techniques in terms of the execution time and the number of interacting agents.

In what follows, we compare our approach with the recent extension of the formalism of interpreted systems. Lomuscio and Sergot [94] extended the interpreted system formalism to model the "correct and incorrect functioning behaviors of the agents" by dividing global states into "green" and "red" states. The resulting logic of deontic interpreted systems is stronger than the standard deontic logic. They also presented the axioms of this logic and proved theoretically its soundness and completeness. The semantics of correct functioning behavior is defined using a new accessibility relation R_i for each agent i on the global states as follows: for any $i \in \mathcal{A}, (l_1, \ldots, l_n)R_i(l'_1, \ldots, l'_n)$ if $l'_i \in G_i$ (the set of green states for i). This deontic accessibility relation defines accessible states (green states) independently from the current state. From this deontic perspective, there are some similarities between this logic and the logic of commitments. However, the deontic accessibility relation is different from our accessibility relation, which considers both current and accessible states. Furthermore, our accessibility relation accounts for communication between the participating agents by considering the communication channels. It is also unclear how in the deontic interpreted systems the green and red states are determined for each agent. The authors also defined a modality $\widehat{K}_i^j\varphi$, called "doubly-indexed operator", which can be read as "agent i knows φ under the assumption that agent j is functioning correctly". This modality attempts to combine deontic and epistemic modalities to specify what an agent is permitted to know. It indirectly captures the interaction between agents, but it differs from our commitment modality $C_{i\to j}\varphi$ as i does not assume anything about j and it has the possibility of fulfilling its commitment (i.e., correct functioning behavior) or violating it (i.e., incorrect functioning behavior), but in both cases, the commitment's content holds in all accessible states, and the commitment is fulfilled if the agent can be in one of these states, otherwise it is violated. Thus, reasoning about social commitments is different from reasoning about doubly-indexed operator.

While formal reduction techniques are to be preferred to informal ones, they add overhead over the actual verification process to perform the translation preprocessing step. They are also hard to scale up well our model to include many agents, (say, 15 agents). In Chapter 6, we develop a new symbolic model checking algorithm to directly and automatically verify commitments and their fulfillment and commitment-based protocols.

Chapter 6

Symbolic Model Checking Commitments and its Complexity Analysis

In this chapter[1], we develop a new symbolic algorithm for model checking the proposed logic, CTLC, in Chapter 5. We proceed to analyze the time complexity of CTLC model checking for explicit models (i.e., Kripke-like structures) and its space complexity for concurrent programs, which provide succinct representations. We prove that although CTLC extends CTL, their model checking algorithms still have the same time complexity for explicit models, which is P-complete with regard to the size of the model and length of the formula, and the same space complexity for concurrent programs, which is PSPACE-complete with regard to the size of the components of these programs. We fully implement the proposed algorithm on top of the multi-agent model checker MCMAS so that communicative commitments and their fulfillment and violation and commitment-based protocols can be directly verified. Furthermore, we provide in this chapter simulation results of an industrial case study, called *Insurance Claim Processing*, to check the effectiveness of our algorithm.

[1]The results of this chapter have been published in the Journal of Autonomous Agent Multi-Agent Systems [51] and Journal of Knowledge-Based Systems [8] regarding respectively to complexity analysis and logical and implementation parts.

6.1 Introduction

Using a logic of commitments to analyze communicative multi-agent systems is of a great significance. For instance, when informally reasoning about a MAS, one often makes statements such as: a merchant agent cannot commit to deliver requested goods to a customer until the customer agent commits to send the agreed payment to the merchant. A logic of commitments formalizes such reasoning. In Chapter 5, we defined a language CTLC for reasoning about such systems. The language is that of the multi-agent social commitment logic defined by enriching CTL with the addition of two new modal operators $C_{i \to j}$ and Fu for representing and reasoning about commitment and fulfillment respectively. Here we recall the syntax of CTLC:

$$\varphi ::= \; p \mid \neg\varphi \mid \varphi \vee \varphi \mid E \bigcirc \varphi \mid E(\varphi \; U \; \varphi) \mid E\Box\varphi \mid C_{i \to j}\varphi \mid Fu(C_{i \to j}\varphi)$$

where $p \in \mathcal{PV}$ is an atomic proposition, $\bigcirc$, $\Box$ and U stand for "next time", "globally" and "until" respectively, and E is the existential quantifier on paths. The semantics of the language is presented via the satisfaction relation $\models$, which holds between pair of the form $(\mathcal{M}_C, s)$ and formula of the language. Here, s is a global state, and $\mathcal{M}_C$ is a structure $(S, I, R_t, \{\sim_{i \to j} \mid (i,j) \in \mathcal{A}^2\}, \mathcal{V})$. This structure is called interpreted system satisfying technical apparatus of Kripke-like structure.

The plan of this chapter is organized as follows. In Section 6.2, we develop a new symbolic algorithm to perform model checking CTLC. An obvious question to ask is what is the complexity results of developing this algorithm. In Section 6.3, we investigate the time complexity of CTLC model checking in explicit models (i.e., Kripke-like structures) and its space complexity in concurrent programs, which provide compact representations. We prove that: (1) the time complexity of CTLC model checking for explicit models is P-complete with regard to the size of the model and length of the formula; and (2) the space complexity of the same problem for concurrent programs is PSPACE-complete with respect to the size of the program's components. Consequently, our model checking algorithm has the same complexity as model checking CTL with regard to both explicit models [115] and concurrent programs [86]. The motivation behind considering the complexity of model checking for concurrent programs is that explicit representations are not feasible in practice, by actual model checking systems. In fact, practical model checkers such as MCMAS (for CTL), NuSMV (for CTL and LTL), SPIN (for LTL), and CWB-NC (for CTL*) have the flavour of succinct model specification languages that differ on details and provide the verifier with a relatively high-level way of defining concurrent programs. These programs are composed of n concurrent processes P_i where each process is described by a transition system (the formal definition is given later in Section 6.3). Those processes are implemented as agents or modules depending on the model checker. More precisely, the Kripke-like structures that are used to model the systems' configurations are in fact obtained by making several components interact altogether, leading to a combinatorial explosion of the number of possible configurations, so the need for compact representations. In such representations, states and transitions are not enumerated explicitly as global states and transitions for the system, but only local states and transitions of each component are represented, so that the actual system can still be represented by combining local states and transitions to build reachable states. In general, the relation between explicit models, which are very large and concurrent programs, which provide compact representations of the systems to be checked, is as stated in [86]: "the Kripke structures to which model checking is applied are often obtained by constructing the reachability graph of concurrent programs". In other terms, the explicit models are obtained as the product of the components P_i of concurrent programs. The size of explicit models is thus exponential in the size of processes P_i as the system's evolution results from joint actions of the components [79]. In Section 6.4, we show how the proposed algorithm is fully implemented on top of the MCMAS model checker [92] to directly verify communicative commitments and their fulfillment and violation. To check the effectiveness of our approach, we report on the experimental results of verifying a concrete industrial case study, called Insurance Claim Processing, which is widely used in the literature.

6.2 Symbolic Model Checking CTLC Formulae

Symbolic approaches have been recently proven as an efficient technique to automatically verify MASs [90, 92, 93, 52]. Hereafter, we adopt those approaches because: (1) they use less memory than automata-based approaches; and (2) their algorithms are applied to Boolean functions not to Kripke-like structures. In practice, space requirements for Boolean functions that can be easily encoded in OBDDs [21] are exponentially smaller than for explicit representations. As a result, symbolic approaches alleviate the state explosion problem, but cannot eliminate it totally as the space still increases when the model is getting larger.

As mentioned in Chapter 2, symbolic model checking techniques alleviate the state explosion problem by computing the set $[\![\varphi]\!]$ of states satisfying φ in the model $\mathcal{M}_C = (S, I, R_t, \{\sim_{i\to j} \mid (i,j) \in \mathcal{A}^2\}, \mathcal{V})$, which is represented as Boolean function that can be in turn encoded in OBDD, and then comparing it against the set I of initial states in $\mathcal{M}_C$ that is also encoded in OBDD. If $I \subseteq [\![\varphi]\!]$, then the model $\mathcal{M}_C$ satisfies the formula; otherwise a counterexample is generated. By reducing the problem of symbolic model checking into a comparison of Boolean functions, we proceed as follows. The number $nv(i)$ of Boolean variables required to encode the local states of an agent i is $nv(i) = \lceil log_2|L_i|\rceil$. Similarly, to encode the agent actions, the number $na(i)$ of Boolean variables w_i required is $na(i) = \lceil log_2|Act_i|\rceil$. Thus, a global state $s \in S$ can be encoded as a vector of Boolean variables $(v_1, \ldots, v_N)$, where $N = \sum_{\forall i \in \mathcal{A}} nv(i)$. A joint action $a \in ACT$ can be encoded as a vector of Boolean variables $(w_1, \ldots, w_M)$, where $M = \sum_{\forall i \in \mathcal{A}} na(i)$. Every Boolean vector can be identified with a Boolean formula, represented by a conjunction of Boolean variables or their negation. For instance, the Boolean vector $(1, 1, 0, 1)$ is identified with the Boolean function $f(v_1, v_2, v_3, v_4) = v_1 \wedge v_2 \wedge \neg v_3 \wedge v_4$, where x_i is a Boolean variable and the formula $v_1 \wedge v_2 \wedge \neg v_3 \wedge v_4$ is called Boolean formula. In this way, the set of global states (or actions) can be identified with a Boolean formula by taking the disjunction of all Boolean formulae encoding each global state in the set. Having encoded local states, global states, and actions by means of Boolean formulae, all the remaining parameters can also be expressed as Boolean formulae. Indeed, a local protocol for each agent i can be translated into a Boolean formula and then a Boolean function representing a joint protocol is obtained by taking the conjunction of the Boolean formulae representing the local protocols of agents. The Boolean function $B_{R_t}(s, s')$ representing the temporal transitions can be obtained from the transition relation R_t by quantifying over the Boolean variables encoding joint actions (see details of encoding the transition relation in Chapter 2). This quantification can be translated into a Boolean formula using a disjunction (see [33] for a similar approach to Boolean quantification). The set of initial states and the set of states in which the atomic propositions hold can be easily translated into Boolean formulae. In addition to the parameters presented above, the BDD-based algorithms presented below require the definition of n Boolean functions $B_{\sim_{i\to j}}(s, s')$ (one for each pair of interacting agents) representing the accessibility relations for the commitment operator. The Boolean function $B_{\sim_{i\to j}}(s, s')$ is defined in a similar way as the Boolean function $B_{R_t}(s, s')$ with respect to Definition 7 introduced in Chapter 5.

The standard symbolic model checking algorithm SMC presented in Section 2.3 in Chapter 2 for CTL can be applied to compute the set $[\![\varphi]\!]$ of states in $\mathcal{M}_C$ satisfying the formula φ. The symbolic model checking algorithm $ESMC$ that extends SMC with BDD-based algorithms of the new modalities in our logic (lines 7 and 8) is reported in Algorithm 3. Notice that all operations on sets of global states appearing in Algorithm 3 may be translated into operations on Boolean functions that can be effectively manipulated by making use of OBDDs. The standard procedures $ESMC_{E\bigcirc}(\varphi_1, \mathcal{M}_C)$ (line 4), $ESMC_{EU}(\varphi_1, \varphi_2, \mathcal{M}_C)$ (line 5) and $ESMC_{E\Box}(\varphi_1, \mathcal{M}_C)$ (line 6) are introduced in [33, 78].

Algorithm 3 $ESMC(\varphi, \mathcal{M}_C)$: the set $[\![\varphi]\!]$ satisfying the CTLC formula φ

```
1: $\varphi$ is an atomic formula: return $\mathcal{V}(\varphi)$;
2: $\varphi$ is $\neg\varphi_1$: return $S \backslash ESMC(\varphi_1, \mathcal{M}_C)$;
3: $\varphi$ is $\varphi_1 \vee \varphi_2$: return $ESMC(\varphi_1, \mathcal{M}_C) \cup ESMC(\varphi_2, \mathcal{M}_C)$;
4: $\varphi$ is $E \bigcirc \varphi_1$: return $ESMC_{E\bigcirc}(\varphi_1, \mathcal{M}_C)$;
5: $\varphi$ is $E(\varphi_1 \; U \; \varphi_2)$: return $ESMC_{EU}(\varphi_1, \varphi_2, \mathcal{M}_C)$;
6: $\varphi$ is $E\Box\varphi_1$: return $ESMC_{E\Box}(\varphi_1, \mathcal{M}_C)$;
7: $\varphi$ is $C_{i\to j}\varphi_1)$: return $ESMC_c(i, j, \varphi_1, \mathcal{M}_C)$;
8: $\varphi$ is $Fu(C_{i\to j}\varphi_1))$: return $ESMC_{fu}(i, j, \varphi_1, \mathcal{M}_C)$.
```

6.2.1 BDD-based Algorithm of Commitment

Algorithm 4 reports the procedure for the social commitment modality, which returns the set of states satisfying $C_{i\to j}\varphi$, i.e., $[\![C_{i\to j}\varphi]\!]$. First, the algorithm computes the set X of states satisfying $\neg\varphi$; then constructs the set Y as a subset of X satisfying the fact that all the shared variables between i and j have the same values; then builds the set Z of states that are indistinguishable for i from the states in Y and indistinguishable for j from the same states in Y with regard to the unshared variables with i. In a nutshell, the set Z includes every state that can "see" by means of the social accessibility relation $\sim_{i\to j}$ a state satisfying $\neg\varphi$. Finally, the procedure returns the complement of the set Z.

Algorithm 4 $ESMC_c(i, j, \varphi, \mathcal{M}_C)$: the set $[\![C_{i\to j}\varphi]\!]$

```
1: $X \leftarrow ESMC(i, j, \neg\varphi, \mathcal{M}_C)$;
2: $Y \leftarrow \{s \in X \mid l_i^v(s) = l_j^v(s) \;\; \forall v \in Var_i \cap Var_j\}$;
3: $Z \leftarrow \{s \in S \mid \exists s' \in Y \text{ such that } l_i(s) = l_i(s') \text{ and } l_j^w(s) = l_j^w(s') \;\; \forall w \in Var_j - Var_i\}$;
4: return $S - Z$.
```

We can calculate the states satisfying $\widehat{C}_{i\to j}\varphi$ by calculating the complement of the set of states satisfying $C_{i\to j}(\neg\varphi)$ using Algorithm 4.

6.2.2 BDD-based Algorithm of Fulfillment

The procedure $ESMC_{fu}(i, j, \varphi, \mathcal{M}_C)$ (see Algorithm 5) starts by computing the set X of states satisfying the commitment $C_{i\to j}\varphi$. It then constructs the set Y of states that are indistinguishable

Algorithm 5 $ESMC_{fu}(i, j, \varphi, \mathcal{M}_C)$: the set $[\![Fu(C_{i\to j}\varphi)]\!]$

```
1: $X \leftarrow ESMC_c(i, j, \varphi, \mathcal{M}_C)$;
2: $Y \leftarrow \{s \in S \mid \exists s' \in X \text{ such that } l_i(s) = l_i(s') \text{ and } l_j^w(s) = l_j^w(s') \; \forall w \in Var_j - Var_i\}$;
3: $Z \leftarrow \{s \in Y \mid l_i^v(s) = l_j^v(s) \;\; \forall v \in Var_i \cap Var_j\}$;
4: return $Z$.
```

for i, and indistinguishable for j with regard to unshared variables with i, from the states in X. Afterwards, the algorithm builds the set Z as a subset of the set Y including only states where the shared variables between i and j have the same values. Finally, the procedure returns the set Z. Simply put, the algorithm computes and returns the set Z of accessible states that are "seen" by means of the social accessibility relation $\sim_{i\to j}$ from a state in X.

The correctness of Algorithm 3 is established by the correctness of: (1) the model checking procedures for temporal operators, which in fact derives from the correctness of the procedures employed in the verification of standard CTL models [33, 78]; and (2) commitment and fulfillment operators using the following lemma whose the proof is straightforward from the calculation steps of Algorithms 4 and 5.

Lemma 2 *When Algorithm 4 terminates*[2]*, we have that for every $s' \in S$ of the form $s \sim_{i\to j} s'$, $(\mathcal{M}_C, s) \models C_{i\to j}\varphi$ iff $(\mathcal{M}_C, s') \models \varphi$. Also, when Algorithm 5 terminates, then there exists $s' \in S$ of the form $s \sim_{i\to j} s'$, $(\mathcal{M}_C, s') \models Fu(C_{i\to j}\varphi)$ iff $(\mathcal{M}_C, s) \models C_{i\to j}\varphi$*

6.3 Complexity Analysis

Researchers usually analyze the complexity of model checking of their logics in order to back the usefulness of the proposal with a formal argument and to compute resources necessary and sufficient for solving all problem's instances, including the hardest (worst) case. The theoretical complexity results conduct us to a clearer understanding of why model checking works well (or does not work) [115]. They also give us a meaningful picture of the actual computational difficulty behind the problem and allow us to compare different model checking approaches and logical formalisms. It is possible to establish a hierarchy for complexity classes, comparing both time and space classes. The inclusions of the most common used complexity classes are defined as follows: L $\subseteq$ NL $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ NPSPACE $\subseteq$ EXP, which are respectively read as *logarithmic space, nondeterministic logarithmic space, polynomial time, nondeterministic polynomial time, polynomial space, nondeterministic polynomial space*, and *exponential time.* The complexity of model checking is always relative to the size of the model and formula representation that we use. For this reason, it is necessary to state how we represent the input and how we measure its size.

[2]Our procedures reported in Algorithms 4 and 5 terminate successfully because: (1) since we implement them on top of the MCMAS model checker [92], then the set of reachable states S employed in such procedures and easily computed by a monotonic operator has a fix-point; and (2) they have the same structure like the temporal operator $A\bigcirc$. A similar arguments apply to prove the termination and correctness of the procedures employed for epistemic operators, which have the same structure used in MCMAS.

In this section, we will first analyze the time complexity of model checking CTLC with regard to the size of the model $\mathcal{M}_C$ and length of the formula to be checked, under the assumption that states and transitions are listed explicitly. Thereafter, we will analyze the space complexity of model checking CTLC for concurrent programs with respect the size of the components of these programs and length of the formula.

6.3.1 Time Complexity

We here start by proving that model checking CTLC is P-complete, so it can be done in polynomial running time in the size of the model and length of the formula.

