GENERATING, RANKING, AND ENACTING COMMITMENT PROTOCOLS

by

Akın Günay

B.S., Computer Engineering, Eastern Mediterranean University, 2005

M.S., Computer Engineering, Boğaziçi University, 2008

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering
Boğaziçi University
2013

GENERATING, RANKING, AND ENACTING COMMITMENT PROTOCOLS

APPROVED BY:

Assoc. Prof. Pınar Yolum (Thesis Supervisor)	....
Prof. H. Levent Akın	....
Assoc. Prof. Esra Erdem	....
Assoc. Prof. Tunga Güngör	....
Assist. Prof. Murat Şensoy	....

DATE OF APPROVAL: 30.09.2013

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to Pınar Yolum for her invaluable guidance, great patience, friendly attitude and encouraging support. This thesis would not be possible without her efforts.

I would like to thank my thesis committee members Levent Akın, Esra Erdem, Tunga Güngör, and Murat Şensoy for their valuable efforts and insightful comments. I would like to thank Michael Winikoff for his contributions in this thesis. His diligence and attention in details was a great inspiration for me.

I would like to thank Suzan Üsküdarlı for her great enthusiasm and tireless encouragements. I am deeply grateful to İlker Birbil and Cem Ersoy for their support.

I would like to thank all members of the MAS group. In particular, I would like to thank Fatma Başak Aydemir, Reyhan Aydoğan, Ayça Kundak Işıksal, Özgür Kafalı, Nadin Kökçiyan, and İlker Yıldırım.

I shared a lot with citizens of CmpE. I would like to thank all of them for their friendship and also for making our environment uniquely enjoyable. In particular, I would like to thank Hande Alemdar, İsmail Arı, Suzan Bayhan, Gül Çalıklı, Derya Çavdar, Serhan Daniş, Orhan Ermiş, Salim Eryiğit, Gaye Genç, Barış Gökçe, Didem Gözüpek, Onur Güngör, Gürkan Gür, Burak Gürdağ, Özlem Durmaz İncel, Yunus Emre Kara, Itır Karaç, Can Kavaklıoğlu, Can Komar, Bilgin Koşucu, Barış Kurt, Çetin "Dayı" Meriçli, Tekin Meriçli, Ergin Özkucur, Levent Özgür, Haşim Sak, Abuzer Yakaryılmaz, and Ahmet Yıldırım for their invaluable support.

My best friends Bircan Tuner and Can Berkol were always with me in my most desperate times. I am grateful to them for their invaluable friendship and support.

Finally, I am most grateful to my parents Birten and Metin for their great love, patience, support and endless faith in me.

This thesis has been partially supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) Scholarship Programs 2211 and 2214, TÜBİTAK project 106M472, the Turkish State Planning Organization (DPT) under the TAM Project (number 2007K120610) and Boğaziçi University Research Fund under grant 13A01P2.

ABSTRACT

GENERATING, RANKING, AND ENACTING COMMITMENT PROTOCOLS

Multiagent systems offer novel techniques to solve computational challenges that involve data interpretation, reasoning and decision making, without human intervention. An important aspect of every multiagent system is interaction among agents, which requires agents to employ regulation mechanisms to coordinate their actions. Commitment protocols provide an effective mechanism for this purpose. Typically, these protocols are defined at design time and embedded into agents' implementation. However, predefined commitment protocols are not adequate for large-scale, open multiagent systems, because of the variety of agents, changes in the agent preferences and changes in the environment. Accordingly, in this thesis we argue that agents should not rely on preexisting commitment protocols and they should be able to generate their own commitment protocols when needed, taking the current context of the multiagent system into account. In order to achieve that, we propose a three-phase agent process. In the first phase an agent generates a set of commitment protocols based on its goals, capabilities and other agents' services. For this purpose we propose two sound and complete algorithms that can efficiently generate commitment protocols. In the second phase, the generated commitment protocols are ranked from the generating agent's perspective. To achieve this we formulate a set of metrics that use cost, benefit and trustworthiness of commitment protocols to rank them. Finally, in the third-phase the agent negotiates with other agents over selected feasible commitment protocols to reach an agreement on a protocol for enactment. In this context we formalize commitment feasibility and provide an algorithm based on constraint satisfaction techniques to check if a set of commitments can be carried out. This three-phase process provides a complete method for agents to generate and enact commitment protocols on demand.

ÖZET

TAAHHÜT PROTOKOLLERİNİN OLUŞTURULMASI, SIRALANMASI VE YÜRÜRLÜĞE KONMASI

Çoketmenli sistemler insan müdahalesi olmadan yorumlama, muhakeme etme ve karar almayı içeren hesaplama problemlerinin çözümü için yeni teknikler sunmaktadır. Etmen iletişimi her çoketmenli sistemin temel bir bileşenidir. Etmen iletişiminde eşgüdümün sağlanabilmesi için bir düzenleme mekanizmasına ihtiyaç duyulur. Taahhüt protokolleri bu amaç için etkin bir mekanizma sağlar. Tipik olarak bu tür protokoller tasarım aşamasında tanımlanarak etmenlerin uygulamaları içine gömülürler. Ancak önceden tanımlı taahhüt protokolleri etmenlerin çeşitliliği, etmen tercihlerindeki değişiklikler ve ortam değişiklikleri nedeniyle büyük ölçekli ve açık çoketmenli sistemler için uygun değildirler. Buna bağlı olarak, bu tezde etmenlerin önceden tanımlı taahhüt protokollerine bel bağlamamaları gerektiğini ve ihtiyaç duyduklarında çoketmenli sistemin mevcut durumunu göz önüne alarak kendi taahhüt protokollerini oluşturabilir olmaları gerektiğini öne sürüyoruz. Buna ulaşmak için üç aşamalı bir etmen süreci öneriyoruz. İlk aşamada, etmen, hedeflerini, yeteneklerini ve diğer etmenlerin hizmetlerini temel alarak taahhüt protokolleri üretir. Bu amaçla, taahhüt protokollerini verimli olarak üreten iki algoritma önerilmiştir. İkinci aşamada, üretilen protokoller etmeninin bakış açısından sıralanır. Bu amaçla, taahhüt protokollerinin maliyetlerini, sağladıkları menfaati ve güvenilirliklerini kullanan ölçütler tanımlanmıştır. Son olarak, üçüncü aşamada etmen diğer etmenlerle pazarlık ederek yapılabilir protokollerden biri üzerinde anlaşmaya varmaya çalışır. Bunun için protokollerin yapılabilirliğini biçimsel olarak tanımlanmış ve protokollerin gerçekleştirilebilir olduğunun denetimi için kısıt sağlama tekniği tabanlı bir algoritma sunulmuştur. Bu üç aşamalı süreç taahhüt protokollerinin ihtiyaç duyulduğunda oluşturulması ve yürürlüğe konması için bir yöntem sunmaktadır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . v
ÖZET . . . vi
LIST OF FIGURES . . . x
LIST OF TABLES . . . xiii
LIST OF SYMBOLS . . . xiv
LIST OF ACRONYMS/ABBREVIATIONS . . . xvi
1. INTRODUCTION . . . 1
  1.1. Challenges . . . 5
  1.2. Contributions . . . 7
2. TECHNICAL PRELIMINARIES . . . 11
  2.1. Background on Commitments . . . 11
    2.1.1. Commitment Concepts . . . 11
    2.1.2. Commitment Protocols . . . 13
  2.2. Technical Framework . . . 16
  2.3. Running Example . . . 19
3. GENERATION OF COMMITMENT PROTOCOLS . . . 23
  3.1. Supporting Goals . . . 23
  3.2. ProtocolBased Algorithm . . . 28
    3.2.1. The Algorithm . . . 28
    3.2.2. Formal Properties of ProtocolBased Algorithm . . . 32
  3.3. GoalBased Algorithm . . . 34
    3.3.1. The Algorithm . . . 35
    3.3.2. Formal Properties of GoalBased Algorithm . . . 38
  3.4. Comparison of Algorithms and Experimental Results . . . 41
    3.4.1. Execution Strategies and Complexity of the Algorithms . . . 41
    3.4.2. Computational Result . . . 46
  3.5. Discussion . . . 55
4. RANKING OF COMMITMENT PROTOCOLS . . . 57

4.1. Utility of Protocols 58
4.1.1. Utility Ranking Method 58
4.1.2. Protocol Evaluation using Utility Ranking 61
4.2. Incorporating Risk 64
4.2.1. Trust Based Ranking Method 64
4.2.2. Protocol Evaluation using Trust Ranking 68
4.3. Discussion 74
5. ENACTMENT OF COMMITMENT PROTOCOLS 75
5.1. Negotiation 76
5.2. Feasibility of Commitments 78
5.2.1. Conflicts between Commitments 79
5.2.2. From Conflicts to Feasibility 83
5.3. Extended Technical Framework 85
5.3.1. State Transition 88
5.3.2. Possible Executions 91
5.3.3. Feasibility of Commitments 93
5.4. Computing Feasibility of Commitments 95
5.5. Feasibility and Analysis of Agent States 101
5.6. Formal Properties of CheckFeasibility Algorithm 109
5.7. Computational Results 112
5.8. Discussion 121
6. DISCUSSION 123
6.1. Related Work 123
6.1.1. Commitments 123
6.1.2. Generation 128
6.1.3. Ranking 131
6.1.4. Enactment 132
6.1.5. Other Topics 134
6.2. Conclusions 137
6.3. Future Directions 138
APPENDIX A: CSP DEFINITION 141

APPENDIX B: PROOFS 144
B.1. Proofs of Chapter 3: Generation of Commitment Protocols 144
B.2. Proofs of Chapter 5: Enactment of Commitment Protocols 152
REFERENCES 157

LIST OF FIGURES

Figure 1.1. The tree-phase process that is executed by the leading agent to come up with a commitment protocol to regulate its interaction while pursuing its goals. 8

Figure 2.1. Commitment protocol to regulate the e-commerce scenario. 14

Figure 2.2. A possible execution of the commitment protocol of the e-commerce scenario. 15

Figure 3.1. Support graph of the protocol π that supports G_{Cus}(*HaveFurniture*). 25

Figure 3.2. ProtocolBased algorithm. 30

Figure 3.3. GoalBased algorithm. 36

Figure 3.4. Execution trace of ProtocolBased algorithm on the running example. 43

Figure 3.5. Execution trace of GoalBased algorithm on the running example. 44

Figure 3.6. Data generation process with parameters $\gamma, \sigma, \rho, \varepsilon, \lambda$. 49

Figure 3.7. Comparison of execution times of the ProtocolBased and GoalBased algorithms based on the number of available services for each goal. 51

Figure 3.8. Comparison of execution times of ProtocolBased and GoalBased algorithms based on the number of preconditions ($\rho = 1$) and incentives ($\varepsilon = 1, \ldots, 5$) for each service. 52

Figure 3.9. Comparison of execution times of ProtocolBased and GoalBased algorithms based on the number of preconditions ($\rho = 2$) and incentives ($\varepsilon = 1, \ldots, 5$) for each service. 53

Figure 3.10. Comparison of execution times of ProtocolBased and GoalBased algorithms based on the number of preconditions ($\rho = 3$) and incentives ($\varepsilon = 1, \ldots, 5$) for each service. 54

Figure 5.1. Monotonic concession procedure to reach agreement on a commitment protocol. 77

Figure 5.2. Possible executions. 93

Figure 5.3. The CSP based CheckFeasibility algorithm to decide on the feasibility of a set of commitments. 96

Figure 5.4. Time intervals of the properties in the CSP Example. 101

Figure 5.5. Comparison of feasibility algorithm's execution time using different combinations of variable and value selection heuristics, for 20 feasible properties. 115

Figure 5.6. Comparison of feasibility algorithm's execution time using different combinations of variable and value selection heuristics, for 20 infeasible properties. 116

Figure 5.7. Comparison of feasibility algorithm's execution time using different combinations of variable and value selection heuristics, for 60 feasible properties. 117

Figure 5.8. Comparison of feasibility algorithm's execution time using different combinations of variable and value selection heuristics, for 60 infeasible properties. 118

Figure 5.9. Execution times of our feasibility algorithm and the DFS algorithm for various size feasible property sets. 119

Figure 5.10. Execution times of our feasibility algorithm and the DFS algorithm for various size infeasible property sets. 120

LIST OF TABLES

Table 2.1. Syntax of the framework elements. 16

Table 2.2. Conditions of the running example and their meanings. 20

Table 2.3. Capabilities of the customer. 21

Table 2.4. Beliefs of the customer. 21

Table 3.1. Generated protocols for the running example. 42

Table 3.2. Summary of parameters to generate the experimental data set. . . 46

Table 3.3. Example service and incentive data generated using $\gamma = 1$, $\sigma = 2$, $\rho = 1$, $\varepsilon = 1$ and $\lambda = 3$. 47

Table 4.1. Condition benefits. 61

Table 4.2. Capability costs. 62

Table 4.3. Customer's evaluation of protocol utility. 63

Table 4.4. Customer's trust in services. 69

Table 4.5. Customer's evaluation of protocol's expected utility values. 73

Table 5.1. Role and resource variations in commitments from the leading agent's point of view. 103

LIST OF SYMBOLS

$\mathcal{B}$	set of agent beliefs
c	commitment variable
C	commitment
$\mathcal{C}$	set of commitments
$\mathcal{D}$	domain knowledge
e	execution
f	agent capability variable
F	agent capability
$\mathcal{F}$	set of agent capabilities
g	agent goal variable
G	agent goal
$\mathcal{G}$	set of agent goals
I	incentive belief
$\mathcal{I}$	set of incentives
$\mathcal{N}$	CSP instance
p	property variable
P	property
$\mathcal{P}$	set of properties
q	condition variable
s	service variable
S	service belief
$\mathcal{S}$	set of services
u	condition variable
v	incentive variable
w	condition variable
$\mathcal{X}$	set of agents
x	leading agent
y	agent

χ	set of agents
δ^θ	condition state update function
δ^p	property state update function
δ^c	commitment state update function
η	agent state
κ	solution of a CSP problem
ϕ	CSP property variable
Φ	set of CSP property variables
π	commitment protocol
Π	set of commitment protocols
ψ	CSP resource variable
Ψ	set of CSP resource variables
θ	condition set

LIST OF ACRONYMS/ABBREVIATIONS

CSP	Constraint Satisfaction Problem
CTL	Computational Tree Logic
DFS	Depth-first Search
FIPA	Foundation for Intelligent Physical Agents
HTN	Hierarchical Task Network
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
KQML	Knowledge Query Manipulation Language
LM	Largest Minimum
MAX	Maximum of Domain
MID	Middle of Domain
MIN	Minimum of Domain
MCMAS	Model Checker for Multi-Agents Systems
RAN	Random in Domain
SD	Smallest Domain
SM	Smallest Minimum
TCP	Transmission Control Protocol
WWW	World Wide Web

1. INTRODUCTION

Internet technologies and applications that are built on top of them, such as World Wide Web and electronic mail have changed the way of computing in the last decades. They transformed personal computing devices from standalone processing units to nodes of a complex computing network, allowing exchange of large amount data in the global scale. However, these systems leave composition and interpretation of the exchanged data to humans, who use these devices. For example, World Wide Web (WWW) allows exchange of hypermedia content (e.g., Web pages), but, composition and interpretation of this content is done by humans. However, as a result of extensive use of these technologies, the volume of the available data on the Internet increases rapidly. Recent studies estimate the total amount of stored data in the World in 2007 as 295 exabytes [1] (which corresponds to roughly 42.7 gigabytes of data per person) and the daily Internet traffic in 2012 as 1.46 exabytes [2]. These numbers show that human-based interpretation of data in such systems is not feasible. Accordingly, application domains such as e-commerce, e-health and pervasive computing require novel computing paradigms that can interpret and reason about data and make decisions without requiring human intervention.

Intelligent agents and multiagent systems offer new techniques to meet the requirements of such domains. An agent is a persistent computation that perceives, reasons and acts in an environment in order to achieve a set of goals [3,4]. However, an agent's capabilities are usually not sufficient to achieve all of its goals. Still there may be other agents that provide services, which can be utilized by the agent to achieve its goals. If this is the case, agents start to interact and form a multiagent system. Nevertheless, an agent is an autonomous and self-interested entity that makes its own decisions without intervention of other agents (or humans). Hence, an agent does not take an action (e.g., providing its capabilities to others) just because another agent requests so. Moreover, an agent's goals usually conflict with other agents goals. In such cases, an agent considers its own interests before others and acts in order to increase its own benefit.

To clarify these concepts, consider a typical e-commerce system that includes a merchant, a customer and a courier. In such a system, the merchant aims to earn money by selling goods for a high profit. On the other hand, the customer aims to purchase the goods that she desires, preferably for a low price. Finally, the courier aims to earn money by conveying purchased goods from the merchant to the customer. Each party in this system has a set of capabilities to do certain tasks and reach her goals. For example, the merchant is capable of selling goods, the courier is capable of delivering goods, and the customer is capable of making payments. In an agent oriented representation of this system, the merchant, the courier and the customer are represented by different agents. Each agent has its own goals, preferences and capabilities that reflect the real situation of the actual party that the agent represents. By examining the given agent capabilities, it is clear that the agents are not capable to achieve their goals on their own. Hence, they have to interact with each other in order to achieve their goals. For example, the merchant depends on the courier for the delivery of sold goods. Hence, the merchant should cooperate with the courier to achieve its own goal. But agents do not take an action just because another agent requests so. For example, the courier does not simply deliver goods just because the merchant requests that. Instead, the courier demands money to make the delivery. Beside cooperation, competition also occur frequently in such systems, when agents' goals conflict with each other. For example, if there was a second merchant, the two merchants would compete with each other to sell goods to customers.

As this example demonstrates, interaction is a major component of any multiagent system and in order to successfully realize any type of interaction (e.g., cooperation or competition), agents need a regulation mechanism that allows them to coordinate their actions. Communication protocols such as IP, TCP and HTTP have been successfully used to regulate interaction among nodes in computer networks and distributed systems for several decades [5, 6]. A communication protocol is a set of well-defined messages and exchange rules over these messages that regulate the communication among the nodes in a network. For example, the HTTP protocol specifies how a client can request a hypermedia (e.g., Web page) from a server and how the server may respond to such a request [7]. For this purpose, the HTTP protocol provides a predefined message for each possible request and response that can be made by the client and the server while communicating. In this way, nodes in a network that have a compliant implementation of the HTTP protocol can communicate with each other using this protocol.

Although communication protocols are effective for systems in which interaction requirements of entities are well known (e.g., WWW), they are not adequate to regulate agent interaction, mainly because they restrict agents to a predefined set of actions that should be executed in a certain order. In a multiagent system agents are autonomous entities and may achieve conditions by using different alternative actions depending on their own preferences. Hence, a regulation mechanism for a multiagent system should respect to agents' autonomy and should not enforce them to follow predefined action sequences. Instead, the regulation mechanism should define the consequences of the agents' actions and allow them to decide on their own actions using their own reasoning based on their own preferences [8]. Note that a communication protocol may try to take every possible action that can be performed by agents into account in order not to restrict agents. However, due to open nature of multiagent systems the number of actions that can be performed by agents is usually large, which makes such an effort infeasible. In addition to that, agents in a multiagent system are typically developed by independent vendors following different design methodologies and implementation techniques. Thus, in order to be effectively used in accordance with any design methodology and implementation technique, the regulation mechanism of a multiagent system should provide an appropriate abstraction to govern agent interaction. More specifically, the mechanism should not require the regulations to be embedded directly into agents' implementation. Instead, the mechanism should allow agents to interpret and reason about the regulations independent from their internal details [9].

Accordingly, development of adequate mechanisms to regulate agent interaction is a major challenge of multiagent systems research [10,11]. To overcome this challenge, multiagent systems community developed a novel approach based on *(social) commitments* [12,13]. A commitment is a contractual binding between two agents in which one agent commits to another to bring about a property, if a condition holds. For example, the merchant commits to deliver the goods to the customer, if the customer pays. A commitment regulates agents' interaction by making participating agents responsible for certain conditions (e.g., the merchant is responsible for delivering the goods to the customer). In many cases, the entire interaction among the agents cannot be regulated by a single commitment. In such cases, multiple commitments are considered together and a commitment protocol is established [8, 14–16]. A commitment protocol is a set of commitments that are related to each other in the context of an interaction, such that once the protocol's commitments are enacted and fulfilled the world evolves to a certain desired state as the result of the agents' actions.