Theorem 4 *The model checking problem for CTLC can be solved in time* $\mathcal{O}(|\mathcal{M}_C| \times |\psi|)$ *where* $|\mathcal{M}_C|$ *and* $|\psi|$ *are the size of the model and length of the formula respectively.*

Proof 6

CTLC extends CTL, and it is known from [32] that the model checking problem for CTL is linear in the size of the model and length of the formula. We just need to analyze the time complexity of Algorithms 4 and 5. Steps 2, 3, and 4 in these two algorithms are simple and it is easy to see that they can be done in linear running time in the size of the model as they are simply constructing sets by performing comparison operations on states. Step 1 in Algorithm 4 calls the model checking procedure recursively on the subformula φ *of the formula* $\psi = C_{i \to j}\varphi$*. The algorithm is recursively called till a CTL subformula is encountered. Thus, the depth of the recursion is bounded by the length of the formula* ψ *(i.e., linear in the length* $|\psi|$*). As again model checking CTL is linear in both the size of the model and length of the formula, we conclude that this algorithm has the same complexity. As Algorithm 5 is simply calling Algorithm 4, the result follows.* ■

Theorem 5 *The model checking problem for CTLC is P-complete.*

Proof 7

`Membership` *in P (i.e., upper bound) follows from Theorem 4.* `Hardness` *in P (i.e., lower bound) follows by a reduction from model checking CTL proved to be P-complete in [115].* ■

6.3.2 Space Complexity

Having discussed the motivation behind the consideration of space complexity for concurrent programs in the introduction, hereafter, we will prove that the complexity of CTLC model checking for concurrent programs is PSPACE-complete. Before that, we give the proof idea and technical background about automata-theoretic approach that we will use in our proof.

Proof Idea. To analyze the space complexity of model checking CTLC in concurrent programs, we use a methodology similar to the one presented in [86]. The idea is as follows. First, we analyze the complexity of model checking GCTL* in the explicit model $\mathcal{M}_G$ introduced in Definition 5 in Chapter 5, and show that by using an on-the-fly (local) and top-down algorithm, it is possible to perform model checking GCTL* in space *polynomial* in the length of the formula, but only *poly-logarithmic* in the size of the explicit model. It is important to mention that the algorithm is on-the-fly, which means we do not hold the whole structure to be checked in memory at any one time, and this is the reason behind the poly-logarithmic space complexity in the size of the explicit model. As in [86], our approach is an automata-theoretic approach, and makes use of a special class of automata called *Alternating Büchi Tableau Automata* (ABTA) [12, 19], which will be introduced later. The approach is based on building an ABTA, combining the model $\mathcal{M}_G$ and the automaton of the formula to be verified, and checking its nonemptiness. This combined ABTA is computed on-the-fly and limited to its reachable states, which avoids exploring the parts of the model $\mathcal{M}_G$ that are irrelevant for the formula to be checked. The type of ABTA employed allows using a top-down, space-efficient model checking algorithm. Then, we prove that the explicit structure complexity of GCTL* model checking (i.e., by fixing the formula) is NLOGSPACE-complete, which means that model checking GCTL* is NLOGSPACE-complete in the size $|\mathcal{M}_G|$ of the explicit model. Thereafter, we use the previous results to obtain the complexity of model checking GCTL* for concurrent programs, exploiting the fact that the combined ABTA whose the nonemptiness has to be checked is obtained as the product of the components of a concurrent program and this product is at most exponentially larger than the program itself. Thus, the fact that: (1) the space complexity of model checking GCTL* is polynomial in the length of the formula and poly-logarithmic in the size of the explicit model; and (2) the model checking algorithm is on-the-fly, imply that GCTL* model checking for a concurrent program can be done in polynomial space with respect to the size of this program rather than of the order of the exponentially larger combined ABTA as is the case of bottom-up approaches to model checking. By logspace reduction to GCTL* model checking with respect to explicit models, we analyze the explicit structure complexity of CTLC model checking and prove that is NLOGSPACE-complete, which, as the case of GCTL*, implies that CTLC model checking can be done in polynomial space with respect to the size of concurrent programs.

A) Preliminaries

We start this subsection by defining ABTA and associated concepts needed to analyze the complexity of model checking GCTL*. Definition of concurrent programs will follow. We use $\mathcal{L} = \mathcal{PV} \cup \{\neg p \mid p \in \mathcal{PV}\}$ to denote the set of state literals and $\mathcal{L}_{act} = \Phi_b \cup \{\neg\theta \mid \theta \in \Phi_b\}$ to denote the set of action literals. Let Θ be a typical subset of $\mathcal{L}_{act}$. An ABTA is defined as follows [19]:

Definition 12 (ABTA) *An ABTA $\mathcal{B}$ is a tuple $(Q, h, \rightarrow_B, q_I, \mathcal{F})$, where Q is a finite set of states; $h : Q \rightarrow \mathcal{L} \cup \{\neg, \wedge, \vee, [\Theta], \langle\Theta\rangle\}$ is the state labeling function; $\rightarrow_B \subseteq Q \times Q$ is the transition relation; $q_I \in Q$ is the start state; and $\mathcal{F} \subseteq 2^Q$ is the set of sets of accepting states. $\rightarrow_B$ should also satisfy:*

$$|\{q'|q \rightarrow_B q'\}| \begin{cases} = 0 & \textit{if } h(q) \in \mathcal{L} \\ = 1 & \textit{if } h(q) \in \{\neg, [\Theta], \langle\Theta\rangle\} \\ \geq 1 & \textit{if } h(q) \in \{\wedge, \vee\} \end{cases}$$

Also if $h(q) = \neg$, then q does not appear on a cycle.

ABTAs have the advantage of supporting efficient model checking for different logics and are used to define the system properties using tableau proof rules [18]. They are used to encode how the properties are to be proved and allow us to encode top-down proofs for temporal formulae (GCTL* formulae in this case). Indeed, an ABTA $\mathcal{B}$ encodes a proof schema in order to prove, in a goal-directed manner, that a model $\mathcal{M}_G$ satisfies a temporal formula. Let us consider the example of proving that a state s in a model $\mathcal{M}_G = (S_G, Ac, l_S, l_{Ac}, \rightarrow, I_G)$ satisfies a temporal formula of the form $F_1 \wedge F_2$, where F_1 and F_2 are two formulae. Regardless of the structure of the system, there would be two subgoals if we want to prove this in a top-down, goal-directed manner. The first would be to prove that s satisfies F_1, and the second would be to prove that s satisfies F_2. Intuitively, an ABTA for $F_1 \wedge F_2$ would encode this proof structure using states for the formulae $F_1 \wedge F_2$, F_1, and F_2. A transition from $F_1 \wedge F_2$ to each of F_1 and F_2 should be added to the ABTA and the labeling of the state for $F_1 \wedge F_2$ being "$\wedge$". Indeed, in an ABTA, we can consider that: (1) states correspond to "formulae"; (2) the labeling of a state is the "logical operator" used to construct the formula or a state literal from $\mathcal{L}$; and (3) the transition relation represents a "subgoal" relationship. Thus, to show that a model state s satisfies an ABTA state q labeled with $\wedge$, one needs to show that s satisfies each of q's children. Regarding the labels $[\Theta]$ and $\langle\Theta\rangle$, for a model state s to satisfy an ABTA state q labeled with $[\Theta]$ (resp. $\langle\Theta\rangle$), one needs to show that for each s' (resp. some s') such that $(s, \alpha, s') \in\rightarrow$ for some α "satisfying" Θ (i.e. $\theta \in l_{Ac}(\alpha)$ for every $\theta \in \Theta$), s' must satisfy the unique successor of q.

In order to decide about the satisfaction of formulae, the notion of accepting runs of an ABTA $\mathcal{B}$ on a model $\mathcal{M}_G$ is used. These runs are infinite and cycle infinitely many times through accepting states. Formally, a run is defined as follows, where the notation $(s, \Theta, s') \in\rightarrow$ means $(s, \alpha, s') \in\rightarrow$ for some $\alpha \in Ac$ satisfying Θ:

Definition 13 (Run of ABTA) *A run of an ABTA $\mathcal{B} = (Q, h, \rightarrow_B, q_I, \mathcal{F})$ on a model $\mathcal{M}_G = (S_G, Ac, l_S, l_{Ac}, \rightarrow, I_G)$ is a maximal tree in which the nodes are classified as positive or negative and are labeled by elements of $Q \times S_G$ as follows.*

- *The root of the tree is a positive node and is labeled with (q_I, i_G) where $i_G \in I_G$*
- *If σ is a positive (resp. negative) node with label (q, s) such that $h(q) = \neg$ and $q \rightarrow_B q'$, then σ has one negative (resp. positive) child labeled (q', s)*
- *For a positive node σ labeled with (q, s):*
  - *If $h(q) \in \mathcal{L}$ then σ is a leaf.*
  - *If $h(q) = \wedge$ and $\{q' | \ q \rightarrow_B q'\} = \{q_1, \ldots, q_m\}$, then σ has m positive children labeled by (q_i, s), $1 \leq i \leq m$.*
  - *If $h(q) = \vee$, then σ has one positive child*[3] *labeled by (q', s) for some $q' \in \{q' | \ q \rightarrow_B q'\}$.*
  - *If $h(q) = [\Theta]$, q' is such that $q \rightarrow_B q'$, and $\{s' | \ (s, \Theta, s') \in\rightarrow\} = \{s_1, \ldots, s_m\}$, then σ has m positive children labeled by (q', s_i), $1 \leq i \leq m$.*
  - *If $h(q) = \langle\Theta\rangle$ and q' is such that $q \rightarrow_B q'$, then σ has one positive child labeled by (q', s') for some s' such that $(s, \Theta, s') \in\rightarrow$.*
- *Otherwise, for a negative node σ labeled with (q, s):*
  - *If $h(q) \in \mathcal{L}$ then σ is a leaf.*
  - *If $h(q) = \wedge$, then σ has one negative child*[4] *labeled by (q', s) for some $q' \in \{q' | \ q \rightarrow_B q'\}$.*
  - *If $h(q) = \vee$ and $\{q' | \ q \rightarrow_B q'\} = \{q_1, \ldots, q_m\}$, then σ has m negative children labeled by (q_i, s), $1 \leq i \leq m$.*
  - *If $h(q) = [\Theta]$ and q' is such that $q \rightarrow_B q'$, then σ has one negative child labeled by (q', s') for some s' such that $(s, \Theta, s') \in\rightarrow$.*
  - *If $h(q) = \langle\Theta\rangle$, q' is such that $q \rightarrow_B q'$, and $\{s' | \ (s, \Theta, s') \in\rightarrow\} = \{s_1, \ldots, s_m\}$, then σ has m negative children labeled by (q', s_i), $1 \leq i \leq m$.*

Every infinite path in a well-formed ABTA has a suffix that contains either only positive or only negative nodes [19]. If only positive (resp. negative) nodes are included, the path is said to be positive (resp. negative). A successful run is then defined as follows:

[3]We only consider one positive child in a run when the node is positive and disjunctive (i.e., labeled by $\vee$ or $\langle\Theta\rangle$) as only one branch in the product graph (see Definition 16) is chosen.

[4]Similar to the positive case, we only consider one negative child in a run when the node is negative and conjunctive (i.e., labeled by $\wedge$ or $[\Theta]$) because again only one branch in the product graph is selected.

Definition 14 (Successful run of ABTA) *Let R be a run of an ABTA $\mathcal{B}$ on a model $\mathcal{M}_G$.*

– *A positive leaf of R labeled (q, s) is successful iff s satisfies $h(q)$ or $h(q) = [\Theta]$ and there is no s' such that $(s, \Theta, s') \in \rightarrow$.*

– *A negative leaf of R labeled (q, s) is successful iff s does not satisfy $h(q)$ or $h(q) = \langle\Theta\rangle$ and there is no s' such that $(s, \Theta, s') \in \rightarrow$.*

– *A positive path is successful iff for each $F \in \mathcal{F}$ some $q \in F$ occurs infinitely often.*

– *A negative path is successful iff for some $F \in \mathcal{F}$ there is no $q \in F$ that occurs infinitely often.*

R is successful iff every leaf and every path in R is successful.

Model Checking GCTL*. Let ψ be a GCTL* state formula. The automata-based model checking procedure for GCTL* proposed in [19] works as follows:

1. Translating ψ into a variant of ABTA: and-restricted Alternating Büchi Tableau Automaton (arABTA). The resulting automaton is denoted by $\mathcal{B}_\psi$.
2. Exploring the product graph of $\mathcal{M}_G$ and $\mathcal{B}_\psi$ to check if it contains a successful run. If such a run does exist, the formula is satisfied and $\mathcal{M}_G$ is said to be accepted by $\mathcal{B}_\psi$ (i.e., $\mathcal{M}_G \models \mathcal{B}_\psi$), otherwise, the formula is not satisfied. The product graph is denoted by $\mathcal{B}_{\mathcal{M}_G,\psi}$.

Definition 15 (arABTA) *An ABTA $\mathcal{B}$ is and-restricted (arABTA) iff every state $q \in Q$ satisfies:*

– *If $h(q) = \wedge$ then there is at most one q' such that $q \rightarrow_B q'$ and there is a path from q' back to q.*

– *If $h(q) = [\Theta]$ and $q \rightarrow_B q'$, then there is no path from q' back to q.*

In an arABTA, the strongly-connected component of a state labeled by $\wedge$ can contain *at most* one of its state's children and a state labeled by $[\Theta]$ is guaranteed to belong to a different strongly-connected component that its child. Thanks to this restrictedness, the handling of recursive children (i.e., children where there is a path from them back to the parents) is simplified, particularly when space is concerned. This makes simple the treatment of recursive calls needed for some GCTL* formulae, which allows for space-efficient model checking this logic (this will be made clear later when we will analyze the complexity of model checking GCTL*). As argued in [19], arABTAs play the same role in model checking GCTL* that do *Hesitant Alternating word Automata* (HAAs) in model checking CTL and CTL* [86] although the two automata are conceptually different[5]. In

[5]arABTA would be hesitant if for every strongly-connected component $Q_i \subseteq Q$ and every node $q \in Q_i$ either $h(q) \in \{\wedge, [\Theta]\}$ or $h(q) \in \{\vee, \langle\Theta\rangle\}$. Details about HAAs are out of scope of this paper and interested reader can refer to [86].

fact, it has been shown that HAAs are the key to the space-efficient model checking algorithms for CTL and CTL* thanks to their restricted alternation structure (every nontrivial[6] strongly-connected component of HAAs is either (1) existential, so contains only nodes that are disjunctively related; or (2) universal, so contains only nodes that are conjunctively related). In this chapter, we will show that arABTAs are the key to the space-efficient complexity of the problem of model checking GCTL* and to the NLOGSPACE membership of the explicit structure complexity of this model checking problem.

Intuitively, the product graph can be seen as an encoding of all the runs of the arABTA. Formally:

Definition 16 (Product graph) *The product graph $\mathcal{B}_{\mathcal{M}_G,\psi}$ of an arABTA $\mathcal{B}_\psi = (Q, h, \rightarrow_B, q_I, \mathcal{F})$ and a model $\mathcal{M}_G = (S_G, Ac, l_S, l_{Ac}, \rightarrow, I_G)$ where $\mathcal{F} = \{F_0, \ldots, F_{n-1}\}$ has vertex set $Ver = Q \times S_G \times \{0, \ldots, n-1\}$ and edges $Edg \subseteq Ver \times Ver$. The edges are defined by: $((q, s, i), (q', s', i')) \in Edg$ iff*

- *there exist nodes σ and σ' in some run of $\mathcal{B}_\psi$ on $\mathcal{M}_G$ labeled (q, s) and (q', s') respectively such that σ' is a child of σ; and*
- *either $q \notin F_i$ and $i' = i$, or $q \in F_i$ and $i' = (i + 1) \mod n$*

A vertex (q, s, i) is said to be accepting iff $q \in F$ for some $F \in \mathcal{F}$ and $i = 0$.

Bhat has proved in [18] that an arABTA can be partitioned uniquely to ordered sets $Q_1, \ldots, Q_n$, which correspond to its strongly-connected components. The number n of these sets is the depth of the arABTA. Finally, we define the sizes of $\mathcal{B}_\psi$, $\mathcal{M}_G$, and $\mathcal{B}_{\mathcal{M}_G,\psi}$ as follows:

- $|\mathcal{B}_\psi| = |Q| + |\mathcal{F}| + |\rightarrow_B|$ where $|Q|$ and $|\rightarrow_B|$ are the respective cardinalities of the sets Q and $\rightarrow_B$, and $|\mathcal{F}|$ is the number of component sets in $\mathcal{F}$.
- $|\mathcal{M}_G| = |S_G| + |Ac| + |\rightarrow|$.
- $|\mathcal{B}_{\mathcal{M}_G,\psi}| = |Ver| + |Edg|$, where the vertex set is bounded in size by $|Q| \cdot |S_G| \cdot |\mathcal{F}|$.

Concurrent programs. Let us now define concurrent programs. As introduced in [86], a concurrent program Pr is composed of n concurrent processes. Each process P_i is described by a transition system D_i defined as follows: $D_i = (AP_i, AC_i, S_i, \Delta_i, s_i^0, H_i)$ where AP_i is a set of local atomic propositions, AC_i is a local action alphabet, S_i is a finite set of local states, $\Delta_i \subseteq S_i \times AC_i \times S_i$ is a local transition relation, $s_i^0 \in S_i$ is an initial state, and $H_i : S_i \rightarrow 2^{AP_i}$ is a local state labeling function. A concurrent behavior of these processes is obtained by the product of the processes and

[6]A strongly-connected component Q_i is nontrivial if $|Q_i| > 1$ or $Q_i = \{q\}$ and q has a self loop.

transition actions that appear in several processes are synchronized by common actions. The joint behavior of the processes P_i can be described using a global transition system D, which is computed by constructing the reachable states of the product of the processes P_i and synchronization is obtained using common action names. Let $AP = \bigcup_{i=1}^{n} AP_i$,
$AC = \bigcup_{i=1}^{n} AC_i$,
$S = \Pi_{i=1}^{n} S_i$,
$s^0 = (s_1^0, s_2^0, \dots, s_n^0)$,
$H(s) = \bigcup_{i=1}^{n} H_i(s[i])$ for every $s \in S$, and $s[i]$ be the ith component of s. Thus, $D = (AP, AC, S, \Delta, s^0, H)$ where $(s, a, s') \in \Delta$ iff $(s[i], a, s'[i]) \in \Delta_i$ or $s[i] = s'[i]$ for all $1 \leq i \leq n$.

B) Space complexity of GCTL*

Hereafter, we prove two results: (1) the explicit structure complexity of GCTL* model checking (i.e. by fixing the formula) is NLOGSPACE-complete; and (2) model checking GCTL* for concurrent programs with respect to the size of the components P_i and the length of the formula being checked is PSPACE-complete.

Let $cl(\psi)$ be the closure of ψ defined as the smallest set such that the following hold:

- $\psi \in cl(\psi)$
- If $\neg\psi' \in cl(\psi)$ then $\psi' \in cl(\psi)$
- If $\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2 \in cl(\psi)$ then $\psi_1, \psi_2 \in cl(\psi)$
- If $E(\psi') \in cl(\psi)$ then $\psi' \in cl(\psi)$
- If $A(\psi') \in cl(\psi)$ then $E(\neg\psi') \in cl(\psi)$
- If $E(\psi_1 \wedge \psi_2) \in cl(\psi)$ then $E(\psi_1, \psi_2) \in cl(\psi)$
- If $E(\psi_1 \vee \psi_2) \in cl(\psi)$ then $E(\psi_1), E(\psi_2) \in cl(\psi)$
- If $E(\psi_1 \; U \; \psi_2) \in cl(\psi)$ then $\psi_1, \psi_2, E \bigcirc (\psi_1 \; U \; \psi_2) \in cl(\psi)$
- If $E \bigcirc \psi' \in cl(\psi)$ then $\psi' \in cl(\psi)$

We start by presenting the following proposition from [18] and [12], where $|\psi|$ denotes the length of the formula ψ measured as the number of elements in $cl(\psi)$.

Proposition 6 *Let ψ be a GCTL* state formula and $|\psi|$ be its length.*

1. $|\mathcal{B}_{\mathcal{M}_G,\psi}| = \mathcal{O}(|\mathcal{M}_G| \cdot |\mathcal{B}_\psi|)$
2. $|\mathcal{B}_\psi| = \mathcal{O}(2^{|\psi|})$

Theorem 6 *Given a GCTL* formula ψ, we can construct an arABTA $\mathcal{B}_\psi$ of size $\mathcal{O}(2^{|\psi|})$ and of depth $\mathcal{O}(|\psi|)$ such that every state in I_G, the set of initial states of $\mathcal{M}_G$, satisfies ψ iff $\mathcal{M}_G \models \mathcal{B}_\psi$.*

Proof 8

Assuming that $\mathcal{B}_\psi$ is an arABTA, which will be shown later in this proof, the first part of the theorem is simply from Proposition 6. For the second part, the algorithm to generate arABTAs from GCTL* formulae uses goal-directed rules aiming to build tableaux from formulae. The formulae have the form $E(\Phi)$ and $A(\Phi)$ where Φ is a set of path formulae, so $E(\Phi)$ denotes $E(\bigwedge_{\varphi\in\Phi}\varphi)$, $A(\Phi)$ denotes $A(\bigvee_{\varphi\in\Phi}\varphi)$, and $E(\neg\Phi)$ denotes $E(\bigvee_{\varphi\in\Phi}\neg\varphi)$. Furthermore, we write $E(\Phi,\psi)$ to represent the formula of the form $E(\Phi\cup\{\psi\})$ and similarly $E(\Phi,\psi_1,\ldots,\psi_n)$ represents $E(\Phi\cup\{\psi_1\ \ldots,\psi_n\})$, where $\psi,\psi_1,\ldots\psi_n$ are path formulae and Φ is possibly empty. When Φ is empty, we can also write $E(\psi_1,\psi_2)$ to represent $E(\psi_1\wedge\psi_2)$, which also represents $E(\{\psi_1\}\cup\{\psi_2\})$. To build an arABTA $\mathcal{B}_\psi$ from a state formula $\psi\equiv E(\Phi)$, one first generates the states and transitions. The initial state is the formula ψ itself and in general, states correspond to state formulae and transitions are linking formulae to their subformulae as defined by the closure of these formulae. The subformulae (in the sense of the closure) are obtained by applying the tableau rules shown in Table 15 in the order R1 to R10, where the top part of each rule being the goal and the bottom part being the subgoals. Assuming a state already associated with a formula ψ, one applies the rules R1-R10 to generate new states by comparing the form of ψ with the formula in the goal part of the rules starting from R1. When ψ and the goal formula match, the label of the rule becomes the label of the state and the subgoal formulae obtained from the rule are added as states and transitions from ψ to these states are added. Leaves are labeled by the state literals and the process stops when no new states are added. The soundness and termination of this algorithm are presented in [19].

Table 15: Tableau rules for GCTL*

R1 $\wedge : \frac{\psi_1\ \wedge\ \psi_2}{\psi_1\quad \psi_2}$	*R2* $\vee : \frac{\psi_1\ \vee\ \psi_2}{\psi_1\quad \psi_2}$	*R3* $\vee : \frac{E(\psi)}{\psi}$	*R4* $\neg : \frac{\neg\psi}{\psi}$	*R5* $\neg : \frac{A(\Phi)}{E(\neg\Phi)}$
R6 $\wedge : \frac{E(\Phi,\psi)}{E(\Phi)\quad E(\psi)}$	*R7* $\wedge : \frac{E(\Phi,\varphi_1\wedge\varphi_2)}{E(\Phi,\varphi_1,\varphi_2)}$	*R8* $\vee : \frac{E(\Phi,\varphi_1\vee\varphi_2)}{E(\Phi,\varphi_1)\ E(\Phi,\varphi_2)}$		
R9 $\vee : \frac{E(\Phi,\varphi_1\ U\ \varphi_2)}{E(\Phi,\varphi_2)\ E(\Phi,\varphi_1,\bigcirc(\varphi_1\ U\ \varphi_2))}$		*R10* $\langle\Psi_1\rangle : \frac{E(\Psi,\bigcirc\varphi_1,\ldots,\bigcirc\varphi_n)}{E(\varphi_1,\ldots,\varphi_n)}$		

Ψ is an ordered set of action literals and Ψ_1 is a subset of Ψ containing only the first element of Ψ

Let us now show that the obtained automaton $\mathcal{B}_\psi$ is an arABTA. We start first by proving that $\mathcal{B}_\psi$ is an ABTA. From the tableau rules R1-R10 and the explanation above, we can see that states are labeled by a subset of $\{\neg,\wedge,\vee,[\Theta],\langle\Theta\rangle\}$, and: (1) leaves (states without children) are labeled by elements of $\mathcal{L}$; (2) states labeled by $\{\neg,\langle\Theta\rangle\}$ have only one child (rules R4, R5, and R10); (3) states labeled by $\{\wedge,\vee\}$ have at least one child (rules R1, R2, R3, R6, R7, R8, and R9); and (4) *states labeled by $\neg$ using rules R4 and R5 do not appear on a cycle as there are no rules linking ψ to $\neg\psi$ or $E(\neg\Phi)$ to $A(\Phi)$ since $\neg\psi$ and $A(\Phi)$ do not appear as subgoal formulae in any of the rules. Consequently, $\mathcal{B}_\psi$ satisfies all the conditions of ABTAs (Definition 12), so it is an ABTA. To show that it is an arABTA, we only need to show that (1) states labeled by $[\Theta]$ have no recursive children; and (2) states labeled by $\wedge$ have at most one recursive child, which means in rules labeled by $\wedge$, at most one subgoal can be recursive, so it can include the formula identified in the goal. The first part is straightforward as no state is labeled by $[\Theta]$ in $\mathcal{B}_\psi$. For the second part, three rules should be discussed: R1, R6, and R7. R1 produces two children, but no one is recursive as there are no rules linking ψ_1 or ψ_2 back to $\psi_1 \wedge \psi_2$. R6 generates two children: $E(\psi)$ and $E(\Phi)$, but the child $E(\psi)$ is not recursive as the only available rule to be applied once $E(\psi)$ is obtained is R3, which will generate a state labeled by ψ and from ψ a formula having the form $E(\Phi, \psi)$ cannot be produced. Thus, R6 can produce at most one recursive child, which could be from $E(\Phi)$. Finally, R7 generates only one child, so again at most one is recursive.*

The partition of the obtained arABTA $\mathcal{B}_\psi$ to $Q_1, \ldots, Q_n$ proceeds as follows: $q_I \in Q_n$ and for each state $q \in Q_i$ we have:

- *If $h(q) \in \{\vee, \langle\Theta\rangle\}$ then for every q' such that $q \rightarrow_B q'$, $q' \in Q_j$ and $j \leq i$.*
- *If $h(q) \in \{[\Theta], \neg\}$ then for every q' such that $q \rightarrow_B q'$, $q' \in Q_j$ and $j < i$.*
- *If $h(q) = \wedge$ then there is exactly at most one state q' from the set $\{q' \mid q \rightarrow_B q'\}$ such that $q' \in Q_j$ and $j \leq i$. For the other states q' we have $q' \in Q_j$ and $j < i$.*

Thus, since each state is associated to a subformula as defined in the closure, this partition shows that each subformula (in the sense of the closure) of a formula ψ induces at most one set Q_i in ψ. Therefore, the depth of $\mathcal{B}_\psi$ is linear in $|\psi|$. ■

The following is an example from [18] showing the tableau and arABTA obtained from a given GCTL* formula.

Example 4 *Let $Ac = \{send, receive\}$. Consider the formula $A\Box(send \supset \Diamond(receive))$. The tableau of the formula along with the applied rules are shown in Table 16. The obtained arABTA is depicted in Figure 20. It is worth noticing that in the first application of R9 (to obtain the formulae in row 3 of Table 16), we have in the goal part Φ is empty, $\varphi_1 \equiv true$, and $\varphi_2 \equiv send \wedge G(\neg receive)$. In the subgoal part, we use the form with "," instead of "$\wedge$" for the left side formula. This choice is simply motivated by the fact that the two components (i.e., send and $G(\neg receive)$) become clearly separated, which allows this row (i.e., row 3) to match the goal part of rule R9. By so doing, it*

Table 16: Tableau of $A\Box(send \supset \Diamond(receive))$

$\neg : A\Box(\neg\ send \vee \Diamond(receive))$ ***(R5)***	
$\vee : E\Diamond(send \wedge \Box(\neg\ receive))$ ***(R9)***	
$\vee : E(send, \Box(\neg\ receive))$ ***(R9)***	$\langle true\rangle : E \bigcirc (\Diamond(send \wedge \Box(\neg\ receive)))$ ***(R10)***
$\langle send\rangle : E(send, \neg\ receive, \bigcirc\Box(\neg\ receive))$ ***(R10)***	$E\Diamond(send \wedge \Box(\neg\ receive))$
$\vee : E\Box(\neg\ receive)$ ***(R9)***	
$\langle\neg\ receive\rangle : E(\neg\ receive, \bigcirc\Box(\neg\ receive))$ ***(R10)***	
$E\Box(\neg\ receive)$	

becomes clear that in the second application of R9 (to obtain the formula in row 4), $\Phi = \{send\}$ and $\varphi_1 U \varphi_2 \equiv G(\neg receive)$. Then we apply R10, where $\Psi = \{send, \neg receive\}$ and $\Psi_1 = \{send\}$. For simplicity, we abuse the notation and label the first and second R10 respectively $\langle send\rangle$ and $\langle\neg receive\rangle$ instead of $\langle\{send\}\rangle$ and $\langle\{\neg receive\}\rangle$. The application of the other rules is straightforward.

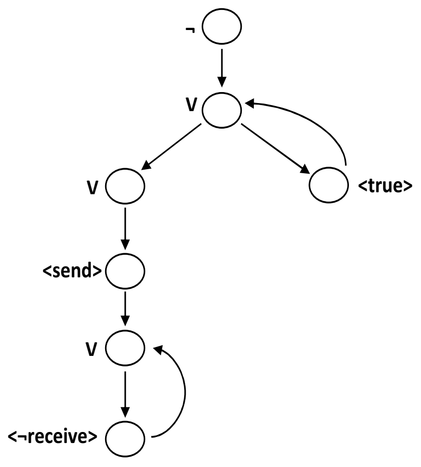


Figure 20: arABTA of $A\Box(send \supset \Diamond(receive))$

Remark 8 *In the tableau rules shown in Table 15, there is a rule labeled by $\langle\Psi_1\rangle$, but no rule is labeled by $[\Psi_1]$. The reason is that those rules are mainly dealing with existential formulae and whenever a universal formula is encountered, it is transformed to an existential one using the rule R5. In fact, a rule labeled by $[\Psi_1]$ would be used to deal with universal formulae having the form $A(\Psi, \bigcirc\varphi_1, \ldots, \bigcirc\varphi_n)$ and the rule would have the form: $[\Psi_1] : \frac{A(\Psi, \bigcirc\varphi_1, \ldots, \bigcirc\varphi_n)}{A(\varphi_1, \ldots, \varphi_n)}$. Having the rule R5, the rule $[\Psi_1]$ would be redundant because applying this new rule followed by R5 (the only rule possible when $A(\varphi_1, \ldots, \varphi_n)$ appears) is equivalent to applying first R5 to $A(\Psi, \bigcirc\varphi_1, \ldots, \bigcirc\varphi_n)$ followed by $\langle\neg\Psi_1\rangle$. In both cases we will end up with $E(\neg\varphi_1, \ldots, \neg\varphi_n)$. Although the new rule is redundant, adding it will still produce arABTAs because there is no rule that can produce $A(\Psi, \bigcirc\varphi_1, \ldots, \bigcirc\varphi_n)$, so states labeled by $[\Psi_1]$ will not have recursive children. On the other hand, notice that it is* *possible to replace the rules R3, R5, R6, R7, R8, R9, and R10 by other rules dealing with universal formulae. In this case, R5 would have the form* $\neg : \frac{E(\Phi)}{A(\neg\Phi)}$ *and the rule* $[\Psi_1]$ *would be used instead of R10. Technically, this means the ABTAs obtained can contain either states labeled by* $\langle\Psi_1\rangle$*, or* $[\Psi_1]$*, but not both, which still complies with the ABTA's definition. However, using rules with universal formulae will not necessarily produce arABTAs because applying the new rule replacing R9 together with the rule* $[\Psi_1]$*, a node labeled by* $[\Psi_1]$ *will have recursive children.*

Theorem 7 *The model checking problem for GCTL* can be solved in space* $\mathcal{O}(|\psi|(|\psi|+log\ |\mathcal{M}_G|)^2)$.

Proof 9

As explained in the preliminaries, the problem of model checking GCTL* is the problem of determining if the product graph $\mathcal{B}_{\mathcal{M}_G,\psi}$ *contains a successful run, which means checking the nonemptiness of the arABTA* $\mathcal{B}_{\mathcal{M}_G,\psi}$*. Here we present the on-the-fly algorithm presented in [19] and then we analyze its space complexity, which has not been done in [19]. The algorithm avoids the storage penalty associated with the construction of strongly-connected components[7] and uses two depth-first searches, `DFS1` and `DFS2`. The algorithm is a top-down marking algorithm. `DFS1` recursively marks nodes as either true or false and `DFS2` is lunched whenever an accepting node is found to check if the node is reachable from itself via nodes not previously traversed by `DFS2`. In fact, the success of `DFS2` means the existence of runs with successful infinite paths. Thus, the motivation behind the requirement for nodes of not being previously traversed by `DFS2` is to avoid unnecessary re-computation of successful paths already found. This is because a node N is already traversed by `DFS2` if it is an accepting state and a recursive child of another accepting state which has been already found by `DFS1` so that the successful infinite path to which N belongs has been already identified. When executing `DFS1`, some nodes are not directly marked true or false, but are marked as **dependant** on their recursive children, which are **previously traversed** by `DFS1` but not marked yet, so they are marked true (resp. false) once the nodes on which they depend are marked true (resp. false). This procedure is called mark propagation and happens in a strongly-connected component because the nodes previously traversed can be marked later by exploring other branches in the same component. Thus, once a node N is marked true or false, the mark is propagated to reachable nodes from N that are marked dependant on N, which means already traversed using `DFS1`. This needs to record a dependency set for each node N. In fact, those dependant nodes (i.e., the elements of the dependency set) are the parent nodes of N that are reachable from N and already traversed. The algorithm proceeds by exploring the*

[7]By storage penalty, we mean the memory cost of constructing and recording the strongly-connected components of the product graph to be checked, which is needed by some automata-based model checking algorithms. As the strongly-connected components should be stored prior to any exploration by those algorithms, the memory (or space) cost is high.

label of the states, which are partitioned into negative, conjunctive and disjunctive states. Negative states are those labeled by $\neg$; conjunctive states are those labeled by $\wedge$ and $[\Theta]$; and disjunctive states are those labeled by $\vee$ and $\langle\Theta\rangle$. The following recursive procedure illustrates the marking algorithm.

1. *Start at the initial state.*
2. *At a leave (q, s, i), mark the node true if s satisfies $h(q)$; otherwise, mark the node false.*
3. *At a negative node, evaluate the state by recursively applying the procedure to the non recursive child, and mark the node true if the child is marked false; otherwise mark the node false.*
4. *At a conjunctive node N, proceed as follows:*
   - (a) *Start by non-recursive children and evaluate the node N by applying the procedure recursively to those children. Label N false if one of the children is labeled false and propagate the mark (i.e., mark the dependant nodes on N (if any) true or false depending on the mark of the node N).*
   - (b) *If all the children are evaluated true and there is no recursive child of the node N, then mark the node true and propagate the mark.*
   - (c) *Otherwise, if the unique recursive child has not been already traversed, then apply the procedure recursively to this unique child and mark the node N true if the child is marked true; otherwise, mark the node false and propagate the mark.*
   - (d) *If the recursive child has been already traversed but not market yet, then mark the node N as dependant on the recursive child.*
   - (e) *If the node N is not marked true or false and if it is accepting, then mark N true and propagate the mark if it is reachable from itself using states not marked false. Mark N false and propagate the mark if not.*
   - (f) *If none of the previous cases apply, mark the node false.*
5. *At a disjunctive node N, proceed as follows:*
   - (a) *Start by non-recursive children and evaluate the node N by applying the procedure recursively to those children. Label N true if one of the children is labeled true and propagate the mark.*
   - (b) *If all the children are evaluated false and there is no recursive children of the node N, then mark the node false and propagate the mark.*
   - (c) *Otherwise, search for a recursive child that has not been traversed yet, and if found, then apply the procedure recursively to this child and mark the node N true if the child is marked true and propagate the mark.*

(d) Otherwise, if all the recursive children are already traversed, then mark the node N as dependant on its recursive children.

(e) If the node N is not marked true or false and if it is accepting, then mark N true and propagate the mark if it is reachable from itself using states not marked false. Mark N false and propagate the mark if not.

(f) If none of the previous cases apply, mark the node false.

Let us now consider the complexity of this algorithm[8]*. For each state (q, s, i) in the product graph $\mathcal{B}_{\mathcal{M}_G,\psi}$, if the children are already marked recursively, then marking the state becomes a problem of evaluating a Boolean expression since the children represent the subformulae of the formula in the state. As we consider Boolean expressions over the set Ver, the length of each expression is linear in the size $|\mathcal{B}_{\mathcal{M}_G,\psi}|$ of the arABTA product. As the problem of evaluating Boolean expressions is in LOGSPACE [95], marking a state assuming all the children states are marked can be done deterministically in space $\mathcal{O}(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. Before analyzing the different cases, let us consider the propagation procedure. In fact, the property of arABTA used in this algorithm is that this propagation can be done deterministically in space $\mathcal{O}(\log^2|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. The procedure consists in recording the dependency set, which means determining if the parent nodes of a given node N are reachable from N and already marked traversed. The reachability from N is a graph accessibility problem, and it is known by Jones [82] that the problem is in NLOGSPACE, so it can be done nondeterministically in space $\mathcal{O}(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$, or, by Savitch's theorem [114], deterministically in space $\mathcal{O}(\log^2|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. A necessary condition for a node to be already traversed is to be a recursive node of a given node. Thus, the size of already traversed nodes is bounded by the size of the recursive children. On the one hand, as in arABTA a node labeled by $\wedge$ has only one recursive child, and a node labeled by $[\Theta]$ has no recursive children, the size of recursive children of a conjunctive node is logarithmic in the size of the product graph $\mathcal{B}_{\mathcal{M}_G,\psi}$. On the other hand, as for a disjunctive node only one recursive child should be recorded at time, the size of recursive children needed at a given moment is also logarithmic in the size of the product graph. Thus, marking a node as already traversed can be done deterministically in space $\mathcal{O}(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$, so the whole procedure can be done in space $\mathcal{O}(\log^2|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. Let us now analyze the different cases. If the state is a leave, marking the state is simply evaluating a positive or negative literal, which can be done deterministically in space $\mathcal{O}(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. If the state is a negative node, assuming the non-recursive child is evaluated, marking the node is simply complementing the evaluation of the child, so it can be done deterministically in space $\mathcal{O}(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. Let us then consider the case of a conjunctive state (the case of a disjunctive state is symmetric). If all the*

[8]The complexity analysis of the algorithm is novel in this chapter and has not been addressed in [19].

children are non-recursive, then evaluating the node assuming that the children are already evaluated recursively can be done, as explained above, deterministically in space $\mathcal{O}(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. If the node has a recursive child, which is already evaluated, then evaluating the node is simply evaluating a Boolean expression deterministically in space $\mathcal{O}(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. Otherwise (i.e., the child is already traversed), the mark is propagated, and this can be done, as explained above, deterministically in space $\mathcal{O}(\log^2|\mathcal{B}_{\mathcal{M}_G,\psi}|)$. If the node is accepting, then marking it becomes a problem of reachability from itself using states not already marked false. Assuming the nodes are already marked, this can be done nondeterministically in space $\mathcal{O}(\log|\mathcal{B}_{\mathcal{M}_G,\psi}|)$ [82], or, by Savitch [114], deterministically in space $\mathcal{O}(\log^2|\mathcal{B}_{\mathcal{M}_G,\psi}|)$.