Commitments provide a natural way to regulate agent interaction respecting the agents' autonomy, since they do not enforce agents to take certain actions. For example, the merchant's above commitment to the customer does not enforce the merchant to take a predefined action to bring about the delivery of the goods, but only requires realization of this condition. The merchant could realize this condition in various ways, such as delivering the goods using her own capabilities or delegating the delivery to a courier agent. Besides, commitments do not enforce agents to follow predefined action sequences while interacting, since agents are free to enact and fulfill their commitments in any order they see fit to their preferences. For example, the merchant might choose to deliver the goods as promotion, without waiting for the customer's payment. Moreover, interpretation of commitments is independent from agents' internal design and implementation. Therefore, participating agents can reason about the results of their actions in a mutual way. Finally, since commitments are defined over conditions, but not over actions, they are more natural to represent complex interactions among agents compared to communication protocols [8, 9, 17].

Reconsider the e-commerce system that we discussed above. In this system, the interaction between the merchant and customer involves payment by the customer and delivery by the merchant. However, since the merchant is not capable of delivering, she actually delegates this task to the courier. By realizing these interactions, the agents can satisfy certain conditions that are needed to achieve their goals. These interactions can be defined as a protocol using commitments as follows. The merchant commits to the customer to deliver the goods, if the customer pays for the goods, and the courier commits to the merchant to deliver the goods to the customer, if the merchant pays the delivery costs. Once these commitments are fulfilled, all the agents achieve their goals. This protocol specifies what the agents should bring about in order to reach their goals over two commitments. On the other hand, since the protocol does not impose certain actions, the agents are free to act as they see fit in order to satisfy the condition. As a side note, we want to emphasize the difference between a commitment protocol and plans as used in the artificial intelligence literature [18]. A plan is a strict sequence of actions that should be followed by agents to achieve a goal. Hence, once there is a plan, all involved agents should execute the plan as it is. On the other hand a commitment protocol states how the world evolves as a result of agents interactions. However, a commitment protocol can be used by agents as a guideline to build a plan for execution.

1.1. Challenges

Typically, commitment protocols are defined at the design time of a multiagent system and are embedded into agents' implementation [15, 19, 20]. This approach is beneficial since it simplifies the design and implementation processes of agents. However, this approach also restricts agents to execute in a limited predefined context. For closed systems where both the agents and the environment are known in advance, providing design time protocols may be applicable, since the system designer already has access to the possible modes of interaction. However, in open systems where heterogeneous agents freely act in the system and both the context and the environment change dynamically, designing protocols up front is not sufficient to establish successful interaction because of three major reasons.

- *Variety of agents:* Since a multiagent system is an open system, agents can freely join and leave the system at any time. Besides, due to the heterogeneity of agents, their compliance with the system's predefined protocols is also not guaranteed all the time. For example, a typical e-commerce protocol similar to the one we outlined above does not work with a barter agent who expects a different type of good instead of the customer's payment.

- *Change in agent preferences:* Based on various factors, agents' preferences can change over time. For example, an agent that runs out of cash may need to come up with a protocol in which the agent either delays its payment or uses a different commodity in return. Since the need becomes evident only at run-time, the agent should have the means to adapt and formulate a protocol that reflects its preferences.
- *Change in environment:* Even if the agents are fixed, the environment may evolve in a way requiring agents to adapt their actions to these changes. Such adaptations usually require novel interactions between the agents. For example, if the courier is temporarily unavailable, then the previous e-commerce protocol will not suffice. Hence, the agents need to find a way to carry out an alternative protocol, such as a purchase and pickup protocol where the customer is expected to pick up the goods after payment. Such cases occur because the environment is dynamic. However, it is difficult to account for all such cases when designing a protocol at design time.

Hence, even though assuming the existence of fixed predefined protocols simplifies the agent development process, this approach fails in open dynamic multiagent systems. Accordingly, in this thesis we argue that *to regulate their interactions while pursuing their goals, agents in a multiagent system should be able to generate their own commitment protocols, taking the current context of the multiagent system into account.* Once agents start to generate their own commitment protocols, two new important questions emerge. The first question is "How can we compare two commitment protocols?". This question emerges, because in order to achieve a goal, agents may generate more than one commitment protocol such that each protocol considers interaction with different agents and use of different capabilities. Accordingly, agents should employ a method to compare and select the most appropriate commitment protocol from a set of (generated) protocols that allow them to reach their goals. The second question is "How can we decide about the feasibility of a commitment protocol?". Typically, agents allocate resources while fulfilling their commitments. However, usually agents have limited resources and if they are over-committed, they cannot fulfill their commitments. Accordingly, agents should have a method to decide whether a generated commitment protocol is feasible (i.e., the agents have enough resources to fulfill all the commitments in the protocol) in the current context of the multiagent system.

1.2. Contributions

Following the challenges we have identified, we make three major contributions in this thesis.

- We develop two algorithms for efficient generation of commitment protocols. These algorithms can be used by agents to generate commitment protocols in order to achieve their goals in the current context of a multiagent system. We show formal properties of these algorithms, such as soundness and completeness, and examine their performance through computational experiments.
- We develop a method that uses an agent's utilities and trust to others to compare and rank commitment protocols, which can be used by agents to select the most favorable protocol from a set of protocols. We evaluate our method using a case study.
- We develop an algorithm that utilizes constraint satisfaction techniques to decide on the feasibility of commitment protocols considering the resources available to the agent and the temporal constraints of commitments. We show formal properties of our algorithm, such as soundness and completeness, and evaluate its performance through computational experiments.

In order to generalize our contributions, we organize them in the context of a three-phase process that can be executed by an agent in a multiagent system. We present the overview of this process in Figure 1.1. The overall objective of this process is to come up with a commitment protocol that can be used by the agent that runs the process and to regulate its interaction with others while pursuing its goals. Since this agent takes care of most the process, we call this the *leading agent*. In the first phase, the leading agent that aims to reach a goal proactively generates a set of commitment protocols. In the second phase, the leading agent ranks the generated protocols with respect to its own preferences. Finally, in the last step, the leading agent negotiates with others to reach an agreement on one of the generated protocols that is feasible, for enactment.

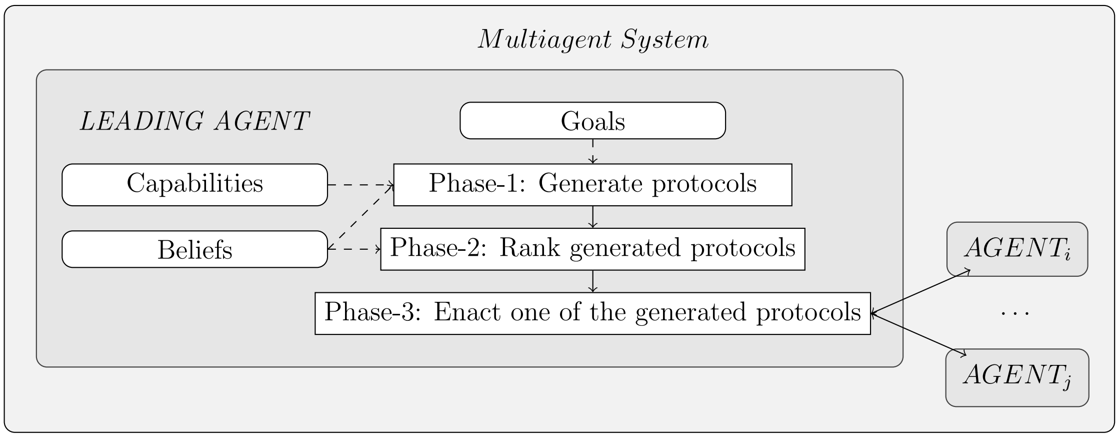
Multiagent System
LEADING AGENT
Goals
Capabilities
Phase-1: Generate protocols
Beliefs
Phase-2: Rank generated protocols
Phase-3: Enact one of the generated protocols
$AGENT_i$
...
$AGENT_j$

Figure 1.1. The tree-phase process that is executed by the leading agent to come up with a commitment protocol to regulate its interaction while pursuing its goals.

As it is customary in multiagent systems, each agent has its own set of goals to achieve and own set of capabilities to perform certain actions. If agents find it beneficial, they may provide their capabilities to other agents as services. Accordingly, each agent has also a set of beliefs about other agents' services and expectations. Finally, all agents share a common domain knowledge about the dynamics of the underlying environment (not shown in figure for readability).

Now, we explain the phases of the agent process and our corresponding contributions in detail. In the first phase, the leading agent proactively generates a set of commitment protocols that satisfy its goals once executed in the context of the multiagent system. To achieve that we develop two algorithms based on depth-first traversal and divide and conquer strategies. The algorithms generate commitment protocols to satisfy the leading agent's goals, taking it's capabilities and also other agents' services into account. When a service of another agent is needed, the generated protocol includes a commitment from that agent for the provision of the required service. However, since agents are autonomous and self-interested, they will not provide their services to others for free. Hence, in order to create incentive and assure service provision, the generated commitments should offer something in return to the provider agent. For this purpose, the generation algorithms take the beliefs about other agents' expectations into account and use them to create incentive.

In the second phase, the set of commitment protocols that are generated in the previous phase are ranked according to the leading agent's preferences. For this purpose we develop a utility based ranking method for commitment protocols. Our method computes the utility of a commitment protocol from the leading agent's perspective using the benefit (i.e., what the leading agent receives) and the cost (i.e., what the leading agent spends) imposed to the leading agent when the protocol is executed. However, execution of a commitment protocol implies interaction with other autonomous and self-interested agents that may choose to violate their commitments, when they find it more beneficial for themselves. Accordingly, fulfillment of the commitments in a commitment protocol is not guaranteed. Hence, our ranking method takes the leading agent's trust in others to realize their commitments also into account and applies a discount on the utility according the level of trust in other agents.

Finally, in the third phase, the leading agent negotiates with others in order to reach an agreement on one of the generated commitment protocols for enactment. Beside providing a negotiation mechanism, our main contribution in this phase is the development of an algorithm to decide on the feasibility of commitment protocols. While negotiating on the enactment of a commitment protocol, agents have to determine whether the commitments in the protocol can be fulfilled in the current context of the multiagent system, taking the available resources and time constraints over the commitments into account. In other words, agents should check the feasibility of commitments in a protocol before they enact the protocol. Accordingly, first we formalize the feasibility of commitment protocols. Then, based on this formalization, we develop an algorithm that utilizes constraint satisfaction techniques to decide on the feasibility of commitment protocols. This method takes availability of resources into account by considering existing and expected commitments of the agents in the current context of the multiagent system. Besides, we enhance this method by introducing temporal constraints over commitments, which allows use of our method in realistic settings.

The rest of this thesis is organized as follows. In Chapter 2, first we provide an overview of commitments. Then we define the technical framework that we use in the rest of the paper. Also we provide a running example which we use to clarify our technical material. In Chapter 3, we present our algorithms for generating commitment protocols. Then we show formal properties of these algorithms. Finally, we present the results of our computational experiments to evaluate the performance of the algorithms. In Chapter 4, we present our ranking method for commitment protocols and evaluate the method over our running example. In Chapter 5, we present the feasibility concept of commitments and develop our constraint satisfaction based method to decide on the feasibility. We evaluate performance of our method over a set of computational experiments. Finally in Chapter 6, we discuss our work in relation to the literature. We conclude this thesis by summarizing our contributions and outlining our future work.

2. TECHNICAL PRELIMINARIES

In this chapter we provide the technical preliminaries required for the rest of this thesis. We start with a conceptual overview of commitments and protocols. Then we present our technical framework that we use in the rest of this thesis. Finally, we present a running example, which we use to explain the methods we develop in the subsequent chapters.

2.1. Background on Commitments

2.1.1. Commitment Concepts

A *commitment* is a contractual binding made from one agent to another to bring about a property, if a certain condition holds [9, 12, 13]. Typically, a commitment is represented as $C(\mathit{debtor}, \mathit{creditor}, \mathit{antecedent}, \mathit{consequent})$ which states that the *debtor* agent is committed to the *creditor* agent to bring about the *consequent*, if the *antecedent* holds. A commitment is called *conditional* when neither the antecedent nor the consequent of the commitment holds. If the antecedent holds without the consequent, then the commitment is *active*. If the consequent is satisfied, then the commitment is *fulfilled*. If the consequent is failed to be satisfied while the antecedent holds, then the commitment is *violated*.

Consider the commitment "the merchant is committed to the customer to deliver the goods, if the customer pays for the goods" from our e-commerce scenario to clarify these concepts. We represent this commitment as $C(Mer, Cus, \mathit{GoodsPaid}, \mathit{GoodsDelivered})$ in which Mer represents the merchant, who is the debtor, and Cus represents the customer, who is the creditor. The condition *GoodsPaid*, which is the antecedent, represents payment of the customer for the goods and *GoodsDelivered*, which is the consequent, represent delivery of the goods. Assume that neither *GoodsPaid* nor *GoodsDelivered* holds initially. When the merchant first makes the commitment, the commitment is conditional. When the merchant pays, *GoodsPaid* starts to hold and the commitment is active. When the merchant delivers the goods, *GoodsDelivered* is satisfied and the commitment is fulfilled. On the other hand, if the merchant fails to satisfy *GoodsDelivered* (i.e., the merchant does not deliver), then the commitment is violated. Commitment violation can be captured in different ways depending on the preferences of the modeled system. In some systems a deadline may be associated with the commitment's consequent and if the consequent is not satisfied before this deadline, then the commitment is treated as violated. For instance, in the above commitment the merchant might be committed to deliver the goods in three days after the customer made the payment. In other systems where there is no associated deadline, the debtor may explicitly state that it is not going to satisfy the consequent even though the commitment is active. If this is the case, then the commitment is treated as violated.

Note that the same treatment can also be made for the antecedent of a commitment. A deadline may be associated with the antecedent and if the antecedent does not hold before this deadline then the commitment expires. For example, the merchant would be committed to the customer to deliver the goods, only if the customer paid in three days after the creation of the above commitment. On the other hand, in the cases where no deadline is associated with the antecedent, the creditor may explicitly release the debtor from its commitment.

Commitments provide a high-level abstraction to model and regulate agent interaction. This is achieved by associating the results of agents' actions to the states of commitments (e.g., customer's payment makes merchant's commitment active). Accordingly, agents can use commitments in their reasoning process in order to make decisions while interacting with other agents. For example, considering the commitment $C(Mer, Cus, GoodsPaid, GoodsDelivered)$, the customer can conclude that if she pays for the goods and satisfies *GoodsPaid*, then the merchant is going to be committed to deliver the goods by satisfying *GoodsDelivered*. Hence, based on this commitment the customer can reason about how she should interact with the merchant in order to achieve *GoodsDelivered*.

Note that every agent (e.g., customer or merchant) that considers a commitment (e.g., $C(Mer, Cus, GoodsPaid, GoodsDelivered)$), infers the same conclusions from that commitment, independent from its internal design and implementation, since commitments are public and their states are defined over public global states, which are interpreted by all agents in the same way. This independence in the interpretation of commitments allow interoperability among heterogeneous agents. Besides, commitments also do not interfere with agents' autonomy. In the previous example, neither the customer nor the merchant are enforced to take certain actions because of the commitment . For example the customer is free to pay for the goods. Furthermore, even if the customer decides to pay, the commitment does not specify how the customer should do this. The commitment only states that the world should be in a certain state (i.e., $GoodsPaid$ should hold) in order to make the commitment active. For example, the customer herself might satisfy this condition or she might delegate it to another agent (e.g., a bank pays on behalf of the customer). On the other hand, the merchant might also choose to violate her commitment. This might cause a sanction to be applied to the merchant depending on the policy of the multiagent system [21, 22]. However, the merchant might choose to take this risk, if violation was more beneficial from her point of view. Finally, commitments also support flexible execution, since they do not specify an order on the actions of the agents. Suppose that in our running example the merchant decided to deliver the goods to the customer for free as promotion. Then, the merchant could deliver the goods without waiting the payment of the customer (i.e., while the commitment was conditional) and fulfilled her commitment.

2.1.2. Commitment Protocols

While acting in a multiagent system, an agent interacts with many other agents about various conditions in order to achieve its goals. Accordingly, in most of the cases a single commitment is not enough to regulate all the aspects of an agent's interaction. In such cases, a set of commitments that are related to each other in the context of an interaction are considered together as a commitment protocol [8, 14–16].

In Figure 2.1 we present such a commitment protocol to regulate the e-commerce scenario that we discussed earlier. In the figure, the rectangles are the agents and the

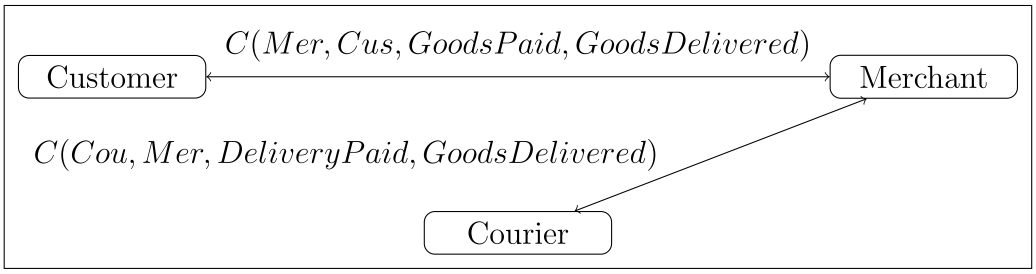
$C(Mer, Cus, GoodsPaid, GoodsDelivered)$
Customer
Merchant
$C(Cou, Mer, DeliveryPaid, GoodsDelivered)$
Courier

Figure 2.1. Commitment protocol to regulate the e-commerce scenario.

bidirectional edges labeled by a commitment shows the binding relation between the agents because of that commitment. This protocol includes the following commitments:

- $c_1 = C(Mer, Cus, GoodsPaid, GoodsDelivered)$
- $c_2 = C(Cou, Mer, DeliveryPaid, GoodsDelivered)$

In c_2, *Cou* corresponds to the courier and *DeliveryPaid* represents the payment that is made to the courier for the delivery. We use the other agents and conditions as defined before. Assuming that none of the conditions hold initially, this protocol can be executed by the agents as we show in Figure 2.2. In the figure the rounded rectangles on the top are the agents. Time flows downwards. The edges from black filled circles to rectangles show actions of agents and their results. The agent that is aligned with the black filled circle of an edge is the agent that performs the action. The result of the action is showed in destination rectangle of the edge and aligned with the relevant agent.

Initially, both commitments c_1 and c_2 are conditional. The customer initiates the execution by paying for goods at t_1. As a result, *GoodPaids* starts to hold and c_1 becomes active. Hence, the merchant has to deliver the goods to the customer (i.e., should satisfy *GoodsDelivered*). Although the merchant herself is not capable of delivering the goods, it is the creditor of c_2, in which the courier is obligated to deliver the goods. Hence, it is enough for the merchant to make c_2 active to satisfy delivery of the goods. Accordingly, the merchant pays to the courier for the delivery at t_2. As a result, *DeliveryPaids* starts to hold and c_2 becomes active. Finally, in order to fulfill her commitment, the courier delivers the goods to the customer at t_3. As a result,

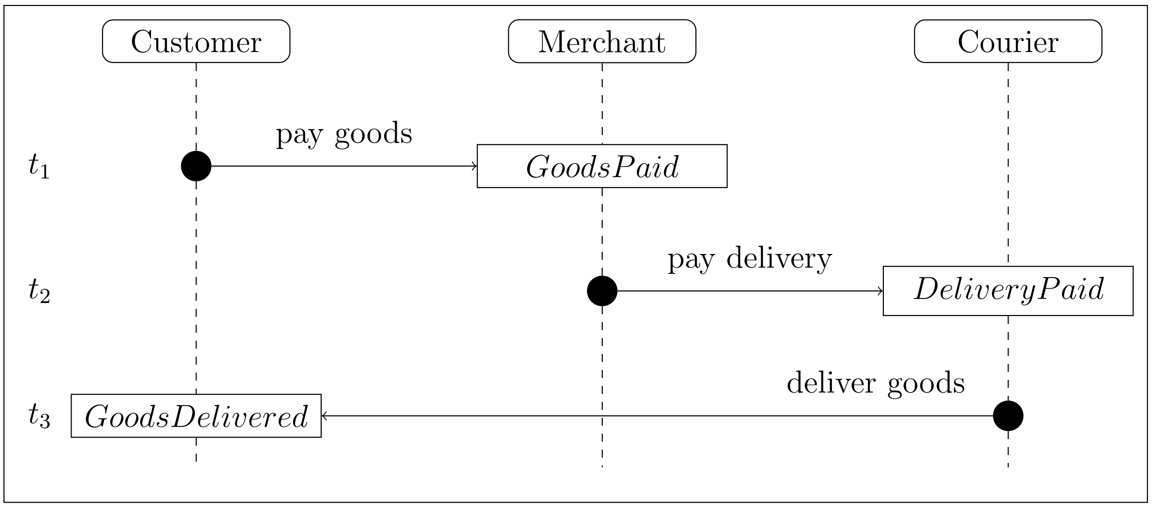

Figure 2.2. A possible execution of the commitment protocol of the e-commerce scenario.

GoodsDelivered starts to hold and both c_1 and c_2 become fulfilled.

Alternative executions of this protocol are possible. For example, assume that the merchant wants to promote a new product. Hence, she decides to send the product to the customer for free. In this situation, without expecting any payment from the customer, the merchant pays to the courier for the delivery of the good. As a result, *DeliveryPaid* starts to holds and c_2 becomes active. Accordingly, in order to fulfill her commitment, the courier delivers the goods to the customer. As a result, *GoodsDelivered* starts to hold and both c_1 and c_2 become fulfilled. Remember that fulfillment of a commitment is independent of its antecedent. Hence, once *GoodsDelivered* start to hold, c_1 becomes fulfilled independent from the state of the antecedent *GoodsPaid*.

As the above two cases demonstrate, a commitment protocol allows flexible interaction of agents. Different than the traditional communication protocols, such as IP, TCP and HTTP, a commitment protocol does not enforce agents to follow any predefined action sequences. Hence, agents are free to fulfill (or even violate) their commitments in any order they see fit.

2.2. Technical Framework

Here we present the technical framework that we use in the rest of this thesis. First, we present the syntax of the framework elements in Table 2.1.

Table 2.1. Syntax of the framework elements.

Goal	$\rightarrow$	G_{AgentID}(Condition)
Capability	$\rightarrow$	F_{AgentID}(ConditionSet, Condition)
Service	$\rightarrow$	S_{AgentID}(AgentID, ConditionSet, Condition)
Incentive	$\rightarrow$	I_{AgentID}(AgentID, Condition, Condition)
Belief	$\rightarrow$	Service | Incentive
Commitment	$\rightarrow$	C(AgentID, AgentID, ConditionSet, Condition)
ConditionSet	$\rightarrow$	Condition | {Condition, ConditionList}
ConditionList	$\rightarrow$	Condition | Condition, ConditionList
Condition	$\rightarrow$	*any propositional symbol*
AgentID	$\rightarrow$	*any agent identifier*

A condition element represents a condition in the real world that the multiagent system represents. We use the variable $u, u_i, u, q, q_u, q', w, w_i, w'$ over conditions. We also note trivially holding conditions using $\top$ symbol. A condition set includes any number of conditions and it holds only if all the conditions in the set hold. We use the variables $\theta, \theta_i, \theta'$ over conditions sets. Note that when a condition set includes only a single condition we omit the enclosing brackets for readability.

An agent is an entity that acts in an environment to achieve its goals. Each agent has a set of capabilities to perform certain tasks (i.e., bring about conditions) and has a set of beliefs about other agents. We use the variable x for the leading agent throughout the thesis. Besides, we use the variables y, y_i, y' for any other agent.

Definition 2.1. *(Agent) An* agent *is a three tuple $\langle \mathcal{G}, \mathcal{F}, \mathcal{B} \rangle$ where $\mathcal{G}$ is the set of agent's goals, $\mathcal{F}$ is the set of agent's capabilities and $\mathcal{B}$ is the set of agent's beliefs.*

A goal of an agent is defined over a condition. The agent achieves a goal when the condition of the goal holds. We use variables g, g_i, g' over goals.

Definition 2.2. *(Goal) $G_x(u)$ denotes the* goal *of the agent x to bring about the condition u. For example, $G_{Cus}(GoodsDelivered)$ denotes that $GoodsDelivered$ is a goal of Cus.*

An agent has a set of capabilities, which can be used by the agent to bring about certain conditions. In general, usability of a capability depends on a precondition (e.g., a customer can make payments, if she has enough money), which is a set of conditions. We use variables f, f_i, f' over capabilities.

Definition 2.3. *(Capability) $F_x(\theta, u)$ denotes the* capability *of the agent x to bring about the condition u, if the precondition θ holds. For example, $F_{Cus}(HasMoney, GoodsPaid)$ denotes that Cus can bring about $GoodsPaid$, if $HasMoney$ holds (i.e., customer has enough money to pay).*

Usually, agents have limited capabilities and they cannot achieve all of their goals using only their own capabilities. In such situations agents can benefit from other agents' capabilities, which are provided as services (e.g., a merchant provides a service to sells goods). These services may have preconditions (e.g., merchant delivers goods, only if the delivery address is provided). Each agent has a set of beliefs about other agents' services. We use variables s, s_i, s' over service beliefs.

Definition 2.4. *(Service Belief) $S_x(y, \theta, u)$ denotes that the agent x believes that the agent y can provide a* service *to bring about the condition u, if the precondition θ holds. For example, $S_{Cus}(Mer, AddressProvided, GoodsDelivered)$ denotes Cus believes that Mer provides a service to bring about $GoodsDelivered$, if $AddressProvided$ holds.*

In general, agents do not provide their services for free. Hence, in order to request a service from an agent, the requesting agent should offer something appropriate in return to create incentive to the provider (e.g., offering payment to a merchant when it sells a good). Note that an incentive for the provision of a service is different than the precondition of a service. A service precondition defines the conditions that is necessary to hold in order to realize the service. On the other hand, the incentive defines what the provider agent expects in return to its service, which depends on the current context of the environment. As in the case of service beliefs, each agent has a set of beliefs about the possible incentives of services provided by other agents. We use variables v, v_i, v' over incentive beliefs.

Definition 2.5. *(Incentive Belief) $I_x(y, w, u)$ denotes that the agent x believes that the agent y considers satisfaction of the condition w as an* incentive *for its services to bring about the condition u. For example, $I_{Cus}(Mer, GoodsPaid, GoodsDelivered)$ denotes Cus believes that Mer accepts $GoodsPaid$ as an incentive for her services in which she satisfies $GoodsDelivered$.*

An agent may acquire the information (i.e., beliefs) about the other agents' services and possible incentives for these services in various ways. For instance, in small and well-defined domains, this information may already be encoded as part of the agent's domain knowledge. In addition to this, such information is usually publicly advertised by the service providers. Even if the domain knowledge of an agent is incomplete or the other agents' services are not advertised, the agent may still infer this information by combining its partial domain knowledge with its observations from previous interactions. Accordingly, in the rest of this thesis we assume that each agent has a set of beliefs about the services of the other agents and their corresponding incentives. However, these beliefs may be incomplete (e.g., an agent may not be aware of all provided services), out of date (e.g., some services may not be provided any more) or even be wrong (e.g., a provider may expect something different as an incentives than what the agent believes). We assume that agents are capable of revising their beliefs in the light of new information as is customary in the literature [23].

Agents interact with each other through provision of their services and these interactions are regulated by commitments. A commitment is a social contract between a debtor and a creditor where the debtor commits to the creditor to bring about the consequent, if the antecedent holds.

Definition 2.6. *(Commitment)* $C(x, y, \theta, u)$ *denotes the* commitment *of the debtor agent* x *to the creditor agent* y *to bring about the consequent* u *if the antecedent* θ *holds. For example* $C(Mer, Cus, GoodsPaid, GoodsDelivered)$ *states that* Mer *is committed to* Cus *to bring about* $GoodsDelivered$*, if* $GoodsPaid$ *holds.*

A commitment is created typically by its debtor. For operational purposes, when a commitment is created, the debtor informs the creditor about this event by sending a predefined message to the creditor. Similarly, when agents take future actions that affect their commitments, they send certain messages to inform the other participants of these commitments. Here we do not show these messages and assume that they are available as defined in the literature [24, 25]. Note that here we also do not explicitly represent commitment states that we explained in Section 2.1. We will consider them later in Chapter 5.

A set of commitments in the context of an interaction is called a *commitment protocol*. We use variables π, π_i, π' over commitment protocols.

Definition 2.7. *(Commitment Protocol) A set of commitments* $\{c_i, \ldots, c_j\}$ *is a* commitment protocol.

A multiagent system is a system of agents and the commitment protocols among these these agents.

Definition 2.8. *(Multiagent System) A* multiagent system *is a two tuple* $\langle \mathcal{X}, \Pi \rangle$ *where* $\mathcal{X}$ *is a set of agents and* Π *is a set of commitments protocols among the agents in* $\mathcal{X}$*.*

2.3. Running Example

Here we present a running example about trading and building furniture. In our running example there are five agents, namely a customer (Cus), two builders (Bui_1 and Bui_2), a merchant (Mer) and a retail store (Ret). The customer wants to have a certain type of furniture. The first builder (i.e., Bui_1) offers a service to

Table 2.2. Conditions of the running example and their meanings.

ID	Condition	Meaning
u_1	$HaveFurniture$	customer owns furniture
u_2	$HaveMaterials$	customer owns materials
u_3	$HaveTools$	customer owns tools
u_4	$MaterialsPaid$	customer has paid to retailer for materials
u_5	$ToolsPaid$	customer has paid to retailer for tools
u_6	$FurniturePaid$	customer has paid to merchant for furniture
u_7	Bui_1Paid	customer has paid service cost to first builder
u_8	Bui_2Paid	customer has paid service cost to second builder
u_9	$Bui_1MaterialsProvided$	customer has provided materials to first builder
u_{10}	$Bui_2MaterialsProvided$	customer has provided materials to second builder
u_{11}	$ToolsProvided$	customer has provided tools to second builder

build custom furniture. However, as a precondition of this service the builder requires the materials to build the furniture to be provided. Similarly, the second builder (i.e., Bui_2) also offers a service to assemble furniture. However, the second builder, who is not a professional builder, requires both the materials and also the tools to be provided as a precondition. On the other hand, the merchant sells ready to use furniture and the retail store sells tools and materials for building furniture. All agents expect to be paid for their services.

We summarize the conditions we use in our running example in Table 2.2. For example, $HaveFurniture$ holds when the customer owns the furniture that she desires. Similarly, $HaveMaterials$ and $HaveTools$ conditions hold when the customer owns the building materials and tools, respectively. Conditions that include payments (e.g, $MaterialsPaid$) reflect the payments from the customer to the corresponding agents. $Bui_1MaterialsProvided$ and $Bui_2MaterialsProvided$, hold when the building materials are provided to the first and second builders, respectively, by the customer. Finally, $ToolsProvided$ holds when the tools are provided to the second builder by the customer.

Table 2.3. Capabilities of the customer.

ID	Capability
f_1	$F_{Cus}(HaveTools, ToolsProvided)$
f_2	$F_{Cus}(HaveMaterials, Bui_1MaterialsProvided)$
f_3	$F_{Cus}(HaveMaterials, Bui_2MaterialsProvided)$
f_4	$F_{Cus}(\top, MaterialsPaid)$
f_5	$F_{Cus}(\top, ToolsPaid)$
f_6	$F_{Cus}(\top, FurniturePaid)$
f_7	$F_{Cus}(\top, Bui_1Paid)$
f_8	$F_{Cus}(\top, Bui_2Paid)$

We examine the case from the customer's perspective (i.e., we consider the customer as the leading agent). The goal of the customer is $g_1 = G_{Cus}(HaveFurniture)$ (i.e., $\mathcal{G}$ of Cus is $\{g_1\}$). We list the capabilities of the customer (i.e., content of $\mathcal{F}$ of Cus) in Table 2.3. The capability f_1 states that if the customer has tools, then she can provide them to the second builder. Similarly, the capabilities f_2 and f_3 state that the

Table 2.4. Beliefs of the customer.

ID	Belief
s_1	$S_{Cus}(Ret, \top, HaveMaterials)$
s_2	$S_{Cus}(Ret, \top, HaveTools)$
s_3	$S_{Cus}(Mer, \top, HaveFurniture)$
s_4	$S_{Cus}(Bui_1, Bui_1MaterialsProvided, HaveFurniture)$
s_5	$S_{Cus}(Bui_2, \{Bui_2MaterialsProvided, ToolsProvided\}, HaveFurniture)$
v_1	$I_{Cus}(Ret, MaterialsPaid, HaveMaterials)$
v_2	$I_{Cus}(Ret, MaterialsPaid, HaveTools)$
v_3	$I_{Cus}(Ret, ToolsPaid, HaveTools)$
v_4	$I_{Cus}(Ret, ToolsPaid, HaveMaterials)$
v_5	$I_{Cus}(Mer, FurniturePaid, HaveFurniture)$
v_6	$I_{Cus}(Bui_1, Bui_1Paid, HaveFurniture)$
v_7	$I_{Cus}(Bui_2, Bui_2Paid, HaveFurniture)$

customer can provide building materials to the first and second builders, respectively. However, in order to use these capabilities the customer has to own the materials beforehand. Capabilities $f_4 - f_8$ state that the customer is capable to make payments to other agents for their respective services. We assume that the customer has enough money to make all the payments. Hence, none of these capabilities has a precondition.

Finally, we list the customer's beliefs about the other agents' services and incentives (i.e., content of $\mathcal{B}$ of Cus) in Table 2.4. s_1 and s_2 denote the services of the retailer to sell the materials and the tools, respectively. s_3 is the service of the merchant to sell the furniture. s_4 and s_5 are the services of the first and second builders, respectively, to build the furniture. Note that while s_1, s_2 and s_3 have no preconditions, s_4 has the precondition $MaterialsProvided$ and s_5 has the preconditions $MaterialsProvided$ and $ToolsProvided$. $v_1 - v_7$ states that other agents expect payments for their services.

3. GENERATION OF COMMITMENT PROTOCOLS

As we discussed earlier agents use protocols to regulate their interaction and typically, such protocols are defined at design time and are embedded into agents' implementation. However, predefined protocols limit agents' autonomy. Besides, because of the open nature of the multiagent systems, in general it is not possible to develop protocols that support every possible situation, which may occur in the actual multiagent system. Hence, agents should be capable to generate their own protocols when needed. Accordingly, in this chapter we present two algorithms that can be used by an agent to generate commitment protocols on demand. These algorithms can be utilized by a leading agent to realize the first phase of the general process that we presented in Chapter 1. Both algorithms generate commitment protocols that allow the leading agent to reach its goals, once executed. For this purpose the algorithms consider the leading agent's capabilities and other agents' services while generating commitment protocols. However, the algorithms do not only consider the leading agent's goals, but also take into account the other agents' interests that are expected to provide their services in the generated protocols, in order to create incentive and persuade them to participate. The main difference between the two algorithms is the strategy that they use to generate commitment protocols. While our first algorithm uses a depth-first traversal strategy to generate commitment protocols, our second algorithm utilizes the divide and conquer strategy for the same purpose. In the rest of this chapter we first define a support relation over goals, which is the basis of our algorithms. Then we define our algorithms, and also show their formal properties and compare their performance via computational experiments.

3.1. Supporting Goals

Agents in a multiagent system achieve their goals utilizing their own capabilities and other agents' services. However, while using other agents' services, agents need a commitment from the service provider agent about the provision of the service, in order to regulate their interaction. Accordingly, in order to effectively achieve their goals, agents should be able to decide whether they can do this in the context of their capabilities and existing commitments. In order to formalize this requirement, we define the concept of goal support. Primarily, we say that an agent *supports* a goal, if the agent can achieve the goal using its own capabilities or there is a commitment from another agent to provide a service that causes the agent to achieve its goal. We formalize goal support in the context of our framework using the following recursive definition.

Definition 3.1. *(Goal Support) Given a multiagent system* $\langle \mathcal{X}, \Pi \rangle$*, an agent* $x = \langle \mathcal{G}, \mathcal{F}, \mathcal{B} \rangle$*, such that* $x \in \mathcal{X}$ *and a goal* $g = G_x(u)$ *such that* $g \in \mathcal{G}$*,* x supports $G_x(u)$ *with respect to a commitment protocol* π *such that* $\pi \in \Pi$*, denoted as* $x, \pi \Vdash G_x(u)$*, if one of the following holds.*

- $x, \pi \Vdash G_x(\top)$
- $x, \pi \Vdash G_x(u)$ *iff* $F_x(\theta, u) \in \mathcal{F}$ *and*
  $\forall q \in \theta,$ *it is the case that* $x, \pi \Vdash G_x(q)$
- $x, \pi \Vdash G_x(u)$ *iff* $C(y, x, \theta \cup \{w\}, u) \in \pi$ *and*
  $S_x(y, \theta, u) \in \mathcal{B}$ *and*
  $I_x(y, w, u) \in \mathcal{B}$ *and*
  $\forall q \in \theta,$ *it is the case that* $x, \pi \Vdash G_x(q)$ *and*
  $x, \pi \Vdash G_x(w)$

The first item is the base case that deals with the trivially true conditions. The second item defines the case in which x can satisfy u (i.e., achieve its goal $G_x(u)$) using one of its own capabilities. That is, if x has the necessary capability to realize u, it can support $G_x(u)$. More specifically, x supports $G_x(u)$, if $F_x(\theta, u)$ is in $\mathcal{F}$. Furthermore, in order to be able to use $F_x(\theta, u)$, x has to support θ, which is the precondition of the capability. Technically, θ is a set of conditions. Hence, x should consider every condition q in θ as a separate goal and has to support each of them. The third item defines the case in which x needs provision of a service by another agent y in order to achieve its goal $G_x(u)$. In this case, x should be the creditor of y's commitment about u (more precisely $C(y, x, \theta \cup \{w\}, u)$). Furthermore, x should believe that y

provides an appropriate service to bring about u (i.e., $S_x(y, \theta, u') \in \mathcal{B}$) and there is an incentive that can be used to persuade y to provide the particular service (i.e., $I_x(y, w, u) \in \mathcal{B}$). Finally, x should support the precondition θ of y's service and the corresponding incentive w. Note that the commitment's antecedent involves not only the incentive w but also the precondition θ of y's service. This is necessary to clarify that x is responsible for satisfying the precondition of the provided service.

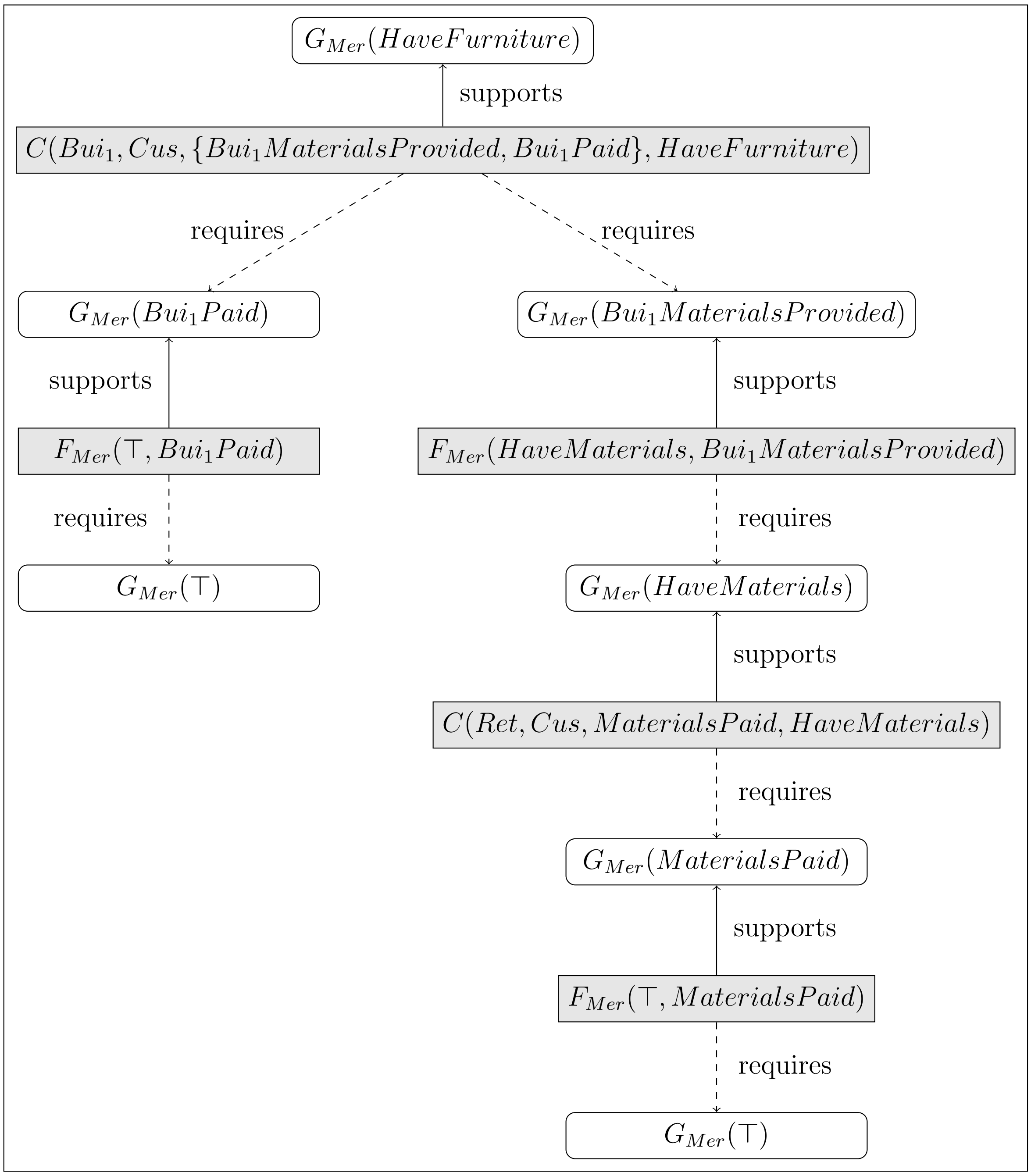
$G_{Mer}(HaveFurniture)$
supports
$C(Bui_1, Cus, \{Bui_1MaterialsProvided, Bui_1Paid\}, HaveFurniture)$
requires
requires
$G_{Mer}(Bui_1Paid)$
$G_{Mer}(Bui_1MaterialsProvided)$
supports
supports
$F_{Mer}(\top, Bui_1Paid)$
$F_{Mer}(HaveMaterials, Bui_1MaterialsProvided)$
requires
requires
$G_{Mer}(\top)$
$G_{Mer}(HaveMaterials)$
supports
$C(Ret, Cus, MaterialsPaid, HaveMaterials)$
requires
$G_{Mer}(MaterialsPaid)$
supports
$F_{Mer}(\top, MaterialsPaid)$
requires
$G_{Mer}(\top)$

Figure 3.1. Support graph of the protocol π that supports $G_{Cus}(HaveFurniture)$.