In practice, we do not keep the Boolean values of the children, but whenever we need such a value, we evaluate it recursively. As argued in [86], the depth of the recursion is bounded by the depth of the automata, which is, from Theorem 6, $\mathcal{O}(|\psi|)$. Thus, marking the initial state can be done deterministically in space $\mathcal{O}(|\psi|(\log^2|\mathcal{B}_{\mathcal{M}_G,\psi}|))$. From Proposition 6, we know that: $|\mathcal{B}_{\mathcal{M}_G,\psi}| = \mathcal{O}(|\mathcal{M}_G|\cdot|\mathcal{B}_\psi|)$ and $|\mathcal{B}_\psi| = \mathcal{O}(2^{|\psi|})$. Thus, the model checking problem of GCTL* can be solved in space $\mathcal{O}(|\psi|(\log^2(|\mathcal{M}_G|\cdot 2^{|\psi|})))$, which means $\mathcal{O}(|\psi|(|\psi|+\log|\mathcal{M}_G|)^2)$. ■

Let us now discuss the explicit structure complexity of GCTL* model checking as the complexity of this problem in terms of the size of the input explicit model $\mathcal{M}_G$, that is assuming the formula fixed. In what follows, the logspace and polynomial reductions are denoted respectively by $\leq_{\log}$ and $\leq_p$.

Proposition 7 *Let Mod(L) be a model of the language L, where L $\in$ {CTL, CTLC, CTL*, GCTL*}.*

1. $Mod(CTL^*) \leq_{\log} Mod(GCTL^*)$
2. $Mod(CTL) \leq_{\log} Mod(CTLC)$
3. $Mod(CTLC) \leq_{\log} Mod(GCTL^*)$

Proof 10

1. *CTL* is a subset of GCTL* and thus any model of CTL* is also a model of GCTL*. So, we can easily imagine a deterministic Turing machine TM that can compute this reduction in space $\mathcal{O}(\log n)$ where n is the size of the input model of CTL*. In fact, TM simply looks at the input and writes in its output tape, one by one, the states (including the initial ones), labeling function, and transitions.*
2. *CTL is a subset of CTLC, so the result follows using a similar proof as 1.*

3. *Here we show that the model reduction presented in Section 5.3.2 in Chapter 5 can be computed by a deterministic Turing machine TM in space $\mathcal{O}(\log n)$ where n is the size of the input model of CTLC. TM reads in the input tape a model of CTLC and generates in the output tape, one by one, the same states including the initial ones and the same state labeling function as the input. Furthermore, TM writes α^o in the set of actions Ac if there are transitions defined in R_t, the transition relation in the model of CTLC, and reads the accessibility relations $\sim_{i\to j}$ in the input model one by one and for each one, it writes α^{ij} and β^{ij} in Ac. Then, for each element in Ac, TM writes in the output tape, l_{Ac} one by one as explained in Section 5.3.2 in Chapter 5. Finally, TM looks at each transition (s, s') in the input model and writes, one by one, the transitions (s, α^o, s'). In the same way, TM writes, one by one, the transitions (s, α^{ij}, s') and (s', β^{ij}, s) for each accessibility relation $s \sim_{i\to j} s'$ in the input model.* ■

Theorem 8 *The explicit structure complexity of GCTL* model checking is NLOGSPACE-complete.*

Proof 11

`Membership:` *By fixing the formula ψ to be checked, we obtain an arABTA of a fixed depth. Using the algorithm presented in the proof of Theorem 7, checking the nonemptiness of this automata can be done deterministically in space $\mathcal{O}(\log^2 |\mathcal{M}_G|)$, that is, the problem is in NLOGSPACE.*

`Hardness:` *The hardness in NLOGSPACE follows directly from Proposition 7 (i.e., $Mod(CTL^*) \leq_{\log} Mod(GCTL^*)$) as it is proven in [86] that the explicit structure complexity (called program complexity) of CTL* model checking is NLOGSPACE-complete.* ■

Theorem 9 *The complexity of GCTL* model checking for concurrent programs is PSPACE-complete.*

Proof 12

`Membership:` *As shown in Theorem 7, the model checking problem of GCTL* can be solved in space* ***polynomial*** *in the length of the formula $|\psi|$, but only* ***poly-logarithmic*** *in the size of the explicit model $|\mathcal{M}_G|$. Since the size of a concurrent program is independent from the length of the formula, the problem can still be solved in space polynomial in the length $|\psi|$. On the other hand, since the explicit model is obtained as the product of the components of a concurrent program and this product is at most exponentially larger than the program, the state space is exponential in the length of the program. Thus, membership in PSPACE follows from the fact that the model checking algorithm presented in the proof of Theorem 7 is on-the-fly, that is, we do not have to store all of the product automaton at once and can store, at each step, only the current configuration (a similar argument is used in [142] and [86]).*

`Hardness:` *The hardness in PSPACE is direct from the fact that CTL* $\leq_p$ GCTL* and model checking CTL* is PSPACE-complete for concurrent programs [86].* ■

It is possible to prove hardness in PSPACE using a reduction from polynomial space Turing machines, for example as done in [86]. However, for the sake of simplicity, we used a direct reduction from the model checking of CTL*. We could also use a direct reduction from the nonemptiness problem of concurrent programs proven in [85] to be PSPACE-complete.

C) Space complexity of CTLC

As we did for the complexity of GCTL*, in this section, two results will be presented: (1) the explicit structure complexity of CTLC model checking (i.e., by fixing the formula) is NLOGSPACE-complete; and (2) model checking CTLC for concurrent programs with respect to the size of the components P_i and the length of the formula being checked is PSPACE-complete.

Theorem 10 *The explicit structure complexity of CTLC model checking is NLOGSPACE-complete.*

Proof 13

`Hardness:` *The hardness in NLOGSPACE follows directly from Proposition 7 (i.e., $Mod(CTL)$ $\leq_{\log}$ $Mod(CTLC)$) and the explicit structure complexity (called program complexity) of CTL model checking is NLOGSPACE-complete [86].*

`Membership:` *From Section 5.3.2 in Chapter 5, Theorem 3, and Proposition 7, we proved, using explicit structures, that: (1) $Mod(CTLC) \leq_{\log} Mod(GCTL^*)$; and (2) the reduction is sound. Thus, the membership in NLOGSPACE follows from Theorem 8.* ■

Theorem 11 *The complexity of CTLC model checking for concurrent programs is PSPACE-complete.*

Proof 14

`Hardness:` *The PSPACE lower bound is direct from the fact that CTL $\leq_p$ CTLC and the complexity of model checking CTL is PSPACE-complete for concurrent programs [86].*

`Membership:` *In Section 5.3.2 in Chapter 5, we have presented a polynomial-time transformation of a model $\mathcal{M}_C$ for CTLC to a model $\mathcal{M}_G$ for GCTL* and a formula φ_{CTLC} to a formula φ_{GCTL^*} so that $\mathcal{M}_C \models \varphi_{CTLC}$ iff $\mathcal{M}_G \models \varphi_{GCTL^*}$. Thus, since the model checking problem of GCTL* can be solved in space polynomial in the length of the formula $|\varphi_{GCTL^*}|$, and poly-logarithmic in the size of the explicit model $|\mathcal{M}_G|$ (Theorem 7), we obtain an upper bound space complexity for model checking CTLC with regard to the length of the formula and the size of the explicit model $\mathcal{M}_C$. On* *the other hand, from Theorem 10, the space complexity of model checking CTLC is poly-logarithmic in the size of the explicit model $|\mathcal{M}_C|$. And since the model checking problem of CTL can also be solved in space polynomial in the length of the formula [86], we obtain the space complexity of model checking CTLC, that is polynomial in the length of the formula $|\varphi_{CTLC}|$, and poly-logarithmic in the size of the explicit model $|\mathcal{M}_C|$. Thus, using a similar proof as the one presented in Theorem 9 and observing that the same on-the-fly algorithm presented in this proof can be used for CTLC thanks to the transformation, the result follows.* ■

6.4 Implementation

One of the main objectives of this chapter is to implement the symbolic model checking algorithm of social commitments and their fulfillment. Moreover, we aim to examine various social properties of MASs when agents commit towards each other and to determine which of these properties are retained and which are compromised when agents violate their commitments.

With our results in Chapter 5, we found that symbolic approaches are better than automata-based approaches. As our aim is to introduce a fully automated methodology that builds upon an existing tool, we selected the MCMAS model checker [92] because it supports the semantics of interpreted systems using its own dedicated programming language, called ISPL (Interpreted Systems Programming Language). This language is ground, modular [79] and provides concurrent programs. In logical term, MCMAS supports CTL, CTLK (epistemic logic) and ATL (alternative temporal logic). It is also based on OBDDs that are implemented by means of the efficient CUDD library. Thus, we fully implemented the model checking technique presented in Section 6.2 on top of MCMAS and we extended the implementation of interpreted systems to include shared and unshared variables and the social accessibility relation. The resulting tool is called MCMASC (an extended version of MCMAS with social commitments). Other comparisons between MCMAS and other developed model checkers for multi-agent systems can be found in [111]. One example for which we have been able to carry out the above objectives is the Insurance Claim Processing (an industrial case study), already applied to show how commitments can specify protocols in business settings [137].

6.4.1 Case Study: Insurance Claim Processing

This industrial case study outlines the manner motor damage claims of policy holders are handled by AGFIL, a private insurance company in Ireland (see Figure 21). To deal with a motor damage claim, the AGFIL company is required to provide claim reception and car repair to its policy holders. It also needs to assess (or adjust) claims to protect itself against fraud. AGFIL depends on its associates, Europ Assist, Lee Consulting Service (Lee CS), and various repairers when executing these tasks. Europ Assist through the call center offers a 24-hour emergency call answering service to policy holders for reporting claims and providing them with the name of an approved repairer facility. Lee CS coordinates with AGFIL and deals with repairers and assessors to handle the claims. A network of approved repairers that provide the repair services. AGFIL holds ultimate control in deciding if a given claim is valid and if payment will be made to the repairer.

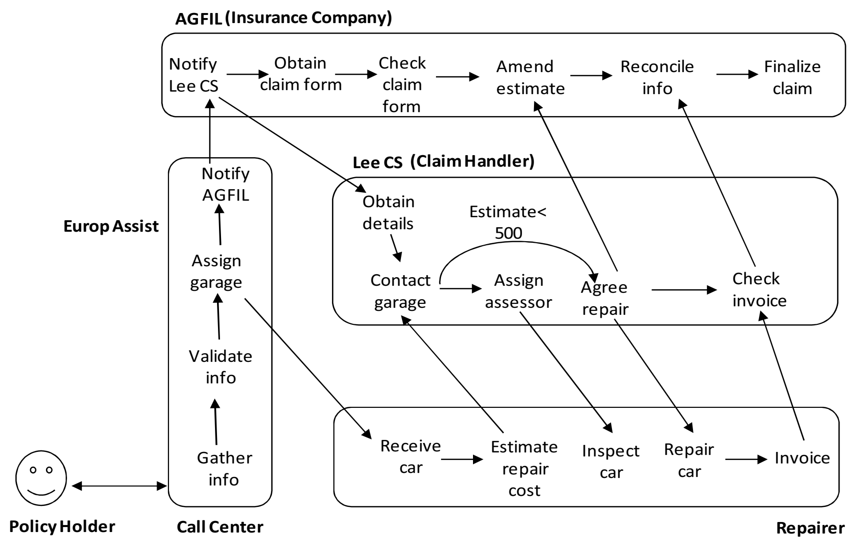


Figure 21: Insurance claim processing [137]

A typical business scenario is as follows. The policy holder phones the call center to notify the company of a new claim. The call center will gather the information using an accident form, validate the information with its database, then assign the nearest garage and give some suggestions about approved repairers to the policy holder. The accident form is then sent to AGFIL and Lee CS, which replicates the AGFIL database to receive notification details. The AGFIL claim handler will check to confirm the cover. In the case of the claim being invalid, Lee CS will be contacted and the process will be stopped. Lee CS will agree upon repair figures if an assessor is not required or will otherwise either appoint an assessor or conduct the assessment itself. When the repairs are completed, the repairer will issue an invoice to the assessor who will check the invoice against the original estimate. Lee CS sends a monthly bordereaux to AGFIL listing all the repairs for that month. More details about the case study can be found in [137].

6.4.2 Specifications

To verify the correctness of the above scenario at design time, various properties can be formalized using CTLC with respect to the model $\mathcal{M}_C$.

Reachability property. A particular situation can be reached from the initial state via some computation sequences. The following formula means that there exists a path where the call center will not commit to the policy holder for gathering claim information until it receives the accident report from the policy holder.

$$E(\neg \mathit{reportAccident}\ U\ (\mathit{reportAccident} \wedge C_{\mathit{Call\_Center} \to \mathit{Policy\_Holder}}\ \mathit{gatherInfo}))$$

Safety property. This property means "something bad never happens". For example, a bad situation is: the policy holder's claim is validated by the call center, but the repairer never commits to repair the car:

$$A\Box\ \neg(\mathit{validClaim} \wedge \neg C_{\mathit{Repairer} \to \mathit{Policy\_Holder}}\ \mathit{carRepair})$$

Liveness property. This property states "something good will eventually happen". For example, in all paths globally if the policy holder reports an accident and his claim is valid, then in all future computations, the call center will commit to assign the garage:

$$A\Box(\mathit{reportAccident} \wedge \mathit{validClaim} \supset A\Diamond\ (C_{\mathit{CallCenter} \to \mathit{Policy\_Holder}}\ \mathit{assignGarage}))$$

Moreover, the following formulae are some examples of commitments (1.a, 2.a, 3.a, and 4.a), their fulfillment (1.b, 2.b, 3.b, and 4.b), and weak and strong violations (1.c, 2.c, 3.c, and 4.c) and (1.d, 2.d, 3.d, and 4.d):

1. Commitment 1
   (a) $E\Diamond(C_{\mathit{Policy\_Holder} \to AGFIL}\ \mathit{insurancePayment})$
   (b) $E\Diamond(Fu(C_{\mathit{Policy\_Holder} \to AGFIL}\ \mathit{insurancePayment}))$
   (c) $\neg A\Box(C_{\mathit{Policy\_Holder} \to AGFIL}\ \mathit{insurancePayment} \supset$
   $A\Diamond(Fu(C_{\mathit{Policy\_Holder} \to AGFIL}\ \mathit{insurancePayment})))$
   (d) $\neg A\Box(C_{\mathit{Policy\_Holder} \to AGFIL}\ \mathit{insurancePayment} \supset$
   $E\Diamond(Fu(C_{\mathit{Policy\_Holder} \to AGFIL}\ \mathit{insurancePayment})))$
2. Commitment 2
   (a) $E\Diamond(C_{AGFIL \to \mathit{Call\_Center}}\ \mathit{receptionPayment})$
   (b) $E\Diamond(Fu(C_{AGFIL \to \mathit{Call\_Center}}\ \mathit{receptionPayment}))$

(c) $\neg A\Box(C_{AGFIL \to Call\_Center}\ receptionPayment \supset$
$A\Diamond(Fu(C_{AGFIL \to Call\_Center}\ receptionPayment)))$

(d) $\neg A\Box(C_{AGFIL \to Call\_Center}\ receptionPayment \supset$
$E\Diamond(Fu(C_{AGFIL \to Call\_Center}\ receptionPayment)))$

3. Commitment 3

   (a) $E\Diamond(C_{Call\_Center \to Policy\_Holder}\ assignGarage)$

   (b) $E\Diamond(Fu(C_{Call\_Center \to Policy\_Holder}\ assignGarage))$

   (c) $\neg A\Box(C_{Call\_Center \to Policy\_Holder}\ assignGarage \supset$
   $A\Diamond(Fu(C_{Call\_Center \to Policy\_Holder}\ assignGarage)))$

   (d) $\neg A\Box(C_{Call\_Center \to Policy\_Holder}\ assignGarage \supset$
   $E\Diamond(Fu(C_{Call\_Center \to Policy\_Holder}\ assignGarage)))$

4. Commitment 4

   (a) $E\Diamond(C_{Repairer \to PolicyHolder}\ carRepair)$

   (b) $E\Diamond(Fu(C_{Repairer \to Policy\_Holder}\ carRepair))$

   (c) $\neg A\Box(C_{Repairer \to Policy\_Holder}\ carRepair \supset A\Diamond(Fu(C_{Repairer \to Policy\_Holder}\ carRepair)))$

   (d) $\neg A\Box(C_{Repairer \to Policy\_Holder}\ carRepair \supset E\Diamond(Fu(C_{Repairer \to Policy\_Holder}\ carRepair)))$

The first formula (1.a) expresses the existence of a computation so that in its future the policy holder commits towards the insurance company AGFIL to pay the insurance and the second formula (1.b) states that such a commitment will eventually be fulfilled, while the third (1.c) and fourth (1.d) formulae are checking the weak and strong violations of this commitment (cf. Section 5.2.2 in Chapter 5). The formula (1.c) says that there is a computation so that in its future the commitment explained in (1.a) is established but from the moment where the commitment is active there is a possible computation where globally the fulfillment never happens, that is the insurance never get paid. The formula (1.d) expresses that after having the commitment from the policy holder towards the insurance company, the fulfillment does not occur in all states of every possible computation.