To clarify the support concept, in Figure 3.1 we show a graphical representation of how an example commitment protocol $\pi = \{c_1, c_2\}$ supports the customer's goal (i.e., $G_{Mer}(HaveFurniture)$) in our running example (Section 2.3 on page 19). In Figure 3.1, white filled nodes represent the goals of the leading agent and gray filled nodes represent the commitments and the capabilities to support the goals. Edges labeled as *support* show that the commitment or the capability in the source node supports the goal in the destination node. Dashed edges labeled as *requires* show that the commitment's or capability's precondition in the source node requires the goal in the destination node to be supported. Content of the c_1 and c_2 are as follows:

- $c_1 = C(Bui_1, Cus, \{Bui_1MaterialsProvided, Bui_1Paid\}, HaveFurniture)$
- $c_2 = C(Ret, Cus, MaterialsPaid, HaveMaterials)$

π supports $G_{Mer}(HaveFurniture)$ with respect to Definition 3.1 as follows. Initially, c_1 is used to support $G_{Mer}(HaveFurniture)$, which matches to the given abstract commitment in the support definition's third case. That is the consequent of the commitment is identical to the goal's condition. Besides, the customer's beliefs include $S_{Cus}(Bui_1, Bui_1MaterialsProvided, HaveFurniture)$ (s_4) to satisfy the consequent and there is $I_{Cus}(Bui_1, Bui_1Paid, HaveFurniture)$ (v_6) to persuade the first builder. Now, it is necessary to support $G_{Mer}(Bui_1MaterialsProvided)$ and $G_{Mer}(Bui_1Paid)$ that are required because of the c_1's precondition and incentive, respectively. These two goals can be supported by the customer herself according to the second case of the support definition using the capabilities $F_{Cus}(HaveMaterials, Bui_1MaterialsProvided)$ (f_2) and $F_{Cus}(\top, Bui_1Paid)$ (f_7). f_7 has no precondition, which means it is supported with respect to the base case of the support definition. However, f_2 has the precondition $HaveMaterials$. Hence, $G_{Mer}(HaveMaterials)$ should also be supported. This is achieved using c_2, with respect to $S_{Cus}(Ret, \top, HaveMaterials)$ (s_1) and $I_{Cus}(Ret, MaterialsPaid, HaveMaterials)$ (v_1) according to the third case of the support definition. s_1 has no precondition and it is supported with respect to the base case. But, v_1 requires $MaterialsPaid$ to be supported. Indeed, it is supported by $F_{Cus}(\top, MaterialsPaid)$ (f_4) with respect to the second case of the support definition. Finally, since f_4 does not have a precondition, it is trivially supported. As a result, the initial goal $G_{Mer}(HaveFurniture)$ and all the goals that are introduced for preconditions and incentives are supported with respect to the Definition 3.1. Hence, it is the case that $Mer, \pi \Vdash G_{Mer}(HaveFurniture)$.

Now we show that Definition 3.1 is sufficient to ensure that a protocol does allow agent x to achieve the desired goal.

Lemma 3.2. *Let $x, \pi \Vdash G_x(u)$ and suppose that all commitments in π that can be fulfilled are eventually fulfilled. Then, if x satisfies its responsibilities in π, u is eventually satisfied and accordingly $G_x(u)$ is achieved.* ■

Proof. We can show this using induction. In the base case we show that if there are no preconditions for capabilities and services, then a goal is supported according to support definition. Then, using induction we can show that a goal is supported according to support definition for capabilities and services that have any number of preconditions. The complete proof is on Page 144 in Appendix B.1. □

Definition 3.1 defines support for an individual goal of an agent with respect to a commitment protocol. However, agents usually aim to achieve a set of goals. Hence, here we define support for a set of goals using the Definition 3.1 as a basis. Intuitively, a set of goals are supported by an agent with respect to a protocol, if every goal in the set is supported individually according to Definition 3.1.

Definition 3.3. *(Goal Set Support) Given an agent x, x's capabilities (i.e., $x.\mathcal{F}$), x's beliefs (i.e., $x.\mathcal{B}$), a set of goals $\mathcal{G}$, such that $\mathcal{G} \subseteq x.\mathcal{G}$ and a protocol π; x* supports *$\mathcal{G}$ with respect to π, denoted as $x, \pi \Vdash_{set} \mathcal{G}$, iff x supports every g in $\mathcal{G}$ with respect to π according to Definition 3.1. Formally:*

$$x, \pi \Vdash_{set} \mathcal{G} \quad \textit{iff} \quad \forall g \in \mathcal{G},\ \textit{it is the case that}\ x, \pi \Vdash g$$

Definitions 3.1 and 3.3 allow infinite number of possible protocols that support a given set of goals, including all supersets of a supporting protocol. That is, given any commitment protocol π such that $x, \pi \Vdash_{set} \mathcal{G}$, we also have $x, \pi' \Vdash \mathcal{G}$ that holds for any π' that is a superset of π (i.e., $\pi \subset \pi'$). Any such π' includes one or more commitments that are actually not needed to support $\mathcal{G}$. Accordingly, we define a commitment protocol that supports a given set of goals $\mathcal{G}$, which includes only the commitments that are actually needed to support $\mathcal{G}$ as a minimal supporting commitment protocol (i.e., there are no irrelevant commitments in the protocol).

Definition 3.4. *(Minimal Supporting Commitment Protocol) Given a set of goals $\mathcal{G}$ of an agent x, we define π to be a minimal commitment protocol supporting $\mathcal{G}$ for x, if it is the case that $x, \pi \Vdash_{set} \mathcal{G}$ and additionally there does not exist a (strict) subset π' of π ($\pi' \subset \pi$) such that $x, \pi' \Vdash_{set} \mathcal{G}$.*

Definition of goal support is essential for both algorithms that we present in the rest of this chapter. The algorithms generate all the minimally supporting commitment protocols for a given set of the leading agent's goals. Hence, the generated protocols allow the leading agent to achieve its goals once successfully executed (i.e., every commitment in the protocol is fulfilled).

3.2. ProtocolBased Algorithm

3.2.1. The Algorithm

We present our first algorithm that uses the depth-first traversal strategy in a recursive manner to generate commitment protocols in Figure 3.2. We call this algorithm ProtocolBased. Basically, this algorithm creates a tree structure as follows. The root of the tree contains the unsupported initial goals of the agent that runs the algorithm. One goal from this set is selected and for each capability and service that can be used to support the selected goal with respect to Definition 3.1, an edge is created with a label that shows the used capability or a commitment that ensures provision of the selected service. The destination node of each edge contains the remaining unsupported goals of the agent. If the used capability or service requires new conditions to hold (e.g., precondition of the service), then goals over these conditions are also added to the set of unsupported goals in the destination node. This process is repeated for each subsequent node until there is no unsupported goal left and a leaf node is reached. As a result, the labels (i.e., commitments) on a path from the root to a leaf node correspond to a protocol.

ProtocolBased algorithm takes three input parameters: (i) a queue of goals that are aimed to be supported ($\mathcal{G}_p$), (ii) set of goals supported so far by the algorithm ($\mathcal{G}_s$), (iii) the protocol generated so far by the algorithm (π). When the algorithm is first invoked $\mathcal{G}_p$ consists of a set of the leading agent's goals for which the leading agent aims to find support, (i.e., $\mathcal{G}_p \subseteq \mathcal{G}$). Both $\mathcal{G}_s$ and π are initially empty. The algorithm returns a set of protocols Π, such that each protocol $\pi_i \in \Pi$ supports all the goals given initially by $\mathcal{G}_p$. If it is not possible to support every goal, the algorithm returns an empty set.

In ProtocolBased algorithm we use the auxiliary functions *Dequeue*, *Enqueue* and *IsEmpty* for regular queue manipulation. *Dequeue*($\mathcal{G}$) modifies the queue $\mathcal{G}$ by removing and returning the first item in the queue. Similarly, *Enqueue*($\mathcal{G}, g$) modifies the queue $\mathcal{G}$ by adding g. *IsEmpty*($\mathcal{G}$) returns *True* if the queue $\mathcal{G}$ is empty, and *False* otherwise.

The details of ProtocolBased algorithm in Figure 3.2 are as follows. The algorithm is divided into tree major parts following the support Definition 3.1. The first part corresponds to the based case. It starts by checking whether the queue of pending goals ($\mathcal{G}_p$) is empty (line 1). If this is the case, then the algorithm returns $\{\pi\}$ (line 2) (i.e., a single protocol that consists of the commitments generated so far)[1] . Otherwise, the algorithm gets the next pending goal $g = G_x(r)$ from $\mathcal{G}_p$ (line 4).

The second part corresponds to the second case of Definition 3.1 in which the leading agent supports a goal using its own capabilities. Accordingly, in order to

[1]Recall that a set containing an empty set, i.e. $\{\emptyset\}$, is distinct from the empty set $\emptyset$, for instance $\{a\} \cup \emptyset = \{a\}$ but $\{a\} \cup \{\emptyset\} = \{a, \emptyset\}$

```
Input: $\mathcal{G}_p$, the queue of goals pending for support
Input: $\mathcal{G}_s$, the set of already supported goals
Input: $\pi$, protocol generated so far
 1: if IsEmpty($\mathcal{G}_p$) then
 2:    return $\{\pi\}$
 3: else
 4:    $g \leftarrow$ Dequeue($\mathcal{G}_p$)        /* $g = G_x(u)$ */
 5:    $\Pi \leftarrow \emptyset$
 6:    for all $\{F_x(\theta, u) : F_x(\theta, u) \in \mathcal{F}\}$ do
 7:       $\mathcal{G}'_p \leftarrow \mathcal{G}_p$
 8:       for all $q \in \theta$ do
 9:          if $G_x(q) \notin \{\mathcal{G}_s \cup \mathcal{G}_p\}$ and $q \not\equiv u$ then
10:             Enqueue($\mathcal{G}'_p, G_x(q)$)
11:          end if
12:       end for
13:       $\mathcal{G}'_s \leftarrow \mathcal{G}_s \cup \{g\}$
14:       $\Pi \leftarrow \Pi \cup$ ProtocolBased($\mathcal{G}'_p, \mathcal{G}'_s, \pi$)
15:    end for
16:    for all $\{S_x(y, \theta, u) : S_x(y, \theta, u) \in \mathcal{B}\}$ do
17:       for all $\{I_x(y, w, u) : I_x(y, w, u) \in \mathcal{B}\}$ do
18:          $\mathcal{G}'_p \leftarrow \mathcal{G}_p$
19:          for all $q \in \theta$ do
20:             if $G_x(q) \notin \{\mathcal{G}_s \cup \mathcal{G}_p\}$ and $q \not\equiv u$ then
21:                Enqueue($\mathcal{G}'_p, G_x(q)$)
22:             end if
23:          end for
24:          if $G_x(w) \notin \{\mathcal{G}_s \cup \mathcal{G}_p\}$ and $w \not\equiv u$ then
25:             Enqueue($\mathcal{G}'_p, G_x(w)$)
26:          end if
27:          Enqueue($\mathcal{G}'_s, g$)
28:          $\pi' \leftarrow \pi \cup \{C(y, x, \theta \cup \{w\}, u)\}$
29:          $\Pi \leftarrow \Pi \cup$ ProtocolBased($\mathcal{G}'_p, \mathcal{G}'_s, \pi'$)
30:       end for
31:    end for
32:    return $\Pi$
33: end if
```

Figure 3.2. ProtocolBased algorithm.

find support for g the algorithm considers the leading agent's capabilities in $\mathcal{F}$. For each matching capability $F_x(\theta, u)$ (line 6), the algorithm performs the following steps. First, for each condition q in θ the algorithm generates a new goal $G_x(q)$ and adds this new goal into the queue of pending goals $\mathcal{G}'_p$, if it is not already considered. More specifically, if $G_x(q)$ is neither in $\mathcal{G}_p$ or in $\mathcal{G}_s$ and q is not equivalent to u, then $G_x(q)$ is added to $\mathcal{G}'_p$ (line 8-12). In this way the algorithm ensures to find support (if possible) for the precondition of the capability in the future recursive calls. Note that these new goals are added into a temporary queue $\mathcal{G}_p'$, instead of the original queue $\mathcal{G}_p$. This is necessary in order to keep track of distinct modifications for each alternative protocol the algorithm generates. This situation applies also for $\mathcal{G}_s'$ and Π' in the rest of the algorithm. After the capability's precondition is considered, the algorithm adds g into the set of supported goals $\mathcal{G}_s'$ (line 13). Finally, the algorithm recursively invokes itself by supplying the updated data structures $\mathcal{G}_p'$, $\mathcal{G}_s'$ and Δ in order to find support for the next goal in $\mathcal{G}_p'$ (line 14). These steps are performed for each matching capability and accordingly an alternative protocol is generated for each such capability.

The last part corresponds to the third case of Definition 3.1 in which the leading agent supports a goal utilizing other agents' services via commitments. Accordingly, the algorithm considers other agents' services in $\mathcal{B}$ to find alternative ways of supporting g. For each such service the algorithm also considers the incentives that can be offered to the service provider for the provision of the services. Accordingly, for each matching service $S_x(y, \theta, u)$ (line 16) and for each incentive $I_x(y, w, u)$ (line 17), the algorithm performs the following steps. First, for each condition q in θ the algorithm generates a new goal $G_x(q)$ and adds this new goal into $\mathcal{G}_p'$, if the goal is not already considered (lines 19-23). In the same manner, the algorithm generates a new goal $G_x(w)$ also for the selected incentive and adds it to $\mathcal{G}_p'$, if it is not already considered (line 24-26). After that, the algorithm adds g into the set of supported goals $\mathcal{G}_s'$ (line 27). Then, the algorithm generates a new commitment $C(y, x, \theta \cup \{w\}, u)$ and adds it to the protocol π' (line 28). After that, the algorithm recursively invokes itself by supplying the updated data structures $\mathcal{G}_p'$, $\mathcal{G}_s'$ and π' in order to find support for the next pending goal in $\mathcal{G}_p'$ (line 29). In this way, the algorithm generates an alternative protocol supporting g for each service and incentive pair. Finally, once every matching capability and service is considered as explained above, the algorithm returns the generated set of protocols Π (line 32).

Before moving to the formal properties of ProtocolBased algorithm, we explain how the algorithm behaves when a goal is not supported (i.e., there is neither a capability nor a service to achieve the goal). If this is the case, the algorithm should not generate a protocol, since we aim to generate protocols that support every goal of the leading agent (either initially given or created by the algorithm to support a precondition). Indeed, the algorithm does not execute the loops in line 6 and line 16, if g is not supported, since there is no matching capability and service, respectively. Hence, it returns Π directly as it is initiated to $\emptyset$ (line 5). If this happens at an intermediate node, then an empty set is returned to the parent node, which has made the recursive call, and all the sub-tree (and the corresponding protocols) that has the child node as a root is automatically discarded. Hence, if there is no protocol that supports every goal, all recursive calls return empty set to the root.

3.2.2. Formal Properties of ProtocolBased Algorithm

In this section we show the formal properties of the ProtocolBased Algorithm. For readability here we present only the proof sketches of the properties. The complete proof of each property is available in Appendix B.1.

We begin with soundness of ProtocolBased algorithm. Basically, ProtocolBased algorithm is sound, if every protocol that it generates contains sufficient commitments to support all of the goals given as input.

Theorem 3.5. *(Soundness) Given an agent x, x's capabilities (i.e., $x.\mathcal{F}$), x's beliefs (i.e., $x.\mathcal{B}$) and a set of goals $\mathcal{G}$, such that $\mathcal{G} \subseteq x.\mathcal{G}$; if a protocol π is generated by* ProtocolBased *algorithm (i.e., $\pi \in \Pi$), then x supports $\mathcal{G}$ with respect to π. Hence,* ProtocolBased *algorithm is sound. Formally: Let* $\Pi = \text{ProtocolBased}(\mathcal{G}, \emptyset, \emptyset)$, *then* $\forall \pi \in \Pi$ *it is the case that* $x, \pi \Vdash_{set} \mathcal{G}$. ■

Proof Sketch: We can prove ProtocolBased algorithm's soundness showing the correspondence between the algorithm and Definitions 3.1 and 3.3. That is, Lemma 3.2 guarantees sufficiency of Definition 3.1 to support a goal and it is trivial to extend this situation to a set of goals using Definition 3.3. The complete proof is on Page 144 in Appendix B.1. □

Now we show that every protocol that is generated by the ProtocolBased algorithm is a minimal supporting commitment protocol.

Lemma 3.6. *(Minimality) Given an agent x, x's capabilities (i.e., $x.\mathcal{F}$), x's beliefs (i.e., $x.\mathcal{B}$) and a set of goals $\mathcal{G}$, such that $\mathcal{G} \subseteq x.\mathcal{G}$; let π be any protocol generated by* ProtocolBased *algorithm. Then π is a minimal commitment protocol supporting $\mathcal{G}$ for x with respect to Definition 3.4.* ■

Proof Sketch: Since ProtocolBased algorithm generates a commitment only to support the goals in $\mathcal{G}_p$, we can show that the algorithm generates only minimal protocols by considering how new goals are added into $\mathcal{G}_p$. That is, by showing that the algorithm adds a goal into $\mathcal{G}_p$ only if it is actually required for support, we can prove that the algorithm generates only minimal protocols. The complete proof is on Page 145 in Appendix B.1. □

We now turn to completeness. We show completeness of ProtocolBased algorithm relative to minimal supporting commitment protocols. The intuition is that ProtocolBased algorithm is complete, if it returns all the minimal supporting commitment protocols for a set of goals. Below, we use $\Pi^{min}_{x,\mathcal{G}}$ to refer to the set of all minimal protocols that support a given set of goals $\mathcal{G}$ for an agent x.

Theorem 3.7. *(Completeness) Given an agent x, x's capabilities (i.e., $x.\mathcal{F}$), x's beliefs (i.e., $x.\mathcal{B}$) and a set of goals $\mathcal{G}$, such that $\mathcal{G} \subseteq x.\mathcal{G}$; let $\Pi^{min}_{x,\mathcal{G}}$ is the set of all minimal protocols that support $\mathcal{G}$ for x with respect to Definition 3.4 and Π is the set of protocols that is generated by* ProtocolBased *algorithm. Then Π is equal to $\Pi^{min}_{x,\mathcal{G}}$. Hence,* ProtocolBased *algorithm is complete. Formally: Let $\Pi^{min}_{x,\mathcal{G}}$ be the set of all minimal protocols for x to support $\mathcal{G}$ and $\Pi =$* ProtocolBased$(\mathcal{G}, \emptyset, \emptyset)$*, then $\Pi = \Pi^{min}_{x,\mathcal{G}}$.* ■

Proof Sketch: Because of Lemma 3.6 we know that ProtocolBased algorithm generates only minimal protocols with respect to Definitions 3.4. Hence, we should only show that ProtocolBased algorithm generates every protocol in $\Pi^{min}_{x,\mathcal{G}}$. For this purpose we can use induction, where we first show that ProtocolBased algorithm generates all the minimal protocol for a single goal. Then using this result and induction we can show that ProtocolBased algorithm generates all the minimal protocols for any given number of goals. The complete proof is on Page 146 in Appendix B.1. □

Finally, we show that ProtocolBased algorithm always terminates.

Theorem 3.8. *(Termination) Given an agent x, x's capabilities (i.e., $x.\mathcal{F}$), x's beliefs (i.e., $x.\mathcal{B}$) and a set of goals $\mathcal{G}$, such that $\mathcal{G} \subseteq x.\mathcal{G}$; if $\mathcal{G}$, $x.\mathcal{F}$ (i.e., number of capabilities and number of preconditions for each capability) and $x.\mathcal{B}$ (i.e., number of services, number of preconditions for each service, and number of incentives) are finite, then* ProtocolBased *algorithm terminates.* ■

Proof Sketch: ProtocolBased algorithm does not terminate if one of the iterators over capabilities, services or incentives does not terminate. However, this is not valid for ProtocolBased algorithm, since these sets are constant and finite, and accordingly iterators terminate eventually. Besides, each goal is considered only once in a protocol. Hence, cycles are not possible. Accordingly, ProtocolBased algorithm terminates. The complete proof is on Page 146 in Appendix B.1. □

3.3. GoalBased Algorithm

In the previous section we have defined ProtocolBased algorithm that uses the depth-first traversal strategy to generate commitment protocols. Although ProtocolBased algorithm corresponds fairly directly to Definition 3.1, it is inefficient mainly because, if a goal is included in different protocols, it finds support for that goal redundantly for each different protocol. To overcome this drawback, in this section we present a more efficient algorithm using divide and conquer strategy and memoization. This algorithm considers each goal of the agent separately by generating a sub-protocol for each individual goal. Then these sub-protocols are merged to come up with a complete protocol. This modular approach allows us to use memoization to reuse previously generated sub-protocols, if the same goal is considered in different protocols. Hence, the algorithm finds support for each goal only once and avoids redundant computation.

3.3.1. The Algorithm

We present our algorithm in Algorithm 3.3. We call it GoalBased. The algorithm uses the divide and conquer strategy to generate all the protocols that support the goals given as queue. More specifically, if the goal queue includes more than one goal, it is divided into two queues. The first queue contains only the first goal in the original queue and the second queue contains the rest of the goals in the original queue. The algorithm first generates the protocols that support the goal in the first queue and then continues for the goals in the second queue applying the same divide and conquer strategy. Finally, the algorithm merges the generated protocols for each goal to come up with a set of protocols. For efficiency, the algorithm uses memoization utilizing a mapping that keeps the generated protocols for each goal and reuse them, if a goal is encountered more than once during the execution.

GoalBased algorithm takes two input parameters: (i) a queue of goals that are aimed to be supported ($\mathcal{G}_p$), (ii) a mapping from goals to a set of protocols that support them ($\mathcal{M}$). When the algorithm is first invoked, $\mathcal{G}_p$ includes a subset of the leading agent's goals (i.e., $\mathcal{G}_p \subseteq \mathcal{G}$), for which the agent aims to find support. $\mathcal{M}$ is initially empty. The algorithm returns a set of protocols (Π), such that each protocol π_i in Π supports all the goals given initially by $\mathcal{G}_p$. If it is not possible to support every goal, the algorithm returns an empty set.

We use the auxiliary functions *Dequeue*, *Enqueue*, *IsEmpty* and *Size*, for regular queue manipulation. *Dequeue*($\mathcal{G}$) modifies the queue $\mathcal{G}$ by removing and returning the first item in the queue. Similarly, *Enqueue*($\mathcal{G}, g$) modifies the queue $\mathcal{G}$ by adding g. $IsEmpty(\mathcal{G})$ returns $True$ if the queue $\mathcal{G}$ is empty, and $False$ otherwise. $Size(\mathcal{G})$ returns the number of goals in $\mathcal{G}$. The auxiliary function $Merge(\Pi', \Pi'')$ is a two phase function. In the first phase, it takes two set of protocols Π' and Π'' and creates a new set of protocols Π taking cartesian product of Π' and Π'' (i.e.,

```
Input: $\mathcal{G}_p$, the queue of not supported goals
Input: $\mathcal{M}$, the map from goals to known supporting protocols
 1: if IsEmpty($\mathcal{G}_p$) then
 2:    return $\{\emptyset\}$
 3: else if Size($\mathcal{G}_p$) $> 1$ then
 4:    $\mathcal{G}' \leftarrow$ Dequeue($\mathcal{G}_p$)
 5:    return Merge(GoalBased($\mathcal{G}', \mathcal{M}$), GoalBased($\mathcal{G}_p, \mathcal{M}$))
 6: else
 7:    /* Size($\mathcal{G}_p$) $= 1$ */
 8:    $g \leftarrow$ Dequeue($\mathcal{G}_p$)        /* $g = G_x(u)$ */
 9:    if $g \in$ GetKeySet($\mathcal{M}$) then
10:       return $\mathcal{M}[g]$
11:    end if
12:    $\Pi \leftarrow \emptyset$
13:    for all $\{F_x(\theta, u) : F_x(\theta, u) \in \mathcal{F}\}$ do
14:       $\mathcal{G}' \leftarrow \emptyset$
15:       for all $q \in \theta$ do
16:          if $q \not\equiv u$ then
17:             Enqueue($\mathcal{G}', G_x(q)$)
18:          end if
19:       end for
20:       $\Pi \leftarrow \Pi \cup$ GoalBased($\mathcal{G}', \mathcal{M}$)
21:    end for
22:    for all $\{S_x(y, \theta, u) : S_x(y, \theta, u) \in \mathcal{B}\}$ do
23:       for all $\{I_x(y, w, u) : I_x(y, w, u) \in \mathcal{B}\}$ do
24:          $\mathcal{G}' \leftarrow \emptyset$
25:          for all $q \in \theta$ do
26:             if $q \not\equiv u$ then
27:                Enqueue($\mathcal{G}', G_x(q)$)
28:             end if
29:          end for
30:          if $w \not\equiv u$ then
31:             Enqueue($\mathcal{G}', G_x(w)$)
32:          end if
33:          $\Pi' \leftarrow$ GoalBased($\mathcal{G}', \mathcal{M}$)
34:          for all $\pi \in \Pi'$ do
35:             $\pi \leftarrow \pi \cup C(y, x, \theta \cup \{w\}, u)$
36:          end for
37:          $\Pi \leftarrow \Pi \cup \Pi'$
38:       end for
39:    end for
40:    $\mathcal{M}[g] \leftarrow \Pi$
41:    return $\Pi$
42: end if
```

Figure 3.3. GoalBased algorithm.

$\Pi = \{(\pi', \pi'') \mid \pi' \in \Pi' \text{ and } \pi'' \in \Pi''\}$. In the second phase, the *Merge* function filters the non-minimal protocols in Π by discarding the protocols that include two or more commitments, which share a common consequent condition (Definition 3.4). For example, if a protocol includes the commitments $C(y, x, \theta, u)$ and $C(y', x, \theta', u)$, then this protocol is discarded. Accordingly, the resulting Π, which is returned by *Merge*, includes only minimal supporting protocols. $\mathcal{M} : g \to \Pi$ is a mapping from a goal g to a set of protocols Π that support g. The function *GetKeySet*($\mathcal{M}$) returns the key set of the mapping $\mathcal{M}$. We use notation $\mathcal{M}[g]$ to access the set of protocols that are associated with the key g.

The details of GoalBased algorithm in Figure 3.3 are as follows. It starts by checking whether the goal queue ($\mathcal{G}_p$) is empty (line 1). If this is the case, then the algorithm returns an empty protocol to indicate that the base case is reached (line 2). Otherwise, if $\mathcal{G}_p$ includes more than one goal, the algorithm divides the queue into two queues such that the first queue includes only the first goal in $\mathcal{G}_p$ and the second queue includes the rest of the goals in $\mathcal{G}_p$. Then, for each queue a recursive call is made and the resulting set of commitment protocols are merged (taking cartesian product of the sets) and returned (lines 3-6).

If $\mathcal{G}_p$ includes only one goal (i.e., $g = G_x(u)$), the algorithm first checks $\mathcal{M}$ to find out whether the protocols that support g are already generated. If this is the case, the algorithm immediately returns the corresponding protocols (lines 9-11). Otherwise, the algorithm checks x's capabilities in $\mathcal{F}$ to find matching capabilities that can be utilized to support g. For every such capability $F_x(\theta, u)$, the algorithm first creates a queue $\mathcal{G}'$ and adds a new goal $G_x(q)$ for each condition q in the capability's precondition θ into $\mathcal{G}'$ (lines 15-19). Then the algorithm calls itself to find support for these recently created goals in $\mathcal{G}'$ and adds the returned protocols to the set of generated protocols Π (line 20). This procedure is performed for each matching capability and accordingly an alternative protocol is generated for each such capability.

After considering x's capabilities, the algorithm considers other agents' services in $\mathcal{B}$ to find alternative ways of supporting g. For each such service the algorithm also considers the incentives that can be offered to the service provider for the provision of the services. Accordingly, for each matching service $S_x(y, \theta, u)$ (line 22) and for each incentive $I_x(y, w, u)$ (line 23), the algorithm performs the following steps. First, for each condition q in θ the algorithm generates a new goal $G_x(q)$ and adds this new goal into $\mathcal{G}'$ (lines 25-29). In the same manner, the algorithm generates a new goal $G_x(w)$ also for the selected incentive and adds it to $\mathcal{G}'$ (lines 30-32). After that, the algorithm calls itself to find support for these recently created goals (line 33). Then the algorithm adds the commitment $C(y, x, \theta \cup \{w\}, u)$ to each returned protocol in Π' (lines 34-36) and adds these updated protocols into Π. Once all the capabilities of x and other agents' services are considered, and the supporting protocols are generated, the algorithm adds an entry to $\mathcal{M}$ for g and maps it to Π (line 40). Finally the algorithm returns Π (line 41).

Before moving to the formal properties of ProtocolBased algorithm, we explain how the algorithm behaves when a goal is not supported. When a goal is not supported (i.e., there is neither a capability nor a service to achieve the goal) the algorithm should not generate any protocol. Indeed, the algorithm does not execute the loops in line 13 and line 22, if g is not supported, since there is no matching capability and service, respectively. Hence, it returns Π directly as it is initiated to $\emptyset$ (line 12). Remember that if one of the sets in cartesian product is an empty set, then the result of the cartesian product is also an empty set. Accordingly, all the merge operations in the previous recursive calls produce an empty set, which is also returned as the final result of the algorithm's first call. Note that, if a goal is not supported, then an empty set is added to the mapping M for that goal. Hence, the algorithm returns empty set in line 10 as it should do.

3.3.2. Formal Properties of GoalBased Algorithm

We begin with soundness. GoalBased algorithm is sound if every protocol that it generates contains sufficient commitments to support all of the goals given as input.

Theorem 3.9. *(Soundness) Given an agent x, x's capabilities (i.e., $x.\mathcal{F}$), x's beliefs* *(i.e., $x.\mathcal{B}$) and a set of goals $\mathcal{G}$, such that $\mathcal{G} \subseteq x.\mathcal{G}$; if a protocol π is generated by* GoalBased *algorithm (i.e., $\pi \in \Pi$), then x supports $\mathcal{G}$ with respect to π. Hence,* GoalBased *algorithm is sound. Formally: Let* $\Pi = \text{GoalBased}(\mathcal{G}, \emptyset, \emptyset)$, *then* $\forall \pi \in \Pi$ *it is the case that* $x, \pi \Vdash_{set} \mathcal{G}$. ■

Proof Sketch: We can prove ProtocolBased algorithm's soundness showing the correspondence between the algorithm and Definitions 3.1 and 3.3. That is, Lemma 3.2 guarantees sufficiency of Definition 3.1 to support a goal and it is trivial to extend this situation to a set of goals using Definition 3.3. The complete proof is on Page 148 in Appendix B.1. □

Now we show that every protocol that is generated by the GoalBased algorithm is a minimal supporting commitment protocol.

Lemma 3.10. *(Minimality) Given an agent x, x's capabilities (i.e., $x.\mathcal{F}$), x's beliefs (i.e., $x.\mathcal{B}$) and a set of goals $\mathcal{G}$, such that $\mathcal{G} \subseteq x.\mathcal{G}$; let π be any protocol generated by* GoalBased *algorithm. Then π is a minimal commitment protocol supporting $\mathcal{G}$ for x with respect to Definition 3.4.* ■

Proof Sketch: Since ProtocolBased algorithm generates a commitment only to support the goals in $\mathcal{G}_p$, we can show that the algorithm generates only minimal protocols by considering how new goals are added into $\mathcal{G}_p$. That is, by showing that the algorithm adds a goal into $\mathcal{G}_p$ only if it is actually required for support, we can prove that the algorithm generates only minimal protocols. The complete proof is on Page 149 in Appendix B.1. □

We show completeness of GoalBased algorithm relative to minimal supporting commitment protocols. The intuition is that GoalBased algorithm is complete, if it returns all the minimal supporting commitment protocols for a set of goals. Below, we use $\Pi^{min}_{x,\mathcal{G}}$ to refer to the set of all minimal protocols that support a given set of goals $\mathcal{G}$ for an agent x.

Theorem 3.11. *(Completeness) Given an agent x, x's capabilities (i.e., $x.\mathcal{F}$), x's beliefs (i.e., $x.\mathcal{B}$) and a set of goals $\mathcal{G}$, such that $\mathcal{G} \subseteq x.\mathcal{G}$; let $\Pi^{min}_{x,\mathcal{G}}$ is the set of all minimal protocols that support $\mathcal{G}$ for x with respect to Definition 3.4 and Π is the set of protocols that is generated by* GoalBased *algorithm. Then Π is equal to $\Pi^{min}_{x,\mathcal{G}}$. Hence,* GoalBased *algorithm is complete. Formally: Let $\Pi^{min}_{x,\mathcal{G}}$ be the set of all minimal protocols for x to support $\mathcal{G}$ and $\Pi = \textsc{GoalBased}(\mathcal{G}, \emptyset, \emptyset)$, then $\Pi = \Pi^{min}_{x,\mathcal{G}}$.* ■

Proof Sketch: Because of Lemma 3.10 we know that GoalBased algorithm generates only minimal protocols with respect to Definitions 3.4. Hence, we should only show that GoalBased algorithm generates every protocol in $\Pi^{min}_{x,\mathcal{G}}$. For this purpose we can use induction, where we first show that GoalBased algorithm generates all the minimal protocol for a single goal. Then using this result and induction we can show that GoalBased algorithm generates all the minimal protocols for any given number of goals. The complete proof is on Page 150 in Appendix B.1. □

Theorem 3.12. *(Termination) Given an agent x, x's capabilities (i.e., $x.\mathcal{F}$), x's beliefs (i.e., $x.\mathcal{B}$) and a set of goals $\mathcal{G}$, such that $\mathcal{G} \subseteq x.\mathcal{G}$; if $\mathcal{G}$, $x.\mathcal{F}$ (i.e., number of capabilities and number of preconditions for each capability) and $x.\mathcal{B}$ (i.e., number of services, number of preconditions for each service, and number of incentives) are finite, then* GoalBased *algorithm terminates.* ■

Proof Sketch: GoalBased algorithm does not terminate if one of the iterators over capabilities, services or incentives do not terminate. However, this is not valid for GoalBased algorithm, since these sets are constant and finite, and accordingly iterators terminate eventually. Besides, each goal is considered only once in a protocol. Hence, cycles are not possible. Accordingly, GoalBased algorithm terminates. The complete proof is on Page 150 in Appendix B.1. □

3.4. Comparison of Algorithms and Experimental Results

In this section we compare our two algorithms. First we use our running example to compare the execution strategies of the algorithms. Then, we provide the results of computational experiments in which we compare execution performance of the algorithms over a set of systematically generated data set.

3.4.1. Execution Strategies and Complexity of the Algorithms

The ProtocolBased and GoalBased algorithms generate the same protocols. We present the protocols that are generated by these algorithms for our running example in Table 3.1. Protocol π_1 includes a single commitment from the merchant to the customer in order to support the customer's goal $HaveFurniture$. However, the customer should bring about $FurniturePaid$ to make this commitment active. In other words, the customer has to support $FurniturePaid$. The customer supports this condition by its capability f_6. Hence, there is no need for another commitment in this protocol. In π_2 and π_3, the customer's goal is supported by the commitment of the first builder to the customer to build the furniture. However, the customer has to support $Bui_1MaterialsProvided$ in order to make this commitment active. But the customer is not capable to do that, since she does not have the necessary materials. Accordingly, π_2 and π_3 include a commitment from the retailer to the customer for provision of the materials, with alternative incentives $MaterialsPaid$ and $ToolsPaid$. Finally, protocols $\pi_4 - \pi_7$ include a commitment from the second builder to the customer to support the customer's goal. However, the customer has to support $ToolsProvided$ as well as providing materials to the second builder to make this commitment active. Since the customer is not capable to provide either of them, these protocols include two commitments from the retailer to the customer, one to support provision of the materials and one to support provision of the tools. Each of these protocols uses different incentive in their commitments to stimulate the retailer to provide the materials and tools.

Although both ProtocolBased and GoalBased algorithms generate the same protocols for a given input, there are significant differences in their strategies to gen-

Table 3.1. Generated protocols for the running example.

π_1	$c_3 = C(Mer, Cus, FurniturePaid, HaveFurniture)$
π_2	$c_4 = C(Ret, Cus, MaterialsPaid, HaveMaterials)$ $c_1 = C(Bui_1, Cus, \{Bui_1MaterialsProvided, Bui_1Paid\}, HaveFurniture)$
π_3	$c_5 = C(Ret, Cus, ToolsPaid, HaveMaterials)$ $c_1 = C(Bui_1, Cus, \{Bui_1MaterialsProvided, Bui_1Paid\}, HaveFurniture)$
π_4	$c_4 = C(Ret, Cus, MaterialsPaid, HaveMaterials)$ $c_6 = C(Ret, Cus, ToolsPaid, HaveTools)$ $c_2 = C(Bui_2, Cus, \{Bui_2MaterialsProvided, ToolsProvided, Bui_2Paid\},$ $HaveFurniture)$
π_5	$c_5 = C(Ret, Cus, ToolsPaid, HaveMaterials)$ $c_6 = C(Ret, Cus, ToolsPaid, HaveTools)$ $c_2 = C(Bui_2, Cus, \{Bui_2MaterialsProvided, ToolsProvided, Bui_2Paid\},$ $HaveFurniture)$
π_6	$c_4 = C(Ret, Cus, MaterialsPaid, HaveMaterials)$ $c_7 = C(Ret, Cus, MaterialsPaid, HaveTools)$ $c_2 = C(Bui_2, Cus, \{Bui_2MaterialsProvided, ToolsProvided, Bui_2Paid\},$ $HaveFurniture)$
π_7	$c_5 = C(Ret, Cus, ToolsPaid, HaveMaterials)$ $c_7 = C(Ret, Cus, MaterialsPaid, HaveTools)$ $c_2 = C(Bui_2, Cus, \{Bui_2MaterialsProvided, ToolsProvided, Bui_2Paid\},$ $HaveFurniture)$

erate the protocols. To show these differences we provide execution traces of ProtocolBased and GoalBased algorithms for our running example in Figures 3.4 and 3.5. In the figures we use identifiers of conditions, capabilities and beliefs as listed in Section 2.3 for readability. In Figure 3.4, each node shows the goals in the goal queues of the ProtocolBased and GoalBased algorithms at each recursive call. For readability we show only the conditions of the goals and omit the full goal syntax (i.e., we write down u instead of $G_x(u)$). The edge labels show how the goal that is listed first in the source node (g in the algorithm) is supported. If it is supported