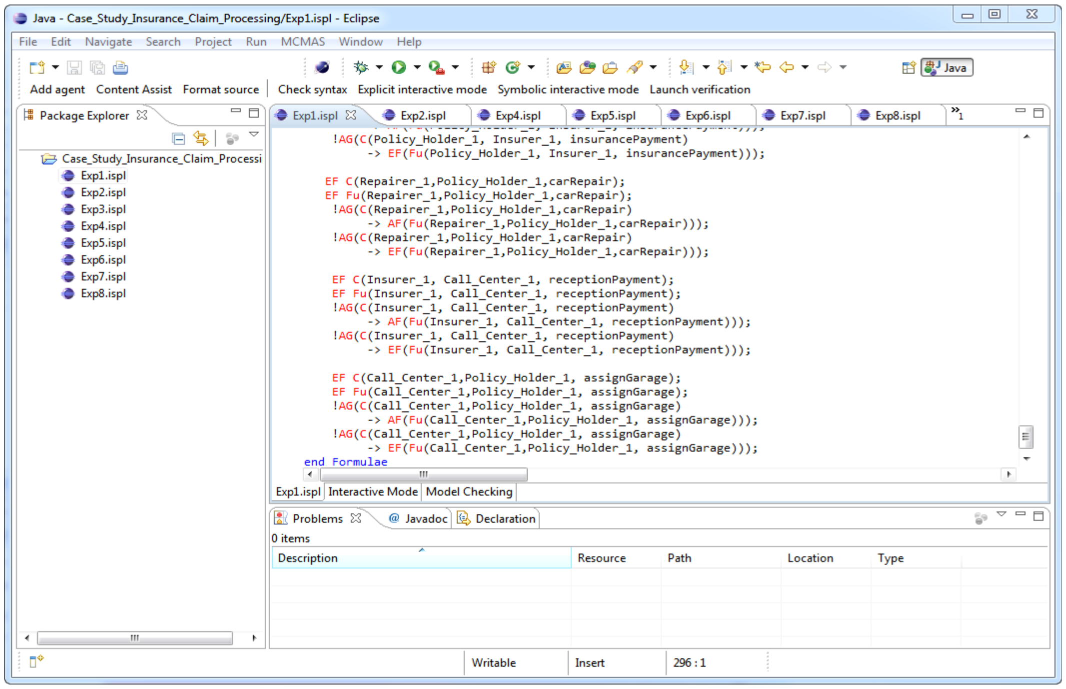
The other formulae in Commitments 2, 3, and 4 can be explained in a similar way. Commitment 2 is from the insurance company AGFIL that commits towards the call center to receive the payment (as compensation for the call service it offers). Commitment 3 is from the call center that commits towards the policy holder that a garage will be assigned. In Commitment 4, the repairer commits to the policy holder to repair his car.

6.4.3 Experimental Results

According to the above scenario, we have six agents: Insurer (or AGFIL), Policy_Holder, Call_Center, Repairer, Assessor and Inspector. As we did in our previous two case studies (the NetBill protocol and the Contract Net protocol) in Chapter 5, we use the model $\mathcal{M}_C$ to formalize the present case study and then compile it using a Büchi automaton. In this formalism, each agent has the set of local states and actions, the set of shared and unshared variables, local protocol function, local evolution function, the valuation function and finally the set of initial states. For instance, the Policy_Holder agent has 29 local states, 3 Boolean variables (communication channels), 21 local actions and 4 initial states. The Policy_Holder agent can be specified using the ISPL language as follows:

```
Agent Policy_Holder
  Vars:
    ph: {ph0,ph1,...,ph29};--local states
    x:  {x0,x1};--x'values can be either zero or one
    z:  {z0,z1};
    f:  {f0,f1};
  end Vars
  Actions={request_EmergencyService,accept_Offer,...,null}; --local actions
  Protocol:--local protocol
     ph=ph0:{request_EmergencyService,...};
     ph=ph2:{accept_Offer,...};
     ...
     Other : {null};
  end Protocol
  Evolution:--local evolution function
    ph=ph1 and x=x0 and Action=request_EmergencyService;
    ...
    ph=ph3 and x=x1 and Action=accept_Offer;
    ...
  end Evolution
end Agent
```

Notice that "−−" refers to the comment in the ISPL language. Our experiments were performed on a laptop equipped with the Intel(R) Core(TM) i5-2430M clocked at 2.4GHz processor and 6GB



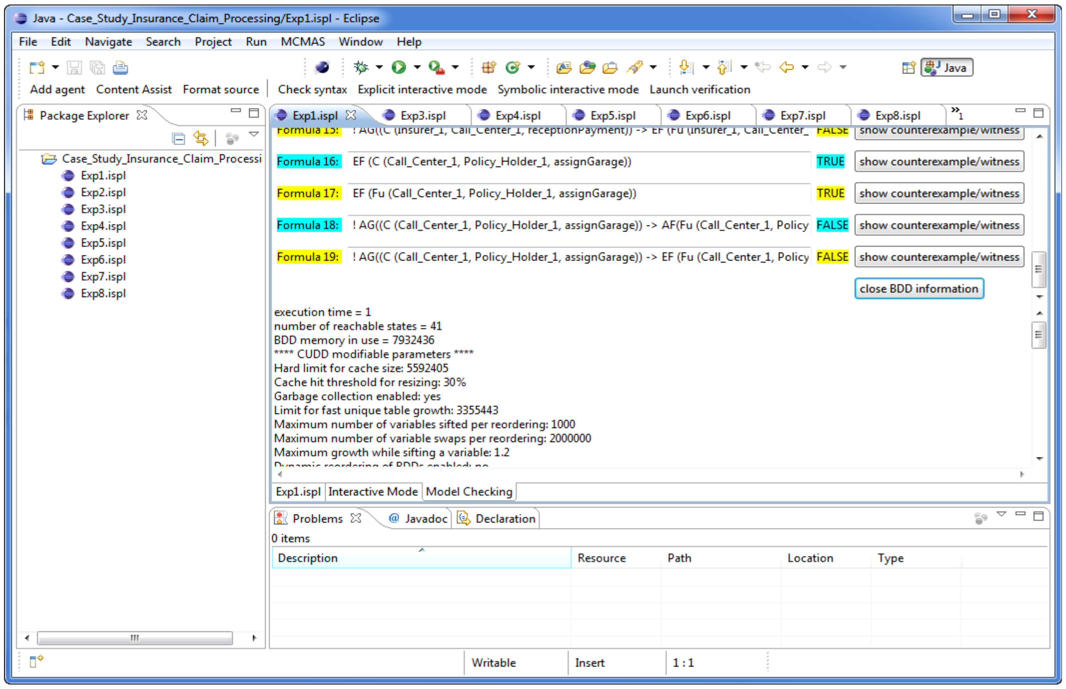
```
   !AG(C(Policy_Holder_1, Insurer_1, insurancePayment)
        -> EF(Fu(Policy_Holder_1, Insurer_1, insurancePayment)));

  EF C(Repairer_1,Policy_Holder_1,carRepair);
  EF Fu(Repairer_1,Policy_Holder_1,carRepair);
   !AG(C(Repairer_1,Policy_Holder_1,carRepair)
        -> AF(Fu(Repairer_1,Policy_Holder_1,carRepair)));
   !AG(C(Repairer_1,Policy_Holder_1,carRepair)
        -> EF(Fu(Repairer_1,Policy_Holder_1,carRepair)));

   EF C(Insurer_1, Call_Center_1, receptionPayment);
   EF Fu(Insurer_1, Call_Center_1, receptionPayment);
   !AG(C(Insurer_1, Call_Center_1, receptionPayment)
        -> AF(Fu(Insurer_1, Call_Center_1, receptionPayment)));
   !AG(C(Insurer_1, Call_Center_1, receptionPayment)
        -> EF(Fu(Insurer_1, Call_Center_1, receptionPayment)));

   EF C(Call_Center_1,Policy_Holder_1, assignGarage);
   EF Fu(Call_Center_1,Policy_Holder_1, assignGarage);
   !AG(C(Call_Center_1,Policy_Holder_1, assignGarage)
        -> AF(Fu(Call_Center_1,Policy_Holder_1, assignGarage)));
   !AG(C(Call_Center_1,Policy_Holder_1, assignGarage)
        -> EF(Fu(Call_Center_1,Policy_Holder_1, assignGarage)));
end Formulae
```

Figure 22: Screen shot of the formulae section in MCMASC.



```
Formula 16:  EF (C (Call_Center_1, Policy_Holder_1, assignGarage))                                      TRUE
Formula 17:  EF (Fu (Call_Center_1, Policy_Holder_1, assignGarage))                                     TRUE
Formula 18:  ! AG((C (Call_Center_1, Policy_Holder_1, assignGarage)) -> AF(Fu (Call_Center_1, Policy    FALSE
Formula 19:  ! AG((C (Call_Center_1, Policy_Holder_1, assignGarage)) -> EF (Fu (Call_Center_1, Policy   FALSE

execution time = 1
number of reachable states = 41
BDD memory in use = 7932436
**** CUDD modifiable parameters ****
Hard limit for cache size: 5592405
Cache hit threshold for resizing: 30%
Garbage collection enabled: yes
Limit for fast unique table growth: 3355443
Maximum number of variables sifted per reordering: 1000
Maximum number of variable swaps per reordering: 2000000
Maximum growth while sifting a variable: 1.2
```

Figure 23: Screen shot of MCMASC output indicating verification results of the first experiment.

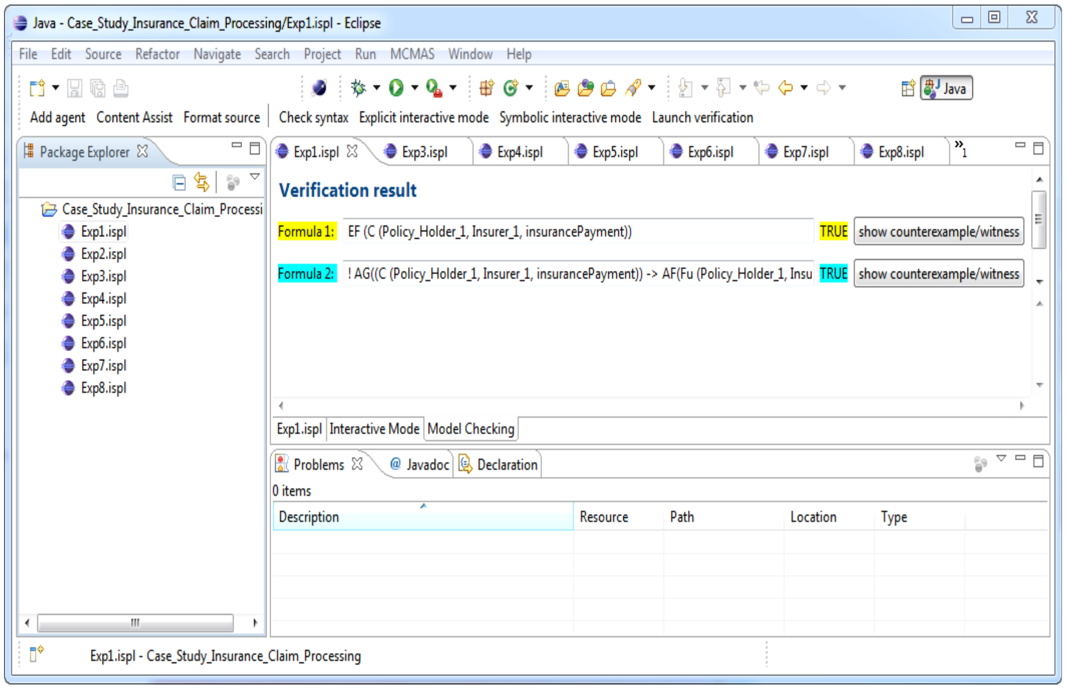
memory running under x86_64 Windows 7. We found that 8 tested formulae (commitments and their fulfillment) hold, 8 tested formulae return false (weak and strong violation) and 3 tested formulae hold (reachability, safety and liveness). Figure 22 depicts the representation of our testing formulae in the MCMASC's formulae section. To check the effectiveness of the proposed symbolic algorithm, we reported 8 experiments in Table 17 where the number of agents (#Agent), the number of reachable states (#States), the execution time (Time) in seconds (sec), and the memory in use (Memory) in MB are given. These experiments are also suitable for testing the scalability of MCMASC as the ISPL code corresponding to any number of interacting agents is generated automatically by making use of a C++ code taking the required number of agents as the only input parameters. The screen shot displayed in Figure 23 reports our verification results with the new tool (MCMASC) regarding to the first experiment. From Table 17, we found that the number of reachable states (which

Table 17: Verification results of the Insurance Claim Processing using MCMASC

#Agents	**#States**	**Time (sec)**	**Memory(MB)**
6	41	1	7.56
12	1669	11	22
18	67529	26	45
24	2.71806e+06	49	47
30	1.08909e+08	278	47
36	4.34657e+09	458	44
42	1.72867e+11	5219	139
48	6.85373e+12	14477	132

reflects state space) increases exponentially when the number of agents increases as we expected. However, the memory usage increases only polynomially, which confirms the theoretical results (i.e., the PSPACE-completeness). Regarding the execution time, the increase is not exponential, but faster than the polynomial, which also confirms the time complexity in concurrent programs, which is APTIME as APTIME = PSPACE (Johnson1990), where APTIME is the class of languages accepted by polynomial-time alternating Turing machines.

Because our model is verified successfully, we consider an imaginary condition under which the commitment violation can occur to further demonstrate our approach. For example, when the policy holder fails to send the agreed amount of insurance payment to the insurer agent, its commitment is violated (see Figure 24). In the Figure 24, the results generated by MCMASC mean that there is no way to reach the insurance payment state from the policy holder's commitment state; so the commitment is strongly violated. In this case, MCMASC generates a counterexample showing an execution indicating the reason for this problem.


Java - Case_Study_Insurance_Claim_Processing/Exp1.ispl - Eclipse
File Edit Source Refactor Navigate Search Project Run MCMAS Window Help
Add agent Content Assist Format source Check syntax Explicit interactive mode Symbolic interactive mode Launch verification
Package Explorer
Case_Study_Insurance_Claim_Processi
Exp1.ispl
Exp2.ispl
Exp3.ispl
Exp4.ispl
Exp5.ispl
Exp6.ispl
Exp7.ispl
Exp8.ispl
Exp1.ispl Interactive Mode Model Checking
Problems Javadoc Declaration
Description Resource Path Location Type
Exp1.ispl - Case_Study_Insurance_Claim_Processing

Verification result

Formula 1: EF (C (Policy_Holder_1, Insurer_1, insurancePayment)) TRUE show counterexample/witness

Formula 2: ! AG((C (Policy_Holder_1, Insurer_1, insurancePayment)) -> AF(Fu (Policy_Holder_1, Insu TRUE show counterexample/witness

0 items

Figure 24: MCMASC output showing violation: failure to send insurance payment.

6.5 Discussion

The present chapter is an improvement and continuation of our previous work published in [52] in which we developed a new symbolic algorithm for model checking an extended logic of CTL with modalities for commitments and their fulfillment and violation. The algorithm has been fully implemented on top of the MCMAS model checker to directly verify commitments and their fulfillment and violation. We studied the time and space complexity of model checking CTLC, which has not been addressed yet and proved that it is respectively P-complete and PSPACE-complete. These results match the time and space complexity of model checking CTL in explicit models and concurrent programs. In our implementation, we used a widely studied industrial case study and conducted 8 experiments with large state space approximately ($6.85373e + 12$ states), which demonstrated the effectiveness and applicability of our approach in terms of execution time and memory in use. Recall that it is the first proposal that develops a model checking algorithm for commitments.

The proof presented in this chapter differs from the proof of PSPACE-complete for model checking CTLK (a combination of CTL logic with the logic of knowledge [105]) in concurrent programs introduced in [111] in that it does not use the results from Turning machine theory. In our perspective, the relationship between automata-theoretic technique and Turning machine technique is not sufficiently well understood despite being widely acknowledged. With these results, our proof provides an alternative proof of the upper bound for the space complexity of model checking CTL in concurrent programs, which can be easily extended to CTLC.

In fact, a dedicated logic and its model checking for commitments play the same role as CTLK and MCMAS do for knowledge. It is worth noticing that CTLC is semantically different from CTLK because a commitment modality cannot be expressed with any CTLK connector under the assumption that the two interacting agents are completely different.

Chapter 7

Conclusion

The main contribution of this thesis lies in proposing an integrated approach for addressing two open problems: (1) defining a computationally grounded semantics for agent communication in terms of social commitments; and (2) model checking commitments and their actions together with commitment-based protocols.

To summarize, based on the technical background introduced in Chapter 2, we started by exploring the state of the art on how computational logics can be advocated to define a formal semantics for agent communication by making use of social commitments and their actions. In particular, we evaluated some prominent and predominate proposals against six crucial criteria to highlight their strengths and limitations. We also evaluated current specification languages and different verification techniques that have been introduced within those proposals to specify and verify commitment-based protocols. Afterward, we proceeded to address the identified challenges and limitations in Chapter 4 by developing a new branching-time temporal logic, called ACTL^{*c}, which extends full computation tree logic (CTL*) with the addition of modal operators for representing and reasoning about social commitments and associated actions. The proposed logical language is expressive and succinct as it is an extension of CTL*. We proposed the notion of accessible and non-accessible paths to interpret the semantics of commitment and action modal operators. Then we used an agency function to check for the existence of two interacting agents at all states along the accessible path. Although the defined semantics satisfies all our proposed criteria and exclude spurious aspects that plague existing proposals regarding the over-specification problem and the intuition and computation problem, it is semi-computationally grounded in the sense of the computational grounding theory of agency introduced in [151]. Also, we used our logical language ACTL^{*c} to derive a new specification language of commitment-based protocols, which waives the current logical languages of actions. To verify commitments and their actions and commitment-based protocols, we developed a new symbolic model checking algorithm (cf. Appendix A) and defined a formal and automatic reduction method to transform the problem of model checking ACTL^{*c} to the problem of model checking GCTL* in order to make use of the CWB-NC model checker. The proposed reduction method is sound and preserves the real and concrete meaning of commitments as the commitment modal formula is transformed into GCTL* formula. To experimentally evaluate the effectiveness of this reduction method, we adopted two business protocols (the NetBill protocol and Contract Net protocol) widely used in the literature. We concluded with our approach is feasible, usable and applicable to check a large real-life business models, which allows us to achieve the first, second, and third objectives of our thesis (cf. Chapter 1).

To balance between expressiveness and verification efficiency aspects, we adopted a refined fragment of ACTL^{*c}, called CTLC, in Chapter 5. In order to define a full-computationally grounded semantics of commitment modal formula, we extended the formalism of interpreted systems with shared and unshared variables and considered the local states of the interacting agents in the definition of the social accessibility relation. We introduced the link between commitments and their fulfillment, which is missing in the literature. We also presented axioms of commitments and their fulfillment, which have not been introduced yet. Our commitment logic is of version at least as strong as $KD45n$. In our verification technique, the problem of model checking CTLC is reduced into the problems of model checking ARCTL and GCTL* to respectively use the extended NuSMV symbolic model checker and the CWB-NC automata-based model checker as (a benchmark). We also prove that our reduction methods are sound. With those theoretical foundations, we proceeded to illustrate the effectiveness and efficiency of the proposed reduction methods using two case studies (the NetBill protocol and Contract Net protocol) taken from business domain. We implemented our reduction tools on top of the extended NuSMV and CWB-NC model checkers. The overall conclusion coincides with the usual considerations in that automatic verification using symbolic techniques is better than automata-based techniques with respect to the execution time and the number of interacting agents. The results in this chapter coincide with the third objective of the thesis.