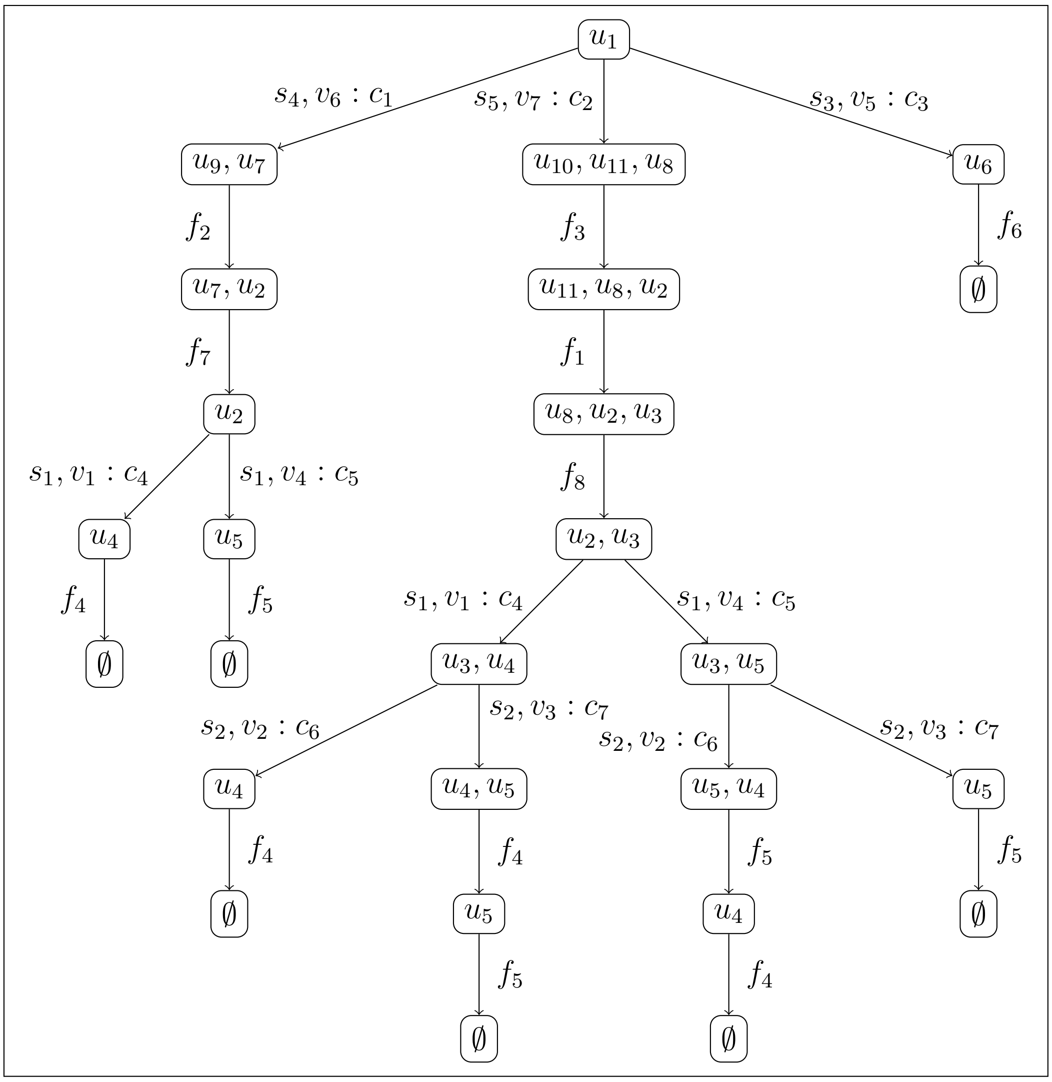

Figure 3.4. Execution trace of ProtocolBased algorithm on the running example.

using a capability then the label of the edge is the identifier of the capability. If it is supported by a service, which causes creation of a commitment, then the label syntax is $s, v : c$ in which s is the supporting service, v is the incentive and c is the generated commitment with respect to s and v (i.e., c's antecedent includes the preconditions of s and the condition of v). The identifiers of the commitments are available in Table 3.1. In the trace of GoalBased algorithm (Figure 3.5) there are also dashed edges between nodes, which show the division of the goal queue. There are also black filled circles, which represent the use of memoization.

Now we compare the algorithms' executions using these traces. ProtocolBased algorithm uses a depth-first traversal strategy to create the tree structure in

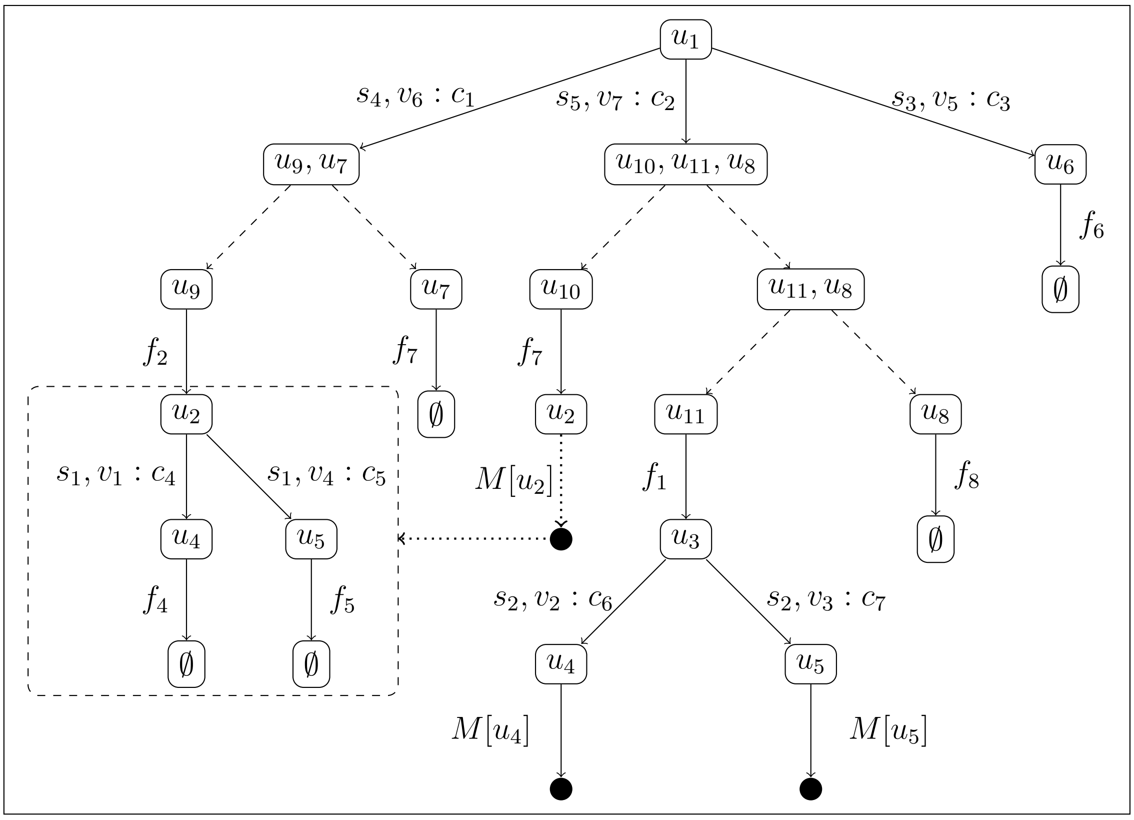

Figure 3.5. Execution trace of GoalBased algorithm on the running example.

Figures 3.4 where each path from the root to a leaf node corresponds to a complete protocol. For example the leftmost path from the leaf node to the root node covers commitments c_1 and c_4, which is the protocol π_2 in Table 3.1. ProtocolBased algorithm generates a complete protocol at a time. Hence, while searching to find support for a goal in the context of a protocol, it does not take into account the support that is already found for the same goal in the context of a previously generated protocol. For example, consider u_2 in Figures 3.4. In the left branch from the root u_2 is supported by the commitments c_4 and c_5. Moreover, in this branch u_4 and u_5, which are the incentives of these commitments, are also supported by f_4 and f_5, respectively. Now consider the middle branch from the root. There is a state that contains u_2 and u_3. As we can see from the figure, the sub-tree of this node finds exactly the same support for u_2, as we explained above for the left branch, performing redundant computations.

GoalBased algorithm uses the divide and conquer strategy to create the tree structure in Figures 3.5. Accordingly GoalBased algorithm divides its goal queue repeatedly until the queue contains only a single goal and then generates sub-protocols for that individual goal. Then these sub-protocols are merged into larger sub-protocols repetitively until a complete protocol is generated. GoalBased algorithm overcomes the redundant computations that occur in the case of ProtocolBased algorithm using memoization. Consider how u_2 is handled by GoalBased algorithm. It first finds support for u_2 in the left branch from the root using the commitments c_4 and c_5. Then it uses memoization and keep the portion of the tree that is enclosed by the dashed rectangle, in $\mathcal{M}$ as the sub-protocols that supports u_2. Then, while considering u_2 in the middle branch from the root, the algorithm just reuses this sub-protocol, instead of searching support for u_2 again. Hence, redundant computations are eliminated by GoalBased algorithm. Note that, in the case of ProtocolBased algorithm such modularity cannot be achieved effectively, since every protocol is generated as a whole, without taking the relations between goals and sub-protocols into account.

Since ProtocolBased algorithm traverses the whole tree to generate all the protocols, both the worst-case and the best-case complexities of ProtocolBased algorithm are linear to the size of the tree (i.e., $\mathcal{O}(E)$, where E is the number of edges). On the other hand the size of the tree depends on several factors, such as the average number of capabilities and services that can be utilized to support a goal, the average number of preconditions of these services and capabilities, the average number of incentives and finally the average depth at which the base case is reached by the algorithm (i.e., every pending goal can be supported by a capability that does not have a precondition). In general increasing these numbers cause to exponential growth in the size of the tree.

The worst-case of the GoalBased algorithm occurs when the precondition of each capability and service is unique. In this case, GoalBased algorithm's complexity is equal to the ProtocolBased algorithm's complexity, since no sub-protocol can be reused by GoalBased algorithm. In other words, the mapping from goals to supporting sub-protocols remains empty during the whole execution and the algorithm has to consider every edge as in the case of ProtocolBased algorithm. On the other hand, the best case of the GoalBased algorithm occurs, when every service

Table 3.2. Summary of parameters to generate the experimental data set.

Parameter	Description
γ	Number of initial goals.
σ	Number of services that can be requested from other agents to achieve a goal.
ρ	Number of preconditions required to use a service.
ε	Number of possible incentives that can be offered to the provider for each service.
λ	Controls when a goal can be achieved using a capability that has no precondition, instead of a service.

and capability to support a goal share the same precondition. In this case the Goal-Based algorithm finds a sub-protocol π only once and reuses this sub-protocol for all service and capability preconditions. Accordingly, best-case complexity of the Goal-Based algorithm is $\mathcal{O}(d)$ where d is equal to the size of θ (i.e., number of conditions in the precondition that is shared among all capabilities and services).

3.4.2. Computational Result

In order to justify the above discussion and test execution performance of our algorithms we conducted computational experiments. To the best of our knowledge there does not exist any large data set of commitment protocols for testing in the literature. Hence, we conducted our experiments on a data set we generated parametrically as we explain below.

The basic idea of this generation process is as follows: we begin with a number of top-level goals. We then repeatedly consider these goals, and for each goal we generate, (i) a set of alternative services with preconditions that can be used to bring about the goal's conditions, and (ii) a set of alternative incentives that can be used to motivate the other agent to adopt the commitment to provide its service. Once services and incentives are generated, preconditions of the generated services and incentive

Table 3.3. Example service and incentive data generated using $\gamma = 1$, $\sigma = 2$, $\rho = 1$, $\varepsilon = 1$ and $\lambda = 3$.

Level	Services	Incentives	Capabilities
1	$s_1 = S_x(y, u_2, u_1)$ $s_2 = S_x(y, u_3, u_1)$	$v_1 = I_x(y, u_4, u_1)$	none
2	$s_3 = S_x(y, u_5, u_2)$ $s_4 = S_x(y, u_6, u_2)$	$v_2 = I_x(y, u_7, u_2)$	none
	$s_5 = S_x(y, u_8, u_3)$ $s_6 = S_x(y, u_9, u_3)$	$v_3 = I_x(y, u_{10}, u_3)$	
	$s_7 = S_x(y, u_{11}, u_4)$ $s_8 = S_x(y, u_{12}, u_4)$	$v_4 = I_x(y, u_{13}, u_4)$	
3	none	none	$f_1 = F_x(\top, u_5)$ ... $f_9 = F_x(\top, u_{13})$

conditions are considered as new goals and in the next iteration we generate new services and incentives for these new goals. In other words the generation process follows the following outline:

(i) Generate initial goal(s)
(ii) For each goal that has not yet been considered: generate services and incentives.
(iii) For each generated service and incentive, create new goals to achieve the service's preconditions and the incentive.
(iv) Go to step (ii).

Now, this process clearly does not terminate, and so we modify it by only iterating a certain number of times, and then finishing the generation by creating a capability (with a true precondition) for each goal that has not yet been considered. Since these capabilities do not have preconditions, they do not generate new goals and the process terminates.

This process generates a set of services ($\mathcal{S}$), incentives ($\mathcal{I}$), and capabilities ($\mathcal{F}$)

that can be used to create support for a given number of initial goals of an agent x. In generating the services, incentives and capabilities, we use a number of parameters to control the generation process. These are summarized in Table 3.2. Most of the parameters control the number of entities generated at different points in the process (e.g. γ is the number of initial goals created in the first step above, σ is the number of services created in step two above, etc.). The exception is λ which specifies the number of iterations (i.e., if we consider the generated problem as a tree of goals, then λ specifies the depth of the tree).

Table 3.3, shows an example set of services, incentives and capabilities that are generated using our data generation algorithm using the following parameter values: $\gamma = 1$, $\sigma = 2$, $\rho = 1$, $\varepsilon = 1$ and $\lambda = 3$. While generating this example set, first we use γ to determine the number of initial goals, which is equal to one. Hence, initially we generate only one goal to achieve the condition u_1. In the first iteration we generate two services s_1 and s_2 to achieve u_1, since σ is equal to two. Since ρ is equal to one, a single precondition is generated for s_1 and s_2, namely u_2 and u_3, respectively. Finally, since $\varepsilon = 1$, a single incentive over u_4 is generated for the services that bring about u_1. As result of the generation of services s_1 and s_2, new goals for conditions u_2 and u_3, which are the preconditions of these services, are also generated. Similarly, a goal for u_4 is generated, because of v_1. In the second iteration we generate the services and incentives for u_2, u_3 and u_4 as listed in Table 3.3 using the same method as in the first iteration. As a result of the generated services and incentives in the second iteration, new goals for conditions $u_5 - u_{13}$, which are the preconditions and incentives of the generated services, that are introduced for the next iteration. However, since iteration limit λ is reached, we do not generate services for these new goals. Instead for each goal a capability that has no precondition (i.e., $f_1 - f_9$) is generated. After that the generation process terminates.

Algorithm 3.6 defines our data generation process in detail, which returns a set of services ($\mathcal{S}$), a set of incentives ($\mathcal{I}$), and a set of capabilities ($\mathcal{F}$), generated with respect to the parameters summarized in Table 3.2 using the process outlined above.

```
1: $\mathcal{S} \leftarrow \emptyset, \mathcal{I} \leftarrow \emptyset, \mathcal{F} \leftarrow \emptyset$
2: $\mathit{currentGoals} \leftarrow \{G_x(u_1), \ldots, G_x(u_\gamma)\}$
3: $i \leftarrow \gamma + 1$
4: for $l \leftarrow 1$ to $\lambda - 1$ do
5:   $\mathit{nextGoals} \leftarrow \emptyset$
6:   for all $g \in \mathit{currentGoals}$ do
7:     /* $g = G_x(u)$ */
8:     $\mathit{start} \leftarrow i$        /* remember where we started generating conditions */
9:     for $j \leftarrow 1$ to $\sigma$ do
10:      $\mathcal{S} \leftarrow \mathcal{S} \cup \{S_x(y, \{q_i, \ldots, q_{i+\rho-1}\}, u)\}$
11:      $i \leftarrow i + \rho$
12:    end for
13:    $\mathcal{I} \leftarrow \mathcal{I} \cup \{I_x(y, q_i, u), \ldots, I_x(y, q_{i+\varepsilon-1}, u)\}$
14:    $i \leftarrow i + \varepsilon$
15:    $\mathit{nextGoals} \leftarrow \mathit{nextGoals} \cup \{G_x(q_{start}), \ldots, G_x(q_{i-1})\}$
16:  end for
17:  $\mathit{currentGoals} \leftarrow \mathit{nextGoals}$
18: end for
19: for all $g \in \mathit{currentGoals}$ do
20:   /* $g = G_x(u)$ */
21:   $\mathcal{F} \leftarrow \mathcal{F} \cup \{F_x(\top, u)\}$
22: end for
23: return $\mathcal{S}, \mathcal{I}, \mathcal{F}$
```

Figure 3.6. Data generation process with parameters $\gamma, \sigma, \rho, \varepsilon, \lambda$.

The process first generates the set of initial goals for x and keeps them in *currentGoals* (Line 2). The number of initial goals is determined by the parameter γ. Then the algorithm generates the services and incentives that can be used to support the goals in *currentGoals*. For each goal, the algorithm generates a set of services provided by some agent y (Lines 6-12). The number of services is controlled by the parameter σ (Line 9). For each generated service, the algorithm generates a set of preconditions and the number of preconditions is determined by the parameter ρ (Line 10). Next, the algorithm generates a set of incentives for the current goal g where the number of incentives is controlled by the parameter ε (Line 13).

Remember that in our algorithms ProtocolBased and GoalBased, once a commitment is generated to guarantee provision of a service to support a goal in the context of a protocol, the preconditions of that service and the incentive are added to the goal queue of the algorithm in order to find support for them. Accordingly, while generating our data set, we generate supporting services and capabilities for preconditions and incentives of generated services. In order to achieve that, we generate a new goal for each precondition and incentive (Line 15). These new goals are kept in a separate set called *nextGoals*. Once we generate services and incentives for all the goals in *currentGoals*, we replace the goals in *currentGoals* with the goals in *nextGoals* (Line 17) and generate services and incentives to support these new goals in the next iteration. In this way the data generation algorithm iteratively creates services for the preconditions and incentives created in the previous iteration. In order to control the number of such iterations we use λ. Once the algorithm makes $\lambda - 1$ iterations, it stops generating new services for the preconditions and incentives of the previous iteration. Instead, the algorithm generates a capability, which has no precondition, in order to satisfy the preconditions and incentives of the previous iteration. Hence, these preconditions and incentives can be supported by the agent's capabilities without requiring any other commitment.

In the rest of this section we present results of our experiments. We implemented both ProtocolBased and GoalBased algorithms and also the data generation process we explained above in Java. We performed our experiments on an Intel i7-2620 2.7 GHz processor with 2 GB memory running Ubuntu 11.04 Linux. Each time measurement reported below is the average of ten runs.

In our first experiment we study the execution performance of our algorithms. In this study, our aim is to understand how our algorithms' performance is affected by the number of alternative protocols that we can generate to support a goal. Since, our algorithms generate an alternative protocol for each service that can be requested to support a goal, the main parameter that determines the number of alternative protocols is the number of available services to support a goal (i.e., σ). Accordingly, in order to investigate the effect of σ, we conducted an experiment where we took the initial value of σ as 1 and then increased it up to 10, while measuring the execution time of our algorithms (in milliseconds). In this experiment we fix the other parameters as $\gamma = 1$, $\rho = 2$, $\varepsilon = 2$ and $\lambda = 3$. In other words, there is initially one goal. Each service has two preconditions, hence whenever a service is used, two new goals are introduced.

Given a service, there are two alternative incentives, hence two alternative protocols are generated for each alternative service. Note that we selected these particular values for other parameters after conducting trials with different values. For larger values of these parameters, the search space (i.e., number of services and incentives) grows rapidly and it is not possible for us to observe how ProtocolBased algorithm performs, since it does not terminate in a reasonable amount of time (e.g., in ten minutes). On the other hand, for smaller values, the difference between the algorithms' performance is not adequate to make any conclusions (e.g., both algorithm terminates in a few milliseconds).

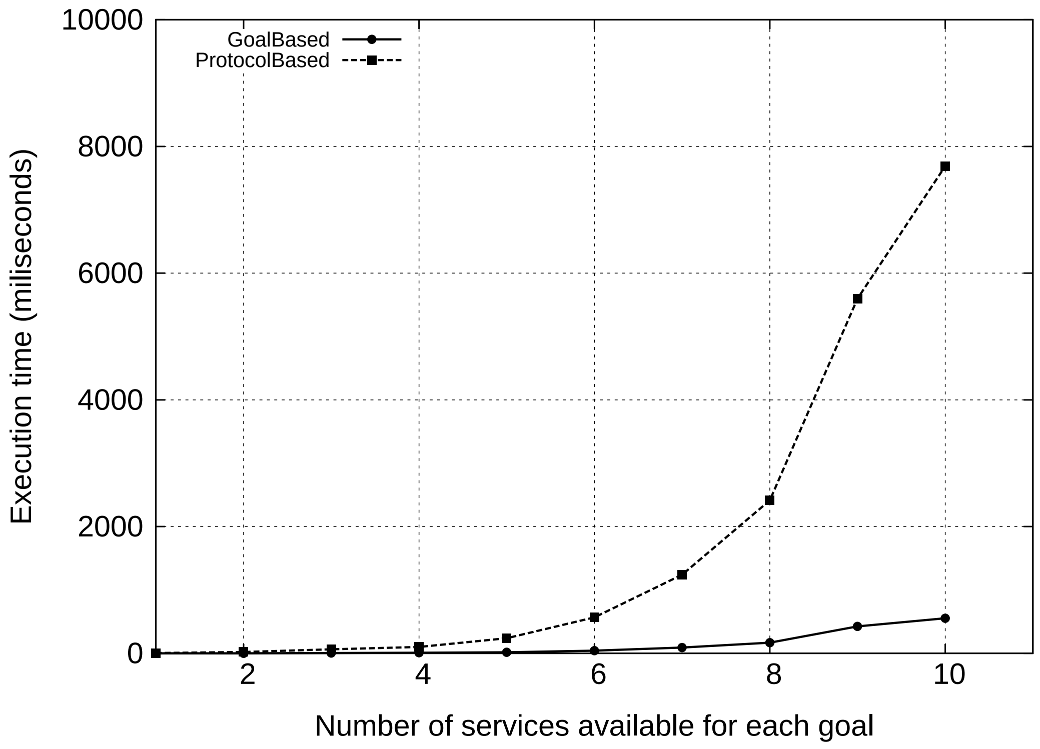

Figure 3.7. Comparison of execution times of the ProtocolBased and GoalBased algorithms based on the number of available services for each goal.

We show the results of the experiment in which we observe the effect of σ in Figure 3.7. In the figure the x-axis is the number of services that can be requested for each goal (i.e., σ) and the y-axis is the execution time in milliseconds. The execution time of ProtocolBased algorithm grows exponentially with respect to σ. On the other hand GoalBased algorithm scales well even for 10 services per goal. Note that in practice we would expect to see that a given goal typically has a relatively low number of different service types that could be used to achieve it. In other words, 10 possible service alternatives per goal is quite a high value. Besides, in practical systems, even if larger number of alternative services are available for a particular task, agents usually do not consider all these alternatives. Instead they usually select a subset depending on various criteria according to the properties of the system [26].

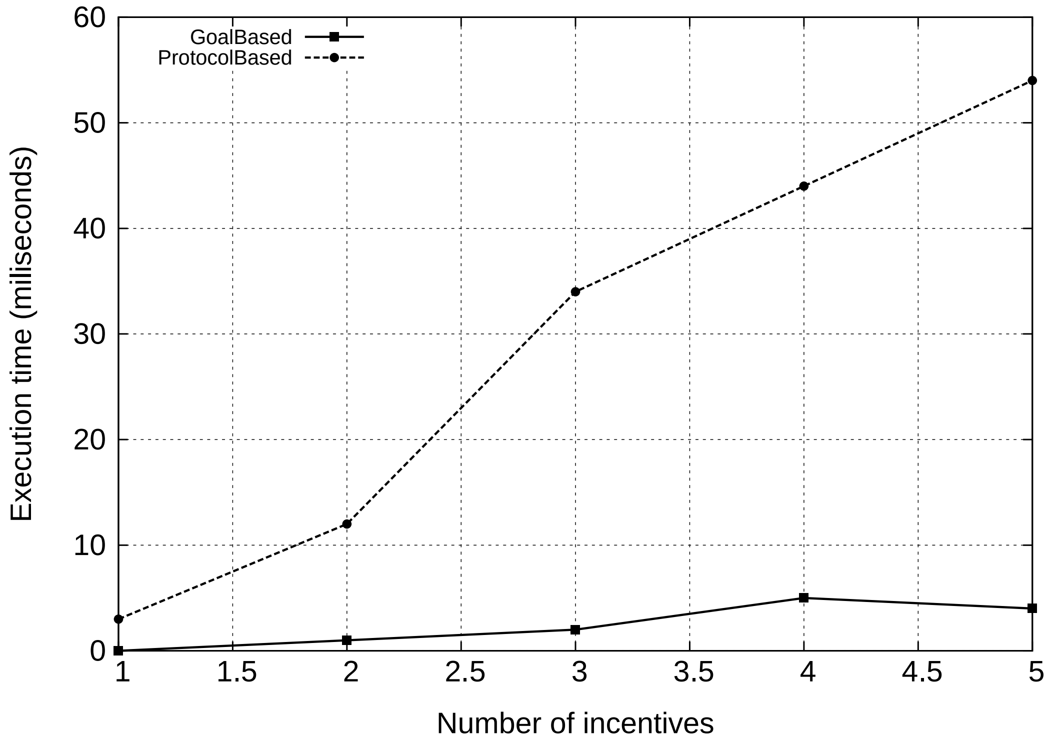

Figure 3.8. Comparison of execution times of ProtocolBased and GoalBased algorithms based on the number of preconditions ($\rho = 1$) and incentives ($\varepsilon = 1, \ldots, 5$) for each service.

By examining the execution traces of our algorithms, we conclude that the main reason for this performance difference between the algorithms is the reuse of sub-protocols by GoalBased algorithm. ProtocolBased algorithm generates the same sub-protocols many times in different protocols. On the other hand GoalBased algorithm reuses previously generated sub-protocols effectively to avoid redundant com-

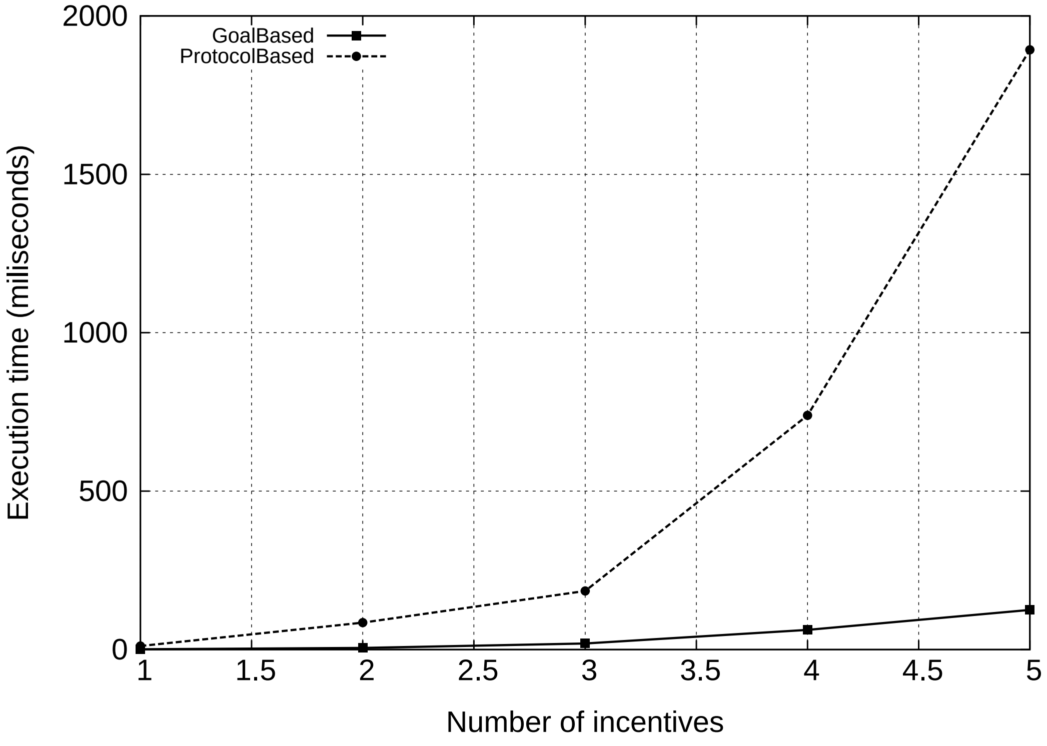

Figure 3.9. Comparison of execution times of ProtocolBased and GoalBased algorithms based on the number of preconditions ($\rho = 2$) and incentives ($\varepsilon = 1, \dots, 5$) for each service.

putation. In order to examine this issue in detail, we conducted another experiment, in which we fixed the number of alternative services and used different values for ρ and ε. These parameters have the following effects on the execution of our algorithms. If there are two incentives for a service (i.e., $\varepsilon = 2$), then our algorithms generate two alternative protocols for each service, such that each protocol includes a commitment that contains one of these incentives to use the service. On the other hand, since both protocols use the same service, all the preconditions of this service should be supported by both protocols. For example, if there are two incentives v_1 and v_2 for a goal and two preconditions q_1 and q_2 for the service that brings about the goal, then there are two protocols, such that the first protocol supports v_1, q_1 and q_2, and the second protocol supports v_2, q_1 and q_2. Since ProtocolBased algorithm does not reuse sub-protocols, for this example it has to find support for q_1 and q_2 redundantly

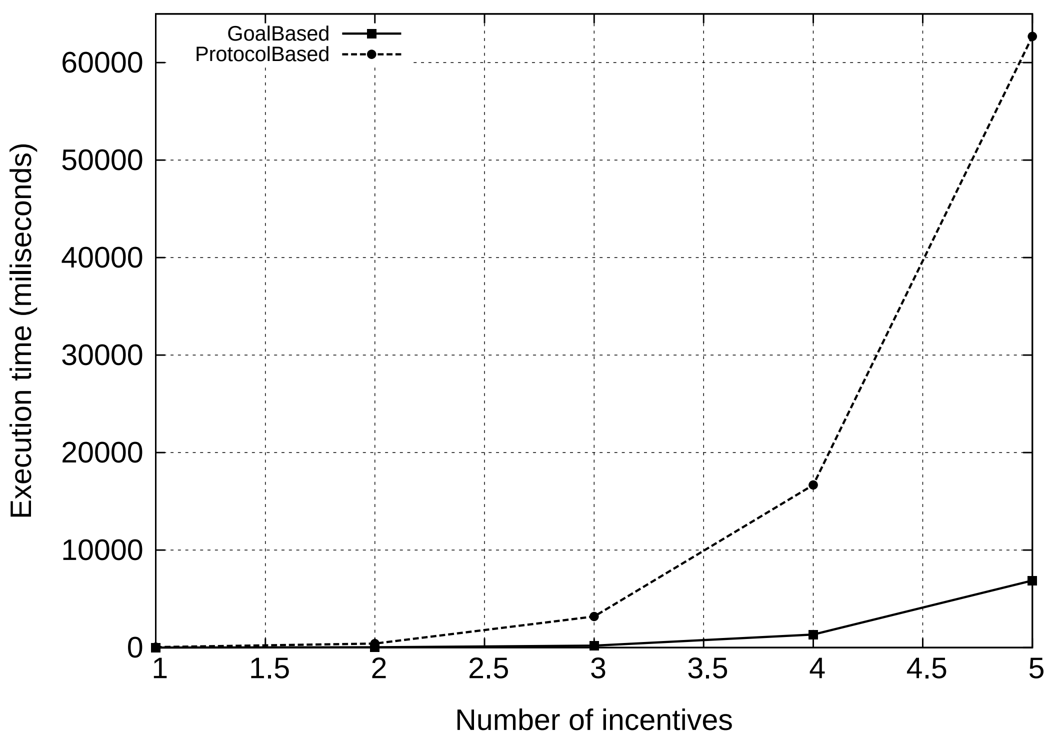

Figure 3.10. Comparison of execution times of ProtocolBased and GoalBased algorithms based on the number of preconditions ($\rho = 3$) and incentives ($\varepsilon = 1, \ldots, 5$) for each service.

for each protocol. More generally, when ε grows, the number of redundant computations of ProtocolBased algorithm also grows, since more incentives means more protocols. On the other hand, ρ determines the total number of preconditions for a service and a higher value for ρ also results in more redundant computation done by ProtocolBased algorithm for each protocol, since there are more preconditions to be supported.

We present our results in Figures 3.8, 3.9 and 3.10. Each figure shows the execution time of the ProtocolBased and GoalBased algorithms for different values of ε (i.e., number of incentives) between one and five. Additionally, in each figure the value of ρ (i.e., number of preconditions) is different. Specifically, ρ is equal to one, two and three in Figures 3.8, 3.9 and 3.10, respectively. For this experiment, we use the other parameters as $\gamma = 1$, $\sigma = 3$ and $\lambda = 3$. In Figure 3.8 the number of preconditions for each service is just one. Hence, the time required to find support for a service's precondition is short. Accordingly, even though the amount of redundant computation increases with the number of incentives (i.e., ε), ProtocolBased algorithm's execution time still increases almost linearly. However, as Figures 3.9 and 3.10 show, when the time required to find support for a service's precondition is longer, since there are more preconditions (i.e., ρ grows), the total execution time of ProtocolBased algorithm grows quickly with the number of incentives. On the other hand, for all ε and ρ values, the GoalBased algorithm scales well thanks to use reuse of sub-protocols via memoization.

3.5. Discussion

In this chapter we presented two novel algorithms that use depth-first traversal, and divide and conquer strategies, respectively, to generate commitment protocols. The protocols generated by these algorithms support the goals of the leading agent. Besides, these algorithms take other agents' goals into account to create incentive for the provision of their services. We showed the formal properties of these algorithms and also compared them and showed that GoalBased algorithm is computationally more efficient than the ProtocolBased algorithm. Besides, GoalBased algorithm is advantageous because of its modular nature.

The proposed algorithms can be improved in several ways. The algorithms that we developed generate commitment protocols from scratch. However, in many cases some protocols may already be known by an agent and can be reused to support some of the agent's goals. These protocols may already be generated to regulate a previous interaction or may be provided to the agent as part of a protocol library, when the agent is first developed. For example, the customer in our running example might already know a commitment protocol to purchase materials from the retailer. If this was the case, the known commitment protocol could be used instead of generating the same protocol again. In the case of GoalBased algorithm, reuse of existing protocol can easily be achieved by keeping these protocols in the mapping that we use for memoization. On the other hand, in the case of ProtocolBased algorithm, reuse of such protocols is not so straightforward because of the reasons that we discussed before. A potential area of improvement for our algorithms is parallelization. We implement both algorithms sequentially. However, especially the GoalBased algorithm is suitable for parallel implementation, because of its modular nature.

4. RANKING OF COMMITMENT PROTOCOLS

After the protocols are generated in the first phase of our process, the next phase is evaluation and ranking of these protocols according to the leading agent's preferences (Figure 1.1). Results of this ranking will provide guidance to the leading agent in the last phase of our process, while negotiating over protocols with other agents.

Ranking of protocols means evaluating each protocol according to a set of criteria, which is important for the agent, and sorting them in some order based on the results of this evaluation. Here, we consider two such criteria: (i) *utility* of a protocol, which is the benefit of enacting the protocol from the agent's point of view , and (ii) *risk-discounted utility* of a protocol, which is the utility of the protocol taking the failure risk of the protocol into account. Note that both of these criteria are subjective to the evaluating agent. Hence, evaluation and ranking of a protocol may vary for different agents. That is, two different agents may have different rankings of a given protocol.

The *utility* of a protocol for an agent reflects how much an agent would gain by enacting the protocol. The utility is basically the difference between the benefit of the protocol and its cost (*utility* = *benefit* − *cost*). Intuitively, each agent would want to maximize its utility, given that two protocols achieve the same goals. There may also be some protocols that are not acceptable to a given agent (e.g., the utility of the protocol is negative, which means the cost exceeds the benefit).

The *risk-discounted utility* extends the utility by also considering the failure risk of the protocol. Since various agents are involved in the enactment of the protocol, it is possible for some of the agents to fail to fulfill their commitments, which leads to failure of the protocol meaning that the expected benefit may not be realized. We therefore extend the definition of utility by *discounting* the benefits based on the failure risk of the protocol. We compute the risk of a protocol based on the trustworthiness of the enacting agents' services that they provide in the context of the protocol. If all agents are fully trustworthy, then there is no risk associated with the protocol. In that situation, there is no risk-based discount, and the utility of the protocol is the only metric that sets two protocols apart. However, if any of the agents in the protocol are somewhat untrustworthy, then the protocol has a failure risk, and the expected utility of the protocol may be jeopardized.

4.1. Utility of Protocols

4.1.1. Utility Ranking Method

We define the utility of a protocol in terms of the benefit that the agent may derive from the protocol's successful enactment, discounted by the cost that the agent incurs in playing its part in the protocol (i.e. *utility* = *benefit* − *cost*). As noted earlier, this means that the utility of a protocol is specific to an agent: in assessing benefit vs. cost, an agent is considering the benefit that it derives, and the costs that it incurs. We assume that each agent is aware of the benefit that it derives from each condition u, which we denote as $Benefit_x(u)$. Similarly, we also assume that each agent is aware of the cost of performing each of its capabilities $f = F_x(\theta, u)$, which we denote as $Cost_x(F_x(\theta, u))$ (or simply $Cost_x(f)$). Note that agents do not need to know each others assigned benefits or costs.

An agent may have more than one capability to satisfy a condition (e.g., an agent may pay using different mediums such as cash, credit card, etc.). Accordingly, given the costs of using capabilities, we define the cost of satisfying a *condition* as the maximal possible cost over the capabilities that can be used to satisfy the condition. We use a maximum since in general we may not be able to decide or control which service is usable to achieve the desired property. Formally:

$$Cost_x(u) = \max_{F_x(\theta', u') \in \mathcal{F} \text{ and } u' \Rightarrow u} Cost_x(F_x(\theta', u'))$$

To calculate the overall benefit and cost of a protocol we need to consider all the conditions involved in the protocol, and then sum their benefits and costs. However, we cannot just calculate the benefit and cost for each commitment and then add them up, because a condition that occurs in more than one commitment is only achieved once. For example, in protocol π_5 (Table 3.1 on page 42), the customer pays for tools only once, even though $ToolsPaid$ appears in two commitments in the protocol. In other words, we compute the benefit and the cost of a protocol by accumulating the relevant conditions in the protocol, then working out what capabilities are needed, and finally working out their costs.

Before defining the utility of a protocol formally, we explain what we mean by *relevant conditions*. When considering a commitment $c = C(x, y, \theta, u)$, the agent x (i.e., debtor of the commitment) is responsible for satisfying u, which is the relevant condition for x in c. Similarly, the relevant conditions for y (i.e., the creditor of the commitment) is θ. Hence, considering c from x's perspective, u should be taken into account while computing cost and θ should be taken into account while computing benefit. On the other hand, when considering c from y's perspective, this is reversed: u should be taken into account while computing benefit and θ should be taken into account while computing cost. Formally, we define this in the following using the auxiliary functions $rel_x(c)$ for cost and $\widehat{rel}_x(c)$ for benefit.

$$\begin{aligned}
rel_x(C(x_1, x_2, \{q_1, \ldots, q_n\}, u)) &= \bigcup_{i=1\ldots n} rel_x(C(x_1, x_2, q_i, u)) \\
rel_x(C(x_1, x_2, q, u)) &= \begin{cases} \{u\}, & \text{if } x = x_1 \\ \{q\}, & \text{if } x = x_2 \\ \emptyset, & \text{otherwise} \end{cases} \\
\widehat{rel}_x(C(x_1, x_2, \{q_1, \ldots, q_n\}, u)) &= \bigcup_{i=1\ldots n} \widehat{rel}_x(C(x_1, x_2, q_i, u)) \\
\widehat{rel}_x(C(x_1, x_2, q, u)) &= \begin{cases} \{q\}, & \text{if } x = x_1 \\ \{u\}, & \text{if } x = x_2 \\ \emptyset, & \text{otherwise} \end{cases}
\end{aligned}$$

In essence, $rel_x(c)$ extracts the conditions that agent x is responsible for satisfying in c and $\widehat{rel}_x(c)$ extracts the conditions in c, which are under the responsibility of the other agent. Note that if the antecedent of c is a set then we break it up (first case above). Also note that if x is neither the debtor nor the creditor in c, then it does not have any responsibilities with respect to c (i.e., $rel_x(c) = \emptyset$ and $\widehat{rel}_x(c) = \emptyset$).