We also developed a new symbolic model checking algorithm to directly and automatically verify commitments and their fulfillment and commitment-based protocols. We analyzed the time and space complexity of the proposed model checking algorithm, which has not been addressed yet. We proved that the time complexity of model checking CTLC matches the time complexity of model checking CTL in explicit models, which is P-complete with regard to the size of the model and length of the formula. Also, the space complexity of model checking CTLC copes with the space complexity of model checking CTL in concurrent programs, which is PSPACE-complete with respect to the size of the components of these programs. We fully implemented our algorithm on top of the MCMAS model checker. Finally, we checked the efficiency and scalability of the proposed algorithm using an industrial case study. This chapter allows us to achieve the fourth objective of the thesis.

7.1 Future Work

This thesis has painted the following picture: model checking commitments and their actions and commitment-based protocols is feasible either by using indirect method (reduction method) or direct method (via developing symbolic algorithm). Nevertheless, some issues still need to be addressed. In particular, open issues not considered in thesis include:

– Some authors introduced a different definition for commitments where there are a debtor, a creditor, an antecedent condition, and a consequent condition (for example [28, 29, 127, 159, 160]). In this definition, a commitment is said to be active when the antecedent condition is true. In Chapter 5, we presented a weak solution to formally define conditional commitments by making use of material implication. Yolum and Singh [158] used the strict implication operator to logically link between the condition and consequent of commitment. Singh [127] pointed out that the conditionality of commitments matches a strong conditional rather than strict and material conditionals in logical terms. However, Singh's semantics of conditional commitment does not use the accessibility relation and possible-worlds semantics, a classic way of defining semantics of modal operators. Using a similar idea is very hard to be model checked. In Chapter 5, we suggested another solution by defining a new accessibility relation for conditional commitments (CC), which can be used to give a computationally grounded semantics to CC. However, this solution needs further investigation to develop a new BDD-based algorithm along with analyzing its time and space complexity.

– Some commitment actions such assign and delegate are not considered. This needs to extend the proposed logic, CTLC, to consider such actions and then develop their BDD-based algorithms to extend our model checker.

We also intend to

– Prove theoretically the soundness and completeness of an axiomatic system for CTLC in order to complement our practical work on model checking. The soundness problem means that if a CTLC formula φ is provable (i.e., $\vdash \varphi$), then it is satisfiable (i.e., $\models \varphi$). The completeness problem is the reverse direction of the soundness problem: $\models \varphi$ implies $\vdash \varphi$.

– Extend our logic with dialogue action formulae to reconcile conflicts and reason about the validity of commitments as in real-life business scenarios and then verify protocols for agent negotiation specified using this logic. We contributed in this direction by introducing a new specification language of agent negotiation protocols using commitments and dialogue actions without considering formal semantics and BDD-algorithms of those actions [47].

– Apply our model checker to automatically verify business interactions among agent-based Web services that are organized within societies called communities under the assumption that they have the same functionalities. In fact, we recently developed a new engineering methodology based on concepts of Tropos engineering methodology [20] for establishing and managing communities of Web services with the alliance structure wherein business interactions among Web services are modeled using commitments augmented with dialogue actions [45].

– Use our commitment-based protocols to regulate the behavior of composing agent-based Web services. This will help overcome the limitation of the standard specification languages introduced into practice, such as BPML (Business Process Modeling Language), which concentrate only on low-level details of the interactions using message exchanges among services in an unnecessarily restrictive temporal order, while failing to capture the business intent of the interactions. In this setting, commitments in the form of business contracts are used to represent and reason about business interactions between Web services from high-level abstractions. Then we can directly use our model checker to verify the correctness of protocols, given desirable properties expressed in CTLC to specify requirements needed for composite Web services.

Another direction of future work is to use Bounded Model Checking (BMC) [106] for CTLC because although symbolic model checking with OBDDs was the first big breakthrough on the state explosion problem and is still widely used, OBDDs have a number of problems: (1) finding the ordering of variables that results in a small OBDD is difficult; and (2) for some Boolean formulae no space-efficient ordering is possible [21]. The main idea in BMC is to search for a counterexample in executions whose length is bounded by some integer k. If it is not found, then one increases k until either a counterexample is found or some pre-known upper bound is reached. The BMC problem is reduced to a propositional satisfiability problem to be solved by SAT methods rather than BDDs.

Bibliography

[1] Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable Agent Interaction in Abductive Logic Programming: The SCIFF Framework. ACM Transactions on Computational Logic 9(4), 1–43 (2008)

[2] Artikis, A., Sergot, M., Pitt, J.: Specifying Norm-Governed Computational Societies. ACM Transactions on Computational Logic 10(1), 1–42 (2009)

[3] Austin, J.: How to Do Things with Words. Oxford University Press: Oxford, England (1962)

[4] Baier, C., Katoen, J.: Principles of Model Checking. The MIT Press (2008)

[5] Baldoni, M., Baroglio, C., Marengo, E.: Behavior Oriented Commitment-Based Protocols. In: Coelho, H., Studer, R., Wooldridge, M. (eds.) ECAI. vol. 215, pp. 137–142. IOS Press (2010)

[6] Baldoni, M., Baroglio, C., Marengo, E.: Commitment-Based Protocols with Behavioral Rules and Correctness Properties of MAS. In: Omicini, A., Sardiña, S., Vasconcelos, W. (eds.) DALT. LNCS, vol. 6619, pp. 60–77. Springer (2011)

[7] Baldoni, M., Baroglio, C., Marengo, E., Patti, V.: Constitutive and Regulative Specifications of Commitment Protocols: A Decoupled Approach. ACM Transactions on Intelligent Systems and Technology 2(3), 1–26 (2011), (In Press)

[8] Bentahar, J., El-Menshawy, M., Qu, H., Dssoulia, R.: Communicative Commitments: Model Checking and Complexity Analysis. Knowledge-Based Systems xx, 1–44 (2012), (In Press)

[9] Bentahar, J., Labban, J.: An Argumentation-Driven Model for Flexible and Efficient Persuasive Negotiation. Group Decision and Negotiation 20(4), 411–435 (2009)

[10] Bentahar, J., Maamar, Z., Wan, W., Benslimane, D., Thiran, P., Subramanian, S.: Agent-Based Communities of Web Services: An Argumentation-driven Approach. Service Oriented Computing and Applications 2(4), 219–238 (2008)

[11] Bentahar, J., Meyer, J.J., Wan, W.: Model Checking Agent Communication. In: Dastani, M., Hindriks, K., Meyer, J.J. (eds.) Specification and Verification of Multi-Agent Systems, chap. 3, pp. 67–102. Springer, first edn. (2010)

[12] Bentahar, J., Meyer, J.J., Wana, W.: Model Checking Communicative Agent-Based Systems. Knowledge-Based Systems 22, 142–159 (2009)

[13] Bentahar, J., Moulin, B., Chaib-draa, B.: Towards A Formal Framework for Conversational Agents. In: Proc. of Int. Workshop on ACLCP (2003)

[14] Bentahar, J., Moulin, B., Chaib-draa, B.: Commitment and Argument Network: A New Formalism for Agent Communication. In: Dignum, F. (ed.) ACL. LNCS, vol. 2922, pp. 146–165. Springer (2004)

[15] Bentahar, J., Moulin, B., Chaib-draa, B.: Specifying and Implementing A Persuasion Dialogue Game using Commitments and Arguments. In: Rahwan, I., Moraitis, P., Reed, C. (eds.) ArgMAS. LNCS, vol. 3366, pp. 130–148. Springer (2005)

[16] Bentahar, J., Moulin, B., Meyer, J.J., Chaib-draa, B.: A Logical Model for Commitment and Argument Network for Agent Communication. In: Proc. of the 3rd Int. Conf. on AAMAS. pp. 792–799. IEEE Computer Society (2004)

[17] Bentahar, J., Moulin, B., Meyer, J.J., Lespérance, Y.: A New Logical Semantics for Agent Communication. In: Inoue, K., Satoh, K., Toni, F. (eds.) CLIMA. LNCS, vol. 4371, pp. 151–170. Springer (2007)

[18] Bhat, G.: Tableau-Based Approaches to Model-Checking. Ph.D. thesis, North Carolina State University (1998)

[19] Bhat, G., Cleaveland, R., Groce, A.: Efficient Model Checking via Büchi Tableau Automata. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV. LNCS, vol. 2102, pp. 38–52. Springer (2001)

[20] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (2004)

[21] Bryant, R.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers 35(8), 677–691 (1986)

[22] Castelfranchi, C.: Commitments: From Individual Intentions to Groups and Organizations. In: Lesser, V., Gasser, L. (eds.) ICMAS. pp. 41–48. The MIT Press (1995)

[23] Chaib-draa, B.: Industrial Applications of Distributed AI. Communications of the ACM 38(11), 49–53 (1995)

[24] Chesani, F., Mello, P., Montali, M., Torroni, P.: Commitment Tracking via the Reactive Event Calculus. In: Boutilier, C. (ed.) IJCAI. pp. 91–96 (2009)

[25] Chopra, A., Singh, M.: Nonmonotonic Commitment Machines. In: Dignum, F. (ed.) ACL. LNCS, vol. 2922, pp. 183–200. Springer (2004)

[26] Chopra, A., Singh, M.: Contextualizing Commitment Protocols. In: Nakashima, H., Wellman, M., Weiss, G., Stone, P. (eds.) AAMAS. pp. 1345–1352. ACM (2006)

[27] Chopra, A., Singh, M.: Constitutive Interoperability. In: Padgham, L., Parkes, D., Müller, J., Parsons, S. (eds.) AAMAS. vol. 2, pp. 797–804. IFAAMAS (2008)

[28] Chopra, A., Singh, M.: Multiagent Commitment Alignment. In: Proc. of the 8th Int. Joint Conf. on AAMAS. pp. 937–944. ACM Press (2009)

[29] Chorpa, A., Artikis, A., Bentahar, J., Colombetti, M., Dignum, F., Fornara, N., Jones, A., Singh, M., Yolum, P.: Research Directions in Agent Communication. ACM Transactions on Intelligent Systems and Technology 2(3), 1–26 (2011), (In Press)

[30] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV: An Open Source Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV. LNCS, vol. 2404, pp. 359–364. Springer (2002)

[31] Clarke, E., Emerson, E.: Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In: Kozen, D. (ed.) Logics of Programs. LNCS, vol. 131, pp. 52–71 (1982)

[32] Clarke, E., Emerson, E., Sistla, A.: Automatic Verification of Finite-State Concurrent Systems using Temporal Logic Specifications. In: Proc. of the 10th ACM SIGACT-SIGPLAN Symposium on PPL. pp. 117–126. POPL'83, ACM (1986)

[33] Clarke, E., Grumberg, O., Peled, D.: Model Checking. The MIT Press, Cambridge, Massachusetts (1999)

[34] Cohen, P., Levesque, H.: Intention is Choice with Commitment. Artificial Intelligence 42(2–3), 213–261 (1990)

[35] Colombetti, M.: A Commitment-Based Approach to Agent Speech Acts and Conversations. In: Proc. of Int. Workeshop on ALCP, 4th Int. Conf. on Autonomous Agents (Agents 2000). pp. 21–29 (2000)

[36] Colombetti, M., Fornara, N., Verdicchio, M.: A Social Approach to Communication in Multiagent Systems. In: Leite, J.A., Omicini, A., Sterling, L., Torroni, P. (eds.) DALT. LNCS, vol. 2990, pp. 191–220. Springer (2004)

[37] Cost, R., Chen, Y., Finin, T., Labrou, Y., Peng, Y.: Using Colored Petri Nets for Conversation Modeling. In: Dignum, F., Greaves, M. (eds.) Issues in Agent Communication. LNCS, vol. 1916, pp. 178–192. Springer (2000)

[38] Courcoubetis, C., Vardi, M., Wolper, P., Yannakakis, M.: Memory-Efficient Algorithms for the Verification of Temporal Properties. Formal Methods in System Design 1(2–3), 275–288 (1992)

[39] Desai, N., Cheng, Z., Chopra, A., Singh, M.: Toward Verification of Commitment Protocols and their Compositions. In: Durfee, E.H., Yokoo, M., Huhns, M.N., Shehory, O. (eds.) AAMAS. pp. 144–146. IFAAMAS (2007)

[40] Desai, N., Mallya, A., Chopra, A., Singh, M.: Interaction Protocols as Design Abstractions for Business Processes. IEEE Transactions on Software Eng. 31(12), 1015–1027 (2005)

[41] Desai, N., Singh, M.: A Modular Action Description Language for Protocol Composition. In: Proc. of the 22nd AAAI Conf. on Artificial Intelligence. pp. 962–967 (2007)

[42] Desai, N., Singh, M.: On the Enactability of Business Protocols. In: Fox, D., Gomes, C.P. (eds.) AAAI. pp. 1126–1131. AAAI Press (2008)

[43] Dignum, F., Greaves, M. (eds.): Issues in Agent Communication, LNCS, vol. 1916. Springer (2000)

[44] Dong, J., Peng, T., Zhao, Y.: Automated Verification of Security Pattern Compositions. Information and Software Technology 52(3), 274–295 (2010)

[45] El-Menshawy, M., Bentahar, J., Dssouli, R.: Enhancing Engineering Methodology for Communities of Web Services. In: Baldoni, M., et al. (eds.) MALLOW. vol. 494, pp. 33–42. CEUR-WS.org (2009)

[46] El-Menshawy, M., Bentahar, J., Dssouli, R.: An Integrated Semantics of Social Commitments and Associated Operations. In: Baldoni, M., et al. (eds.) MALLOW. vol. 494, pp. 222–230. CEUR-WS.org (2009)

[47] El-Menshawy, M., Bentahar, J., Dssouli, R.: Modeling and Verifying Business Interactions via Commitments and Dialogue Actions. In: Jedrzejowicz, P., Nguyen, N.T., Howlett, R.J., Jain, L.C. (eds.) KES-AMSTA. LNCS, vol. 6071, pp. 11–21. Springer (2010)

[48] El-Menshawy, M., Bentahar, J., Dssouli, R.: Verifiable Semantic Model for Agent Interactions using Social Commitments. In: Dastani, M., Fallah-Seghrouchni, A.E., Leite, J., Torroni, P. (eds.) LADS. LNCS, vol. 6039, pp. 128–152 (2010)

[49] El-Menshawy, M., Bentahar, J., Dssouli, R.: Model Checking Commitment Protocols. In: Mehrotra, K.G., et al. (eds.) IEA–AIE. LNCS, vol. 6704, pp. 37–47. Springer (2011)

[50] El-Menshawy, M., Bentahar, J., Dssouli, R.: Symbolic Model Checking Commitment Protocols using Reduction. In: Omicini, A., Sardina, S., Vasconcelos, W. (eds.) DALT. LNAI, vol. 6619, pp. 185–203. Springer (2011)

[51] El-Menshawy, M., Bentahar, J., Kholy, W.E., Dssouli, R.: Reducing Model Checking Commitments for Agent Communication to Model Checking ARCTL and GCTL*. Autonomous Agent Multi-Agent Systems xx, 1–47 (2012), (In Press)

[52] El-Menshawy, M., Bentahar, J., Qu, H., Dssouli, R.: On the Verification of Social Commitments and Time. In: Sonenberg, L., Stone, P., Tumer, K., Yolum, P. (eds.) AAMAS. pp. 483–490. IFAAMAS (2011)

[53] El-Menshawy, M., Bentahar, J., Wan, W., Dssouli, R.: Verifying Conformance of Commitment Protocols via Symbolic Model Checking. In: Int. Workshop on Agent Communication, 9th Int. Joint Conf. on AAMS. pp. 53–72 (2010)

[54] El-Menshawy, M., Wan, W., Bentahar, J., Dssouli, R.: Symbolic Model Checking for Agent Interactions: Extended Abstract. In: Proc. of the 9th Int. Joint Conf. on AAMAS. pp. 1555–1556. IFAAMAS (2010)

[55] El-Menshawy, M., Bentahara, J., Kholya, W.E., Dssouli, R.: Verifying Conformance of Multi-Agent Commitment-based Protocol. Expert Systems with Applications xx, 1–44 (2012), (In Press)

[56] Emerson, E.: Temporal and Modal Logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, chap. 16, pp. 995–1072. Elsevier (1990)

[57] Emerson, E., Halpern, J.: Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. Computer and System Sciences 30(1), 1–24 (1985)

[58] Emerson, E., Halpern, J.: Sometimes and NotNever, Revisited: On Branching versus Linear Time Temporal Logic. Journal of ACM 33(1), 151–178 (1986)

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

[60] Finin, T., Fritzson, R., McKay, D., McEntire, R.: KQML as An Agent Communication Language. In: Proc. of the 3rd Int. Conf. on CIKM. pp. 456–463. ACM (1994)

[61] Fisher, M.: An Introduction to Practical Formal Methods Using Temporal Logic. Wiley (2011)

[62] Fornara, N., Colombetti, M.: Operational Specification of a Commitment-Based Agent Communication Language. In: Proc. of the 1st Int. Joint Conf. on AAMS. pp. 535–542 (2002)

[63] Fornara, N., Colombetti, M.: Defining Interaction Protocols using a Commitment-Based Agent Communication Language. In: Proc. of the 2nd Int. Joint Conf. on AAMAS. pp. 520–527 (2003)

[64] Fornara, N., Viganò, F., Verdicchio, M., Colombetti, M.: Artificial Institutions: A Model of Institutional Reality for Open Multi-Agent Systems. AI and Law 16(1), 89–105 (2008)

[65] Gerard, S., Singh, M.: Formalizing and Verifying Protocol Refinements. ACM Transactions on Intelligent Systems and Technology 2(3), 1–35 (2011), (In Press)

[66] Giordano, L., Martelli, A., Schwind, C.: Specifying and Verifying Systems of Communicating Agents in a Temporal Action Logic. In: Cappelli, A., Turini, F. (eds.) AI*IA. LNCS, vol. 2829, pp. 262–274. Springer (2003)

[67] Giordano, L., Martelli, A., Schwind, C.: Specifying and Verifying Interaction Protocols in a Temporal Action Logic. Journal of Applied Logic 5(2), 214–234 (2007)

[68] Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic Causal Theories. Artificial Intelligence 153(1–2), 49–104 (2004)

[69] Gray, J.: Notes on Database Operating Systems. In: Flynn, M.J., Gray, J., Jones, A.K., Lagally, K., Opderbeck, H., Popek, G.J., Randell, B., Saltzer, J.H., Wiehle, H. (eds.) Advanced Course: Operating Systems. LNCS, vol. 60, pp. 393–481 (1978)

[70] Greaves, M., Holmback, H., Bradshaw, J.: What Is a Conversation Policy? In: Dignum, F., Greaves, M. (eds.) Issues in Agent Communication. LNCS, vol. 1916, pp. 118–131. Springer (2000)

[71] Guerin, F., Pitt, J.: Guaranteeing Properties for E-Commerce Systems. In: Padget, J.A., Shehory, O., Parkes, D.C., Sadeh, N.M., Walsh, W.E. (eds.) AMEC IV. LNCS, vol. 2531, pp. 253–272. Springer (2002)

[72] Habermas, J.: The Theory of Communicative Action. The Polity Press (1984)

[73] Hamblin, C.: Fallacies. Methuen, London (1970)

[74] Holzmann, G.: The Model Checker Spin. Software Engineering 23(5), 279–295 (1997)

[75] Holzmann, G.: SPIN Model Checker: Primer and Reference Manual. Addison Wesley Professional (2003)

[76] Holzmann, G., Godefroid, P., Pirottin, D.: Coverage Preserving Reduction Strategies for Reachability Analysis. In: Linn, R., Üyar, M. (eds.) PSTV. pp. 349–363. IFIP Transactions, North-Holland (1992)

[77] Hughes, G., Cresswell, M.: A New Introduction to Modal Logic. Routledge (1996)

[78] Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about System. Cambridge University Press, 2nd edn. (2004)

[79] Jamroga, W., Ågotnes, T.: Modular Interpreted Systems. In: Durfee, E.H., Yokoo, M., Huhns, M.N., Shehory, O. (eds.) AAMAS. pp. 131:1–131:8. FAAMAS (2007)

[80] Jennings, N.: Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems. Knowledge Engineering Review 8(3), 223–250 (1993)

[81] Jones, A., Parent, X.: A Convention-Based Approach to Agent Communication Languages. Group Decision and Negotiation 16(2), 101–141 (2007)

[82] Jones, N.: Space-Bounded Reducibility among Combinatorial Problems. Computer and System Sciences 11(1), 68–85 (1975)

[83] Kone, M., Shimazu, A., Nakajima, T.: The State of the Art in Agent Communication Languages. Knowledge and Information Systems 2(3), 259–284 (2000)

[84] Kova, M., Bentahar, J., Maamar, Z., Yahyaoui, H.: A Formal Verification Approach of Conversations in Composite Web Services using NuSMV. In: Fujita, H., Marík, V. (eds.) SoMeT. vol. 199, pp. 245–261. IOS Press (2009)

[85] Kozen, D.: Lower Bounds for Natural Proof Systems. In: Proc. of the 18th IEEE Symposium on Foundation of Computer Science. pp. 254–266 (1977)

[86] Kupferman, O., Vardi, M., Wolper, P.: An Automata-Theoretic Approach to Branching-Time Model Checking. ACM 47(2), 312–360 (2000)

[87] Labrou, Y., Finin, T.: Semantics and Conversations for an Agent Communication Language. In: Singh, M.P., Rao, A.S., Wooldridge, M. (eds.) ATAL. LNCS, vol. 1365, pp. 209–214. Springer (1998)

[88] Lamport, L.: Proving the Correctness of Multiprocess Programs. IEEE Transactions on Software Engineering 3(2), 125–143 (1977)

[89] Lichtenstein, O., Pnueli, A.: Checking that Finite State Concurrent Programs Satisfy their Linear Specification. In: Proc. of the 12th ACM SIGACT-SIGPLAN symposium on PPL. pp. 77–107. ACM (1985)

[90] Lomuscio, A., Pecheur, C., Raimondi, F.: Automatic Verification of Knowledge and Time with NuSMV. In: Proc. of the 20th Int. Joint Conference on AI. pp. 1384–1389 (2007)

[91] Lomuscio, A., Penczek, W., Qu, H.: Partial Order Reductions for Model Checking Temporal-epistemic Logics over Interleaved Multi-agent Systems. Fundamenta Informaticae 101(1-2), 71–90 (2010)

[92] Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: A Model Checker for the Verification of Multi-Agent Systems. In: Bouajjani, A., Maler, O. (eds.) CAV. LNCS, vol. 5643, pp. 682–688. Springer (2009)

[93] Lomuscio, A., Qu, H., Solanki, M.: Towards Verifying Contract Regulated Service Composition. Autonomous Agents and Multi-Agent Systems 24(3), 345–373 (2012)

[94] Lomuscio, A., Sergot, M.: Deontic Interpreted Systems. Studia Logica 75(1), 63–92 (2003)

[95] Lynch, N.: Log Space Recognition and Translation of Parenthesis Languages. Journal of ACM 24(4), 583–590 (1977)

[96] Mallya, A., Huhns, M.: Commitments Among Agents. IEEE Internet Computing 7(4), 90–93 (2003)

[97] Mallya, A., Singh, M.: Incorporating Commitment Protocol into Tropos. In: Müller, J.P., Zambonelli, F. (eds.) AOSE. LNCS, vol. 3950, pp. 69–80. Springer (2006)

[98] Mallya, A., Singh, M.: An Algebra for Commitment Protocols. Autonomous Agents and Multi-Agent Systems 14(2), 143–163 (2007)

[99] Mallya, A., Yolum, P., Singh, M.: Resolving Commitments among Autonomous Agents. In: Dignum, F. (ed.) ACL. LNCS, vol. 2922, pp. 166–182. Springer (2004)

[100] Manna, Z., Pnueli, A.: The Temporal Logic of Concurrent and Reactive Systems: Specification. Springer–Verlag, NY, USA (1992)

[101] Marengo, E., Baldoni, M., Baroglio, C., Chopra, A., Patti, V., Singh, M.: Commitments with Regulations: Reasoning about Safety and Control in REGULA. In: Tumer, K., Yolum, P., Sonenberg, L., Stone, P. (eds.) AAMAS. pp. 467–474. IFAAMAS (2011)

[102] McMillan, K.: Symbolic Model Checking: An Approach to the State Explosion Problem. Ph.D. thesis, Carnegie Mellon University (1992)

[103] Meyer, J.J., Veltman, F.: Intelligent Agents and Common Sense Reasoning. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logic, vol. 3, chap. 18, pp. 991–1029. Elsevier (2007)

[104] Pecheur, C., Raimondi, F.: Symbolic Model Checking of Logics with Actions. In: Edelkamp, S., Lomuscio, A. (eds.) Model Checking and Artificial Intelligence. LNCS, vol. 4428, pp. 113–128. Springer (2007)

[105] Penczek, W., Lomuscio, A.: Verifying Epistemic Properties of Multi-Agent Systems via Bounded Model Checking. Fundamenta Informaticae 55(2), 167–185 (2003)

[106] Penczek, W., Wozna, B., Zbrzezny, A.: Bounded Model Checking for the Universal Fragment of CTL. Fundamenta Informaticae 51(1–2), 135–156 (2002)

[107] Pham, D., Harland, J.: Temporal Linear Logics as a Basis for Flexible Agent Interactions. In: Durfee, E., Yokoo, M., Huhns, M., Shehory, O. (eds.) AAMAS. pp. 124–131 (2007)

[108] Pnueli, A.: The Temporal Semantics of Concurrent Programs. Theoretical Computer Science 13, 45–60 (1981)

[109] Pnueli, A.: The Temporal Logic of Programs. In: Proc. of the 18th Annual Symposium on FOCS. pp. 46–57. IEEE Computer Society Press (977)

[110] Queille, J., Sifakis, J.: Specification and Verification of Concurrent Systems in CESAR. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) Symposium on Programming. LNCS, vol. 137, pp. 337–351. Springer (1982)

[111] Raimondi, F.: Model Checking Multiagent Systems. Ph.D. thesis, University College London (2006)

[112] Reichenbach, H.: Elements of Symbolic Logic. MacMillan, New York (1947)

[113] Sacerdoti, E.: The Structure of Plans and Behavior. Elsevier, New York (1977)

[114] Savitch, W.: Relationships between Nondeterministic and Deterministic Tape Complexities. Computer and System Sciences 4(2), 177–192 (1970)

[115] Schnoebelen, P.: The Complexity of Temporal Logic Model Checking. In: Advances in Modal Logic. vol. 4, pp. 1–44 (2002)

[116] Searle, J.: Speech Acts: An Essay in the Philosophy of Lanaguge. Cambridge University Press: Cambridge, England (1969)

[117] Searle, J., Vanderveken, D.: Foundations of Illocutionary Logic. Cambridge, England: Cambridge University (1985)

[118] Shanahan, M.: Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia. The MIT Press (1997), cambridge

[119] Shanahan, M.: An Abductive Event Calculus Planner. Logic Programming 44, 207–239 (2000)

[120] Singh, M.: Social and Psychological Commitments in Multiagent Systems. In: AAAI Fall Symposium on KA at SOL. pp. 104–106 (1991)

[121] Singh, M.: A Conceptual Analysis of Commitments in Multiagent Systems. Tech. rep., North Carolina State University, Raleigh, NC (1996)

[122] Singh, M.: Commitments Among Autonomous Agents in Information-Rich Environments. In: Boman, M., van de Velde, W. (eds.) MAAMAW. LNCS, vol. 1237, pp. 141–155. Springer (1997)

[123] Singh, M.: Agent Communication Languages: Rethinking the Principles. IEEE Computer Society 31(12), 40–47 (1998)

[124] Singh, M.: An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts. AI and Law 7(1), 97–113 (1999)

[125] Singh, M.: A Social Semantics for Agent Communication Languages. In: Dignum, F., Greaves, M. (eds.) Issues in Agent Communication. LNCS, vol. 1916, pp. 31–45. Springer (2000)

[126] Singh, M.: Formalizing Communication Protocols for Multiagent Systems. In: Veloso, M.M. (ed.) IJCAI. pp. 1519–1524 (2007)

[127] Singh, M.: Semantical Considerations on Dialectical and Practical Commitments. In: Fox, D., Gomes, C.P. (eds.) AAAI. pp. 176–181. AAAI Press (2008)

[128] Singh, M., Chopra, A.: Programming Multiagent Systems without Programming Agents. In: Braubach, L., Briot, J.P., Thangarajah, J. (eds.) ProMAS. LNCS, vol. 5919, pp. 1–14. Springer (2010)

[129] Singh, M., Chopra, A., Desai, N.: Commitment-Based Service-Oriented Architecture. IEEE Computer 42(11), 72–79 (2009)

[130] Singh, M., Huhns, M.: Service-Oriented Computing: Semantics, Processes, Agents. Wiley, London (2005)

[131] Sirbu, M.: Credits and Debits on the Internet. IEEE Spectrum 34(2), 23–29 (1997)

[132] Sistla, A., Clarke, E.: The Complexity of Propositional Linear Temporal Logics. ACM 32, 733–749 (1985)

[133] Spoletini, P.: Verification of Temporal Logic Specification via Model Checking. Ph.D. thesis, Politecnico di Milano, Italy (2005)

[134] Spoletini, P., Verdicchio, M.: Commitment Monitoring in a Multi-Agent System. In: Burkhard, H.D., Lindemann, G., Verbrugge, R., Varga, L.Z. (eds.) CEEMAS. LNCS, vol. 4696, pp. 83–92. Springer (2007)

[135] Spoletini, P., Verdicchio, M.: An Automata-Based Monitoring Technique for Commitment-Based Multiagent Systems. In: Hübner, J.F., Matson, E.T., Boissier, O., Dignum, V. (eds.) COIN. LNCS, vol. 5428, pp. 172–187. Springer (2009)

[136] Sycara, K.: Multiagent Systems. AI Magazine 10(2), 79–93 (1998)

[137] Telang, P., Singh, M.: Business Modeling via Commitments. In: Kowalczyk, R., Vo, Q., Maamar, Z., Huhns, M. (eds.) SOCASE. LNCS, vol. 5907, pp. 111–125 (2009)

[138] Telang, P., Singh, M.: Enhancing Tropos with Commitments. In: Borgida, A., Chaudhri, V.K., Giorgini, P., Yu, E.S.K. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. 5600, pp. 417–435. Springer (2009)

[139] Telang, P., Singh, M.: Specifying and Verifying Cross-Organizational Business Models: An Agent-Oriented Approach. IEEE Transactions on Services Computing 4(1), 1–14 (2011), (In Press)

[140] 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. LNCS, vol. 5948, pp. 228–243. Springer (2010)

[141] Vardi, M., Wolper, P.: Automata-Theoretic Techniques for Modal Logics of Programs. Computer and System Sciences 32(2), 183–221 (1986)

[142] Vardi, M., Wolper, P.: Reasoning about Infinite Computations. Information and Computation 115, 1–37 (1994)

[143] Venkatraman, M., Singh, M.: Verifying Compliance with Commitment Protocols: Enabling Open Web-Based Multiagent Systems. Autonomous Agents and Multi-Agent Systems 2(3), 217–236 (1999)

[144] Verdicchio, M., Colombetti, M.: A Logical Model of Social Commitment for Agent Communication. In: Proc. of the 2nd Int. Joint Conf. on AAMAS. pp. 528–535 (2003)

[145] Verdicchio, M., Colombetti, M.: A Logical Model of Social Commitment for Agent Communication. In: Dignum, F. (ed.) ACL. LNCS, vol. 2922, pp. 128–145 (2004)

[146] Verdicchio, M., Colombetti, M.: Dealing with Time in Content Language Expressions. In: van Eijk, R., Huget, M., Dignum, F. (eds.) AC. LNCS, vol. 3396, pp. 91–105. Springer (2005)

[147] Verdicchio, M., Colombetti, M.: From Message Exchanges to Communicative Acts to Commitments. Electronic Notes in Theoretical Computer Science 157, 75–94 (2006)

[148] Walton, D., Krabbe, E.: Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning. State University of New York Press, Albany (1995)

[149] Winikoff, M.: Implementing Commitment-Based Interactions. In: Durfee, E., Yokoo, M., Huhns, M., Shehory, O. (eds.) AAMAS. pp. 873–880 (2007)

[150] Winikoff, M., Liu, W., Harland, J.: Enhancing Commitment Machines. In: Leite, J.A., Omicini, A., Torroni, P., Yolum, P. (eds.) DALT. LNAI, vol. 3476, pp. 198–220. Springer (2005)

[151] Wooldridge, M.: Computationally Grounded Theories of Agency. In: Proc. of the 4th Int. Conf. on Multi-Agent Systems. pp. 13–22. IEEE Computer Society (2000)

[152] Wooldridge, M.: Semantic Issues in the Verification of Agent Communication Languages. Autonomous Agents and Multi-Agent Systems 3(1), 9–31 (2000)

[153] Wooldridge, M.: An Introduction to Multi-Agent Systems. John Wiley and Sons (2009)

[154] Wooldridge, M., Jennings, N.: Intelligent Agents: Theory and Practice. Knowledge Engineering Review 2(10), 115–152 (1995)

[155] Xing, J., Singh, M.: Formalization of Commitment-Based Agent Interaction. In: SAC. pp. 115–120. ACM (2001)

[156] Xing, J., Singh, M.: Engineering Commitment-Based Multiagent Systems: A Temporal Logic Approach. In: Proc. of the 2nd Int. Joint Conf. on AAMAS. pp. 891–898 (2003)

[157] Yolum, P.: Design Time Analysis of Multiagent Protocols. Data and Knowladge Engineering 63(1), 137–154 (2007)

[158] Yolum, P., Singh, M.: Commitment Machines. In: Meyer, J.J.C., Tambe, M. (eds.) ATAL. LNCS, vol. 2333, pp. 235–247. Springer (2002)

[159] Yolum, P., Singh, M.: Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In: Proc. of the Int. Joint Conf. on AAMAS. pp. 527–534 (2002)

[160] Yolum, P., Singh, M.: Reasoning about Commitments in the Event Calculus: An Approach for Sepcifying and Executing Protocols. Annals of Mathematics and Artificial Intelligence 42(1–3), 227–253 (2004)

Appendix A

Symbolic Model Checking Technique for ACTL^{*c}

The aim of this appendix is to complete the third part of our approach by developing a new symbolic model checking technique dedicated to ACTL^{*c}. This technique alleviates the state explosion problem of automata-based techniques. In a nutshell, given the model M representing a commitment-based protocol and a formula φ describing a property, the problem of model checking can be defined as establishing whether or not M satisfies φ. Clarke et al. in their seminal book shown that the problem of model checking CTL* can be reduced to the problems of model checking CTL and LTL (page 48, [33]). The present appendix follows a similar approach by effectively reducing the problem of model checking ACTL^{*c} into the problems of model checking ALTL^c and ACTL^c. ALTL^c and ACTL^c are LTL and CTL augmented with modalities for commitments and associated actions. The motivation of this reduction is to use the standard CTL and LTL procedures introduced in [78, 33].

A.1 Symbolic Model Checking Algorithm for ACTL^{*c}

The proposed symbolic model checking algorithm $SMC(\varphi, M)$ takes the model M and an ACTL^{*c} formula φ and returns the set of states satisfying the formula φ (i.e., $[\![\varphi]\!]$). We divided our algorithm into main algorithm and sub-algorithms that can be called in the main algorithm. In the main algorithm, we compute the set $[\![\varphi]\!]$ using the following operations on sets: *difference, union, intersection, existential universal quantification.* When sets of states are encoded using Boolean functions, all these operations on sets are translated into operations on Boolean functions that can be easily represented in OBDDs [21]. For example, the *intersection* of two sets means the *conjunction* of the Boolean functions encoding the two sets. The computation of $[\![\varphi]\!]$ is also based on the use of paths, $\approx$ and $\mathbb{R}_c$, which are encoded as Boolean functions.

The basic idea of our main algorithm is inspired by the algorithm introduced in [33] to verify CTL* formulae using the combination of LTL and CTL model checking algorithms. We extend this algorithm by adding the procedures that deal with the new operators of our logic (see Algorithm 6). As in Clarke et al.'s algorithm [33], we assume that each ACTL^{*c} formula φ can be divided into "maximal state sub-formulae" such that each maximal state sub-formula ψ: (1) includes the existential path quantifier "E"; (2) differs from φ; and (3) is not contained in any other state sub-formula of φ [33]. Each maximal state sub-formula is an ACTL^{*c} formula, which can be divided into

Algorithm 6 $SMC(\varphi, M)$: the set $[\![\varphi]\!]$ satisfying the ACTL^{*c} formula φ

```
1: $\varphi$ is an atomic formula: return $\mathbb{V}(\varphi)$
2: $\varphi$ is $\neg\varphi_1$: return $\mathbb{S}\backslash SMC(\varphi_1, M)$
3: $\varphi$ is $\varphi_1 \vee \varphi_2$: return $SMC(\varphi_1, M) \cup SMC(\varphi_2, M)$
4: $\varphi$ is $C(Ag_1, Ag_2, \varphi_1)$: return $SMC_c(Ag_1, Ag_2, \varphi_1, M)$
5: $\varphi$ is $E\varphi_1$: return $SMC_{E\varphi}(\varphi, M)$
```

other maximal state sub-formulae. At level 0, each maximal state sub-formula of φ is identified and at level 1, each of these sub-formulae is divided into other maximal state sub-formulae, and so on until no new maximal state sub-formula can be identified. The algorithm also works in stages such that in stage i all maximal state sub-formulae of φ of level smaller than i are processed (i.e., all states of the model M satisfying these sub-formulae are labeled with them). More precisely, the algorithm starts by checking atomic formula (line 1) and logical operators: negation and disjunction (lines 2 and 3). It then checks the commitment modality (line 4) by calling the procedure $SMC_c(Ag_1, Ag_2, \varphi_1, M)$ (see Algorithm 7). Line 5 deals with an ACTL^{*c} formula that contains the path quantifier E (i.e., $\varphi = E\varphi_1$) (see Algorithm 16).

The procedure $SMC_c(Ag_1, Ag_2, \varphi, M)$ is performed in three steps. In step 1, it computes the set X of states satisfying the formula $E\varphi$ using the standard procedure $SMC_{exi}(E\varphi, M)$ introduced in [78] where φ is the commitment content. The motivation behind computing this set is to ensure

Algorithm 7 $SMC_c(Ag_1, Ag_2, \varphi, M)$: the set $[\![C(Ag_1, Ag_2, \varphi)]\!]$

```
1: $X \leftarrow SMC_{exi}(E\varphi, M)$
2: $Y \leftarrow \{\pi |\ \pi \in \mathbb{R}_c(s, Ag_1, Ag_2)$ and $s \in X$ and $P\_SAT(\pi, \varphi, M)\}$
3: $Z \leftarrow \{s |\ s = \pi(0)$ s.t. $\pi \in Y$ and $\forall \pi' \in \mathbb{R}_c(s, Ag_1, Ag_2) : \pi' \in Y\}$
4: return $Z$
```

that there is a path along which the content holds. In step 2, the procedure builds the set Y of accessible paths from every state in the set X such that the formula φ holds along these paths. The computation of Y is completed by calling the procedure $P\_SAT(\pi, \varphi, M)$ that returns true if the accessible path π satisfies φ and false otherwise (see Algorithm 8). In step 3, the procedure computes the set Z of states satisfying the commitment. Z contains all states s at which the accessible paths computed in Y start, such that all accessible paths from s are in Y. This last condition allows excluding from Z the states from which an accessible path emerges, which does not satisfy φ.

$P\_SAT(\pi, \varphi, M)$ depends on the structure of φ so that when φ is an atomic formula, it returns true if the first state of the path π is in the set of states satisfying φ (i.e., $\pi(0) \in SMC(\varphi, M)$). Otherwise, $P\_SAT(\pi, \varphi, M)$ operates recursively on the structure of φ and calls one of the procedures described in Algorithms 9 to 15. Each procedure takes a path π and the corresponding parameters and returns either true (i.e., the formula holds along the path) or false. For instance, the procedure

Algorithm 8 $P\_SAT(\pi, \varphi, M)$: Boolean

```
1: $\varphi$ is an atomic formula: return ($\pi(0) \in SMC(\varphi, M)$)
2: $\varphi$ is $\neg\varphi_1$: return not($P\_SAT(\pi, \varphi_1, M)$)
3: $\varphi$ is $\varphi_1 \vee \varphi_2$: return $P\_SAT(\pi, \varphi_1, M)$ or $P\_SAT(\pi, \varphi_2, M)$
4: $\varphi$ is $\bigcirc\varphi_1$: return $P\_SAT(\pi \uparrow \pi(1), \varphi_1, M)$
5: $\varphi$ is $\varphi_1 \; U \; \varphi_2$: return $P\_SAT_U(\pi, \varphi_1, \varphi_2, M)$
6: $\varphi$ is $Wi(Ag_1, Ag_2, \mathcal{C})$: return $P\_SAT_{wi}(\pi, \mathcal{C}, M)$
7: $\varphi$ is $Fu(Ag_1, Ag_2, \mathcal{C})$: return $P\_SAT_{fu}(\pi, \mathcal{C}, M)$
8: $\varphi$ is $Vi(Ag_1, Ag_2, \mathcal{C})$: return $P\_SAT_{vi}(\pi, \mathcal{C}, M)$
9: $\varphi$ is $Re(Ag_2, Ag_1, \mathcal{C})$: return $P\_SAT_{re}(\pi, Ag_2, Ag_1, \mathcal{C}, M)$
10: $\varphi$ is $De(Ag_1, Ag_3, \mathcal{C})$: return $P\_SAT_{de}(\pi, Ag_1, Ag_3, \mathcal{C}, M)$
11: $\varphi$ is $As(Ag_2, Ag_3, \mathcal{C})$: return $P\_SAT_{as}(\pi, Ag_2, Ag_3, \mathcal{C}, M)$
```

$P\_SAT_U(\pi, \varphi, \psi, M)$ (see Algorithm 9) starts by checking whether or not the intersection of the set of states in this path π and the set $[\![\psi]\!]$ is empty. If this is the case, the procedure returns false as

Algorithm 9 $P\_SAT_U(\pi, \varphi, \psi, M)$: Boolean

```
1: if ($SMC(\psi, M) \cap \{t | \; t \in \pi\} = \emptyset$) return false
2: else   $i \leftarrow 0$
3:   While ($P\_SAT(\pi \uparrow \pi(i), \varphi, M)$ and $P\_SAT(\pi \uparrow \pi(i), \neg\psi, M)$) do
4:   $i \leftarrow i + 1$
5:   end while
6: return $P\_SAT(\pi \uparrow \pi(i), \psi, M)$
7: end if
```

ψ will never hold. Otherwise, the procedure iterates over π's transition steps until either φ is not satisfied or ψ becomes true. The procedure returns true if ψ becomes true, and false otherwise.

The procedure $P\_SAT_{wi}(\pi, C(Ag_1, Ag_2, \varphi), M)$ only returns true when there exists an accessible state s from the first state of the path π which is in the set $[\![C(Ag_1, Ag_2, \varphi)]\!]$ and at the same time $\pi(0)$ does not belong to this set and π is accessible for Ag_1 and Ag_2 (see Algorithm 10). In a similar way, the procedure $P\_SAT_{rel}(\pi, C(Ag_1, Ag_2, \varphi), M)$ is reported in Algorithm 11.

Algorithm 10 $P\_SAT_{wi}(\pi, C(Ag_1, Ag_2, \varphi), M)$: Boolean

```
1: if ($\exists s$ s.t. $\pi(0) \approx_{1,2} s,\ s \in SMC_c(Ag_1, Ag_2, \varphi, M)$) then
2: if $\pi(0) \in \mathbb{S} - SMC_c(Ag_1, Ag_2, \varphi, M)$ and $\pi \in \mathbb{R}_c(\pi(0), Ag_1, Ag_2)$ then
3:   return true
4: else return false
5: end if end if
```

Algorithm 11 $P\_SAT_{re}(\pi, Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi), M)$: Boolean

```
1: if ($\exists s$ s.t. $\pi(0) \approx_{2,1} s,\ s \in SMC_c(Ag_1, Ag_2, \varphi, M)$) then
2: if $\pi(0) \in \mathbb{S} - SMC_c(Ag_1, Ag_2, \varphi, M)$ and $\pi \in \mathbb{R}_c(\pi(0), Ag_1, Ag_2)$ then
3:   return true
4: else return false
5: end if end if
```

The procedure $P\_SAT_{fu}(\pi, C(Ag_1, Ag_2, \varphi), M)$ checks if every accessible state from the first state of the path π and the first state itself are in the set $[\![C(Ag_1, Ag_2, \varphi)]\!]$ and if π is accessible using $\mathbb{R}_c$. In this case, it returns true otherwise, it returns false (see Algorithm 12).

Algorithm 12 $P\_SAT_{fu}(\pi, C(Ag_1, Ag_2, \varphi), M)$: Boolean

```
1: if ($\forall s$ s.t. $\pi(0) \approx_{1,2} s,\ s \in SMC_c(Ag_1, Ag_2, \varphi, M)$) then
2: if $\pi(0) \in SMC_c(Ag_1, Ag_2, \varphi, M)$ and $\pi \in \mathbb{R}_c(\pi(0), Ag_1, Ag_2)$ then
3:   return true
4: else return false
5: end if end if
```

The procedure $P\_SAT_{vi}(\pi, C(Ag_1, Ag_2, \varphi), M)$ is similar to the procedure $P\_SAT_{fu}(\pi, C(Ag_1, Ag_2, \varphi), M)$ except that it does not check the accessibility of π but if π satisfies $\neg\varphi$, in which case it returns true, otherwise false (see Algorithm and 13).

According to the proposed semantics, the procedure $P\_SAT_{de}(\pi, Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi), M)$ (reps. $P\_SAT_{as}(\pi, Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi), M)$) starts by checking if the agent Ag_1 (resp. Ag_2) cancels (resp. releases) his commitment along the path π and its first state is in the set $[\![C(Ag_3, Ag_2, \varphi)]\!]$ (reps. $[\![C(Ag_1, Ag_3, \varphi)]\!]$). In this case, the procedure returns true or false otherwise.

The procedure $SMC_{E\varphi}(\varphi, M)$ is the core of our main algorithm as it is responsible for checking the two main cases of $E\varphi$ (see Algorithm 16). In fact, as for CTL* [33], any ACTL^{*c} formula is either ACTL^c formula, ALTL^c formula, a combination of both using $\wedge$ or $\vee$, or a complex nested formula. The case of combined formulae using $\wedge$ or $\vee$ is handled by Algorithm 6 (lines 2 and 3). If the formula is a complex nested formula, then an ALTL^c formula is obtained by replacing each maximal state sub-formula by a new (or fresh) atomic proposition. Algorithm 16 first checks if φ is an ACTL^c formula in that case it calls the procedure $SMC_{actlc}(\varphi, M)$. This procedure is obtained by

Algorithm 13 $P\_SAT_{vi}(\pi, C(Ag_1, Ag_2, \varphi), M)$: Boolean

```
1: if ($\forall s$ s.t. $\pi(0) \approx_{1,2} s$, $s \in SMC_c(Ag_1, Ag_2, \varphi, M)$) then
2: if $\pi(0) \in SMC_c(Ag_1, Ag_2, \varphi, M)$ then
3:   return $P\_SAT(\pi, \neg\varphi, M)$
4: else return false
5: end if end if
```

Algorithm 14 $P\_SAT_{de}(\pi, Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi), M)$: Boolean

```
1: if ($P\_SAT_{wi}(\pi, C(Ag_1, Ag_2, \varphi), M)$) and ($\pi(0) \in SMC_c(Ag_3, Ag_2, \varphi, M)$) then
2:   return true
3: else return false
4: end if
```

extending the standard procedure introduced in [78] for CTL formula with the following algorithms: SMC_{Ewi} for $Wi(Ag_1, Ag_2, \mathcal{C})$ (see Algorithm 18), SMC_{Efu} for $Fu(Ag_1, Ag_2, \mathcal{C})$ (see Algorithm 20), SMC_{Evi} for $Vi(Ag_1, Ag_2, \mathcal{C})$ (see Algorithm 21), SMC_{Ere} for $Re(Ag_2, Ag_1, \mathcal{C})$ (see Algorithm 19), SMC_{Ede} for $De(Ag_1, Ag_3, \mathcal{C})$ (see Algorithm 22), and SMC_{Eas} for $As(Ag_2, Ag_3, \mathcal{C})$ (see Algorithm 23).

If the formula is not an ACTL^c formula, then each maximal state sub-formula (having the form $E\psi_i$) is replaced by a new atomic proposition (a_i) (line 2), which is added to the set $\mathcal{PV}$ (line 4). Recursively, the set of states satisfying each sub-formula $E\psi_i$ is computed by calling $SMC_{E\phi}(E\psi_i, M)$ (line 5). The procedure is called recursively for the sub-formulae of each level until all the sub-formulae are processed. Finally, the ALTL^c obtained formula φ' is processed and the set of states satisfying it is returned by calling $SMC_{altlc}(\varphi', M)$ (see Algorithm 17). The obtained ALTL^c formula is either an LTL formula, which is handled using the standard LTL model checking [33] (line 1 of Algorithm 17), or a formula containing commitments. In this case, each state commitment formula is replaced by a new atomic proposition (line 3) and the set of states satisfying this formula is computed using $SMC_c(\mathcal{C}_i, M)$ (line 6). The final formula after performing all the replacements in a recursive way is an LTL formula.

The procedure $SMC_{Ewi}(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi), M)$ first computes the set X (resp. Y) of states that does not satisfy (resp. satisfy) the commitment. It then constructs the set Z of those states (i.e., X) that have accessible states in Y and from which an accessible path emerges. In a similar way, the procedure $SMC_{Ere}(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi), M)$ is reported in Algorithm 19.

The procedure $SMC_{Efu}(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi), M)$ computes the set X_1 of states satisfying the commitment. It then constructs and returns the set X_2 of those states in X_1 that can only see states in the same set using $\approx_{1,2}$ (see Algorithm 20). Notice that we do not need to filter X_2 to only keep states from which an accessible path can emerge because any state satisfies the commitment

Algorithm 15 $P\_SAT_{as}(\pi, Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi), M)$: Boolean

```
1: if ($P\_SAT_{re}(\pi, Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi), M)$) and
2: ($\pi(0) \in SMC_c(Ag_1, Ag_3, \varphi, M)$) then
3:   return true
4: else return false
5: end if
```

Algorithm 16 $SMC_{E\varphi}(\varphi, M)$: the set $[\![E\varphi]\!]$

```
1: $\varphi$ is an ACTL$^c$ formula: return $SMC_{actlc}(\varphi, M)$,
2:   $\varphi' \leftarrow \varphi[a_1/E\psi_1, \ldots, a_\mathcal{K}/E\psi_\mathcal{K}]$,
3:     for $i = 1, \ldots, \mathcal{K}$ do
4:       $\mathcal{PV} \leftarrow \mathcal{PV} \cup \{a_i\}$ //introducing a new proposition $a_i$
5:       $\mathbb{V}_s(a_i) \leftarrow SMC_{E\varphi}(E\psi_i, M)$ //adding $a_i$ to each state $s$ that satisfies $E\psi_i$
6:     end for
7:   return $SMC_{altlc}(\varphi', M)$
```

has such path.

In procedure $SMC_{Evi}(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi), M)$ (see Algorithm 21), the computation of the sets X_1 and X_2 is defined as in Algorithm 20. The set X_3 contains all states s at which a path π starts such that along this path the negation of the commitment content holds using the procedure P_SAT.

The procedure $SMC_{Ede}(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi), M)$ (resp. $SMC_{Eas}(Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi), M)$) computes the set X of states satisfying the delegated (resp. assigned) commitment; then proceeds to build the set Y of states satisfying the withdraw (resp. release) action, see Algorithms 22 and 23 respectively.

Theorem 12 (Complexity) *The complexity of ACTL^{*c} model checking problem is PSPACE-complete with respect to symbolic representations.*

Proof 15 (sketch) *The complexity of ACTL^{*c} depends on the complexity of ACTL^c and ALTL^c. The complexity of ACTL^c depends on the complexity of CTL (P-complete [115, 32]) and P_SAT algorithm. This algorithm is similar to "PATH" algorithm presented in [111] and its complexity is PSPACE-complete. Here we should notice that the accessibility relations $\mathbb{R}_c$ and $\approx_{x,y}$ only need a polynomial space as only three states should be recorded in memory. From Algorithm 17, the complexity of ALTL^c depends on the complexity of LTL (PSPACE-complete [132, 33, 115]) and Algorithm 7 whose complexity depends also on the complexity of P_SAT algorithm. Consequently, its complexity is PSPACE-complete.*

Algorithm 17 $SMC_{altlc}(\varphi, M)$: the set $[\![\varphi]\!]$

```
1: if $\varphi$ is an LTL formula then return $SMC_{ltl}(\varphi, M)$
2: else
3:   $\varphi' \leftarrow \varphi[a_1/\mathcal{C}_1, \ldots, a_J/\mathcal{C}_J]$
4:   for $i = 1, \ldots, J$ do
5:     $\mathcal{PV} \leftarrow \mathcal{PV} \cup \{a_i\}$ //introducing a new proposition $a_i$
6:     $\mathbb{V}_s(a_i) \leftarrow SMC_c(\mathcal{C}_i, M)$ //adding $a_i$ to each state $s$ that satisfies $\mathcal{C}_i$
7:   end for
8:   return $SMC_{altlc}(\varphi', M)$ end if
```

Algorithm 18 $SMC_{Ewi}(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi), M)$: the set $[\![EWi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))]\!]$

```
1: $X \leftarrow \mathbb{S} - SMC_c(Ag_1, Ag_2, \varphi, M)$
2: $Y \leftarrow SMC_c(Ag_1, Ag_2, \varphi, M)$
3: $Z \leftarrow \{s \in X | \exists s' \in Y$ s.t. $s \approx_{1,2} s'$ and $\exists \pi \notin \mathbb{R}_c(s, Ag_1, Ag_2)\}$
4: return $Z$
```

Algorithm 19 $SMC_{Ere}(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi), M)$: the set $[\![ERe(Ag_2, Ag_1, C(Ag_1, Ag_2, \varphi))]\!]$

```
1: $X \leftarrow \mathbb{S} - SMC_c(Ag_1, Ag_2, \varphi, M)$
2: $Y \leftarrow SMC_c(Ag_1, Ag_2, \varphi, M)$
3: $Z \leftarrow \{s \in X | \exists s' \in Y$ s.t. $s \approx_{2,1} s'$ and $\exists \pi \notin \mathbb{R}_c(s, Ag_1, Ag_2)\}$
4: return $Z$
```

Algorithm 20 $SMC_{Efu}(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi), M)$: the set $[\![EFu(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))]\!]$

```
1: $X_1 \leftarrow SMC_c(Ag_1, Ag_2, \varphi, M)$
2: $X_2 \leftarrow \{s \in X_1 | \forall s'$ s.t. $s \approx_{1,2} s'$ we have $s' \in X_1\}$
3: return $X_2$
```

Algorithm 21 $SMC_{Evi}(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi), M)$: the set $[\![EVi(Ag_1, Ag_2, C(Ag_1, Ag_2, \varphi))]\!]$

```
1: $X_1 \leftarrow SMC_c(Ag_1, Ag_2, \varphi, M)$
2: $X_2 \leftarrow \{s \in X_1 | \forall s'$ s.t. $s \approx_{1,2} s'$ we have $s' \in X_1\}$
3: $X_3 \leftarrow \{s \in X_2 | \exists \pi$ s.t. $s = \pi(0)$ and $P\_SAT(\pi, \neg\varphi, M)\}$
4: return $X_3$
```

Algorithm 22 $SMC_{Ede}(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi), M)$: the set $[\![EDe(Ag_1, Ag_3, C(Ag_1, Ag_2, \varphi))]\!]$

```
1: $X \leftarrow SMC_c(Ag_3, Ag_2, \varphi, M)$
2: $Y \leftarrow SMC_{Ewi}(C(Ag_1, Ag_2, \varphi), M)$
3: return $X \cap Y$
```

Algorithm 23 $SMC_{Eas}(Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi), M)$: the set $[\![EAs(Ag_2, Ag_3, C(Ag_1, Ag_2, \varphi))]\!]$

```
1: $X \leftarrow SMC_c(Ag_1, Ag_3, \varphi, M)$
2: $Y \leftarrow SMC_{Ere}(C(Ag_1, Ag_2, \varphi), M)$
3: return $X \cap Y$
```