Now we are ready to formally define the utility of a protocol π that is derived to achieve a set of goals $\mathcal{G}$, from the perspective of agent x (denoted $\mathit{Utility}_x(\pi)$) as follows:

$$\begin{aligned}
\mathit{Utility}_x(\pi) &= \mathit{Benefit}_x(\pi) - \mathit{Cost}_x(\pi) \\
\mathit{Benefit}_x(\pi) &= \sum_{u \in \theta \cup \theta'} \mathit{Benefit}_x(u) \\
&\quad \text{where } \theta = \bigcup_{c \in \pi} \widehat{rel}_x(c) \text{ and } \theta' = \{u \mid G_x(u) \in \mathcal{G}\} \\
\mathit{Cost}_x(\pi) &= \sum_{u \in \theta} \mathit{Cost}_x(u) \\
&\quad \text{where } \theta = \bigcup_{c \in \pi} rel_x(c)
\end{aligned}$$

Note that when calculating the benefit we also include the top-level goals θ'. These goals are achieved by all of the protocols, but are not always explicitly represented as a condition within the generated commitments. For example, suppose that in our running example the customer herself might be capable of assembling the furniture, if she had the tools and materials (e.g., there is a capability such as $F_{Mer}(\{HaveTools, HaveMaterials\},\ HaveFurniture)$). If this was the case, there would be a protocol π that supports the top-level goal $HaveFurniture$, including only the commitments to obtain the tools and the materials from the retailer. But $HaveFurniture$ would not appear in any commitment in π, although $HaveFurniture$ would be achieved once π was enacted.

4.1.2. Protocol Evaluation using Utility Ranking

In order to evaluate the protocols that are generated by our algorithms for our running example, we show example benefits and costs values in Table 4.1 and 4.2, respectively. We picked these to reflect realistic assumptions about our running example. For instance, having furniture (*HaveFurniture*) has the highest benefit, since this is the ultimate goal of the customer. Having tools (*HaveTools*) has some benefit since tools can be reused but yields less benefit compared to having the furniture. The scenario that we model assumes that the customer provides tools to builders as a temporary loan, hence the benefit for having tools is realized even in protocols that involve providing (i.e., lending) the tools to the builder (i.e., making *ToolsProvided* true). On the other hand, having materials by itself has no benefit, since they are useless for the customer.

Table 4.1. Condition benefits.

Condition	Benefit
HaveFurniture	15
HaveTools	8
HaveMaterials	0

In terms of costs, we select costs that make sense for the domain. Specifically, we select costs that make buying furniture from the Merchant cheaper than the combined cost of obtaining tools, materials, and paying the second builder (but the first builder is still cheaper, since only materials has to be provided). However, if the agent already has tools and materials, then paying the second builder is cheaper than buying ready-made furniture from the merchant. Specifically, if the customer does not already have tools and materials, then the cost of obtaining the furniture is less than the costs of first obtaining materials ($f_4 = 2$) and tools ($f_5 = 3$), then providing the tools ($f_1 = 5$) and the materials to the second builder ($f_3 = 1$) and finally paying for assembly ($f_8 = 5$). However, if the customer already has materials and tools, then the cost of buying furniture ($f_6 = 12$) is greater than the combined costs of providing the materials to the builder ($f_3 = 1$), providing tools ($f_1 = 5$) and paying for assembly ($f_8 = 5$).

Table 4.2. Capability costs.

ID	Capability	Cost
f_1	$F_{Cus}(HaveTools, ToolsProvided)$	5
f_2	$F_{Cus}(HaveMaterials, Bui_1MaterialsProvided)$	1
f_3	$F_{Cus}(HaveMaterials, Bui_2MaterialsProvided)$	1
f_4	$F_{Cus}(\top, MaterialsPaid)$	2
f_5	$F_{Cus}(\top, ToolsPaid)$	3
f_6	$F_{Cus}(\top, FurniturePaid)$	12
f_7	$F_{Cus}(\top, Bui_1Paid)$	4
f_8	$F_{Cus}(\top, Bui_2Paid)$	5

To demonstrate computation of a protocol's utility, we consider protocol $\pi_4 = \{c_1, c_2, c_3\}$ from Table 3.1 (page 42) as an example in which the commitments are:

- $c_1 = C(Ret, Cus, MaterialsPaid, HaveMaterials)$
- $c_2 = C(Ret, Cus, ToolsPaid, HaveTools)$
- $c_3 = C(Bui_2, Cus, \{Bui_2MaterialsProvided, ToolsProvided, Bui_2Paid\}, HaveFurniture)$

To derive the cost of π_4 from the customer's perspective, the relevant conditions (that the customer has to satisfy) are *ToolsPaid*, *Bui*$_2$*MaterialsProvided*, *Tools-Provided*, and *Bui*$_2$*Paid*. Each of these conditions in fact can only be achieved by a single capability (respectively f_5, f_3, f_1 and f_8). Using the example costs in Table 4.2, the cost of π_4 from the customer's perspective is just: $Cost_{Cus}(f_5) + Cost_{Cus}(f_3) + Cost_{Cus}(f_1) + Cost_{Cus}(f_8) = 3 + 1 + 5 + 5 = 14$. On the other hand, from the customer's perspective, benefit of π_4 is the benefit of the relevant conditions, which are *HaveMaterials*, *HaveTools* and *HaveFurniture*. Thus, benefit of π_4 is $Benefit_{Cus}(HaveMaterials) + Benefit_{Cus}(HaveTools) + Benefit_{Cus}(HaveFurniture) = 0 + 8 + 15 = 23$. As result, utility of π_4 is $23 - 14 = 9$.

Table 4.3 provides an evaluation of the generated protocols (Table 3.1 on page 42).

Table 4.3. Customer's evaluation of protocol utility.

Protocol	Benefit	Cost	Utility	Cost Rank	Utility Rank
π_1	15	12	3	3	7
π_2	15	7	8	1	3
π_3	15	8	7	2	5
π_4	23	16	7	6	5
π_5	23	14	9	5	2
π_6	23	13	10	4	1
π_7	23	16	7	6	5

The first three labeled columns denote the benefit, cost and utility of the protocol, respectively and the remaining columns are the respective ranks (i.e., the protocol with the lowest cost has Cost Rank of 1, and the protocol with the highest utility has Utility Rank of 1). We split ties (e.g., π_4 and π_7 have the same cost, and so instead of ranking them as 6^{th} and 7^{th}, they are both given rank 6).

In Table 4.3, π_6 is the top ranked protocol with respect to utility. In π_6 the customer offers $MaterialsPaid$ (i.e., f_4 that costs 2) to the retailer to purchase both the tools and materials. Hence, the cost of the protocol is low comparing to its high benefit. However, note that in a realistic situation this would cause negative utility from the retailer's point of view and would reduce acceptability of the protocol. The situation is similar for π_5 also where the customer offers $ToolsPaid$ both for tools and also materials. In π_4, the customer offers $ToolsPaid$ for tools and $MaterialsPaid$ for materials to the retailer. Accordingly, the cost of the protocol is higher and utility rank if π_4 is lower than π_5 and π_6. In π_2, the overall benefit of the protocol is lower since the customer does not own the tools at the end of the protocol. However, the cost is also lower, since the customer does not have to buy the tools. In π_3, instead of $MaterialsPaid$ the customer offers $ToolsPaid$, which is more expensive than $MaterilasPaid$, to the retailer. Accordingly, cost of this protocol is higher than π_2, even though benefits of these protocols are equal. Finally, although it includes only a single commitment, π_1 has the lowest utility, since the cost of f_6 is considerably higher comparing to other services.

Independent from utility, a protocol's cost itself is an important indicator for its usefulness, since cost is a bound on the worst case. In other words, if other agents fail to fulfill their commitments, then this is the highest possible cost that the agent will pay whilst receiving no benefit. Note that this "worst case scenario" may actually be too pessimistic. For example, in π_4, if the retailer failed to fulfill her commitments (the first two commitments, satisfying $HaveMaterials$ and $HaveTools$), then the customer would not be able to proceed and would not be able to interact with the builder. Hence, even though the cost of the protocol is 16, this cost could only be incurred, if the retailer fulfilled her role, but the builder failed to do so. Furthermore, even if this was the case, the customer would still derive some benefit from having tools. The dual of this worst case scenario is the "best case scenario", which occurs when everyone fulfills their commitments. In that case, the agent will also receive some benefit and the overall gain will be reflected in the utility. For example, even though π_4 incurs the highest cost, it is not the worst protocol in terms of utility since it has a benefit of 23 and the overall utility comes out to be 7. On the other hand π_1 is the worst protocol in this respect, since even if all the agents fulfill their commitments, the gain is only 3.

4.2. Incorporating Risk

As expected, looking from the worst-case and the best-case ranks the protocols differently. The next obvious issue is the likelihood of the worst-case. This intuitively will help us to understand how risky a protocol is. In this section we extend the definition of benefit to include a risk assessment. In essence, we *discount* the benefit of a protocol by the risk that the benefit may not be realized.

4.2.1. Trust Based Ranking Method

In order to quantify the risk of a protocol, we start from the trust relation among the agents that are involved in that protocol. In general, an agent's trust in another depends on the particular service in question. For example, the customer might trust a merchant for delivering furniture, but might not trust her for actually assembling the furniture. Hence, we consider the trust of an agent x in another agent y with respect to a particular service $S_x(y, \theta, u)$, denoted as $\mathit{Trust}_x(y, S_x(y, \theta, u)) \in [0, 1]$. This trust value represents the completion likelihood of $S_x(y, \theta, u)$ by y from x's perspective. As is customary in the trust literature, we assume that the higher values represent more trust whereas lower values represent lower trust. Hence, a trust value of 1 means that x believes y would definitely carry out the service whereas a trust value of 0 means that x believes y would definitely *not* carry out the service. In general, the trust values are updated dynamically based on the outcomes of agents' interactions or new information coming in about the agent from other (trusted) sources. There are many existing mechanisms in the literature for managing and disseminating trust (e.g. [27–29]). Here, we assume that the agent has such a mechanism to maintain an accurate trust evaluation of other agents.

The actual value of $\mathit{Trust}_x(y, S_x(y, \theta, u))$ is subject to a number of constraints. First, if the service is to satisfy a trivial condition, then trust is always 1. Formally, $\mathit{Trust}_x(y, S_x(y, \theta, \top)) = 1$. Second, since we consider trust between agents based on a particular service, x can only consider y trustful for a service s that x believes y provides. Formally, if there does not exist a service $S_x(y, \theta, u)$ in $\mathcal{B}$ (recall that $\mathcal{B}$ is the agent's beliefs about other agents' services) then $\mathit{Trust}_x(y, S_x(y, \theta, u)) = 0$. Third, an agent always trusts itself about its capabilities. Note that in order to simplify the definitions in this section we assume for convenience that for any capability $F_x(\theta, u)$ that an agent x has in $\mathcal{F}$, there is also a corresponding implicit service belief: $S_x(x, \theta, u) \in \mathcal{B}$. Accordingly, we represent an agent's trust to its capabilities formally, $\mathit{Trust}_x(x, S_x(x, \theta, u)) = 1$.

As it is customary, an agent can provide several services to bring about a condition. Accordingly, we proceed to define the trust that an agent x has in agent y to bring about a condition u (denoted $\mathit{Trust}_x(y, u)$) in terms of the relevant services. One special case is that if y does not have any relevant services, then the trust is 0. Otherwise we define $\mathit{Trust}_x(y, u)$ by considering all the relevant services that y can provide to satisfy u:

$$\begin{aligned} Trust_x(y, \top) &= 1 \\ Trust_x(y, u) &= \begin{cases} 0, \quad \text{if } \nexists\, S_x(y, \theta, u) : S_x(y, \theta, u) \in \mathcal{B} \\ \bigoplus_{S_x(y,\theta,u) \in \mathcal{B}} Trust_x(y, S_x(y, \theta, u))), \quad \text{otherwise} \end{cases} \end{aligned}$$

Thus, x considers all services that would enable u to be realized and combines the trust for these services using an auxiliary function $\oplus$. This auxiliary function can be defined using max, meaning that the combined trust can at most be equal to the most trusted service. For example, if y can provide n services to bring about u, then $Trust_x(y, u) = Trust_x(y, s_1) \oplus \ldots \oplus Trust_x(y, s_n) = \max(Trust_x(y, s_1), \ldots, Trust_x(y, s_n))$.

This definition considers the trust independent from a particular protocol. That is the trust of agent x in agent y to satisfy u is computed with respect to all services provided by y to satisfy u. But, when we deal with satisfaction of a condition in the context of a protocol, we may not consider every service that is provided by an agent to satisfy this condition, because of services' preconditions. Specifically, in order to be able to consider a service to satisfy a condition in the context of a protocol, we should check that the service's preconditions are supported in the context of the protocol. If this is not the case, then it is clear that the service cannot be utilized in the context of the protocol. Consider the protocol $\pi = \{C(y, x, \{q, w\}, u)\}$. Suppose that x believes that y has two services, $s = S_x(y, q, u)$ and $s' = S_x(y, q', u)$ to satisfy u. Also assume that x is capable of satisfying q and w, unconditionally. By examining s, we can see that s's precondition is supported by π, since it's precondition q is included in the antecedent of the commitment and x is capable of q. On the other hand, this is not the case for s', since it's precondition is not included in the commitment's antecedent. As a result, when assessing x's trust in y to satisfy u in the context of π, we should only consider s and ignore s'.

Beside determination of the services that can be utilized in the context of a protocol, another issue that we should take into account while assessing trust in a condition is the agent's trust in preconditions and incentives that are involved in the commitments. Consider the above example again. There, it is not enough to use only x's trust in y for s. Instead we should have take x' trust in q and w also into account, since only if q and w is achieved, then s can be utilized.

Accordingly, we generalize assessment of x's trust in u with respect to the protocol π as follows:

$$\begin{aligned} \mathit{Trust}_x^{\pi}(\top) &= 1 \\ \mathit{Trust}_x^{\pi}(u) &= \begin{cases} 0, \quad \text{if } \nexists\, C(y, x, \theta \cup \{w\}, u) : C(y, x, \theta \cup \{w\}, u) \in \pi \\ \bigoplus_{S_x(y,\theta,u) \in \mathcal{S}} \mathit{Trust}_x(y, S_x(y, \theta, u)) \times \Big(\otimes_{q \in \theta \cup \{w\}} \mathit{Trust}_x^{\pi}(q) \Big), \text{where} \\ \qquad \mathcal{S} = \{S_x(y, \theta, u) : S_x(y, \theta, u) \in \mathcal{B} \; \mathit{and} \\ \qquad\qquad C(y, x, \theta \cup \{w\}, u) \in \pi\}, \text{ otherwise} \end{cases} \end{aligned}$$

In the first case, trust to a trivial condition is equal to 1 as before. In the second case, if the protocol does not include a commitment in which the consequent is equal to the condition u, then the trust is 0. In other words, the condition is not supported by the protocol and accordingly it is not possible to satisfy the condition using this protocol. Otherwise, if there is a commitment, first the services that can be utilized in the context of this protocol is determined and stored in S. Then trust in these services is computed taking trust in their preconditions and incentives, into account. For this purpose, trust in a service is multiplied with the combined trust of the corresponding preconditions and incentive. To compute the combined trust we use an auxiliary operator $\otimes$, which can be defined in different ways. Here we use $\otimes$ as multiplication.

Now, given that the agent can assess the trust value for a condition with respect to a protocol, we reflect this in the calculation of the benefit, to derive an *expected value* on the utility. The intuition is to calculate how likely it is that the commitment will be fulfilled, thereby yielding its expected value. Hence, rather than using the utility measure we defined in Section 4.1, we extend it by adding $\mathit{Trust}_x^\pi(u)$. Note that below we use cost as we defined in Section 4.1:

$$\begin{aligned}
\mathit{Utility}_x(\pi) &= \mathit{Benefit}_x(\pi) - \mathit{Cost}_x(\pi) \\
\mathit{Benefit}_x(\pi) &= \sum_{u \in \theta \cup \theta'} \mathit{Benefit}_x(u) \times \mathit{Trust}_x^\pi(u) \\
&\quad \text{where } \theta = \bigcup_{c \in \pi} \widehat{\mathit{rel}}_x(c) \text{ and } \theta' = \{u \mid G_x(u) \in \mathcal{G}\}
\end{aligned}$$

The difference here, compared with the previous definition, is that the benefit is being *discounted* based on the trust that agent x has in the ability of the condition u to be brought about by the relevant agents in the system. Note that if the trustworthiness ($\mathit{Trust}_x^\pi(u)$) is 0, then the benefit will come down to 0. Conversely, if the trustworthiness is 1, then the utility will reflect the best outcome, reflecting the “best-case scenario”.

4.2.2. Protocol Evaluation using Trust Ranking

In order to evaluate the protocols that are generated by our algorithms for our running example incorporating trust, we show example trust values in Table 4.4. We use the values in Table 4.1 and 4.2 as before to compute benefit and cost, respectively.

Here, we consider $\pi_4 = \{c_1, c_2, c_3\}$ again to show computation of a protocol’s utility applying our risk-based discount. The commitments of π_4 are:

- $c_1 = C(Ret, Cus, MaterialsPaid, HaveMaterials)$
- $c_2 = C(Ret, Cus, ToolsPaid, HaveTools)$
- $c_3 = C(Bui_2, Cus, \{Bui_2MaterialsProvided, ToolsProvided, Bui_2Paid\},$ $HaveFurniture)$

Table 4.4. Customer's trust in services.

Service	Trust
$S_{Cus}(Ret, \top, HaveMaterials)$	0.7
$S_{Cus}(Ret, \top, HaveTools)$	0.6
$S_{Cus}(Mer, \top, HaveFurniture)$	0.9
$S_{Cus}(Bui_1, Bui_1MaterialsProvided, HaveFurniture)$	0.8
$S_{Cus}(Bui_2, Bui_2MaterialsProvided, ToolsProvided, HaveFurniture)$	0.2

The cost of π_4 is calculated as in the previous section. We first determine the set of relevant conditions for the customer, which are $MaterialsPaid$, $ToolsPaid$, Bui_2-$MaterialsProvided$, $ToolsProvided$, and Bui_2Paid. We then determine the cost of each condition, and take the sum of them. In this case each condition has exactly one relevant capability (respectively f_4 with cost 2; f_5 with cost 3; f_3 with cost 1; f_1 with cost 5; and f_8 with cost 5), which gives a total cost of $2 + 3 + 1 + 5 + 5 = 16$.

Now we compute the discounted benefit of protocol π_4. We firstly determine the set of relevant conditions, which are $HaveMaterials$, $HaveTools$ and $HaveFurniture$ and the top-level goal set, which includes only $HaveFurniture$. Note that $Have$-$Furniture$ is included in both sets, but since we take the union of these sets, it is considered only once. Accordingly, for each of the three resulting conditions, we look up the benefit (Table 4.1, on page 61) and compute the risk-based discount:

$$\begin{aligned} \mathit{Benefit}_{Cus}(\pi_4) &= \mathit{Benefit}_{Cus}(HaveMaterials) \times \mathit{Trust}^{p4}_{Cus}(HaveMaterials) \\ &\quad +(\mathit{Benefit}_{Cus}(HaveTools) \times \mathit{Trust}^{p4}_{Cus}(HaveTools) \\ &\quad +(\mathit{Benefit}_{Cus}(HaveFurniture) \times \mathit{Trust}^{p4}_{Cus}(HaveFurniture) \\ &= 0 + 8 \times \mathit{Trust}^{p4}_{Cus}(HaveTools) + 15 \times \mathit{Trust}^{p4}_{Cus}(HaveFurniture) \end{aligned}$$

We now consider the values of $\mathit{Trust}^{\pi_4}_{x}(HaveTools)$ and $\mathit{Trust}^{\pi_4}_{x}(HaveFurniture)$. For $\mathit{Trust}_{Cus}(p_4, HaveTools)$ we consider the commitments in π_4 and find that only c_2 is relevant, and hence the retailer is the only relevant agent with service s_2. Similarly, for $\mathit{Trust}^{\pi_4}_{Cus}(HaveFurniture)$ there is only a single relevant commitment (c_3) and only

Bui_2 is relevant with service s_5. Note that in both cases the customer herself does not have a relevant capability. Also note that although services s_3 and s_4 of the merchant and the first builder can be utilized, respectively, to satisfy $HaveFurniture$, these two services are also irrelevant in the context of π_4, since these agents are not the debtor of c_3 and the antecedent of c_3 does not include precondition of these services. Computation of $Trust^{\pi_4}_{Cus}(HaveTools)$ is straightforward, since the relevant service s_2 does not have a precondition. We therefore have:

$$\begin{aligned} Trust^{\pi_4}_{Cus}(HaveTools) &= Trust_{Cus}(Ret, s_2) \times Trust^{\pi_4}_{Cus}(\top) \\ &= 0.6 \times 1 \\ &= 0.6 \end{aligned}$$

On the other hand, while computing $Trust^{\pi_4}_{Cus}(HaveFurniture)$, we have to take the customer's trust in satisfaction of s_5's preconditions (i.e., $Bui_2MaterialsProvided$ and $ToolsProvided$). We therefore have:

$$\begin{aligned} Trust^{\pi_4}_{Cus}(HaveFurniture) = \; & Trust_{Cus}(Bui_2, s_5) \times \\ & \Big(Trust^{\pi_4}_{Cus}(Bui_2MaterialsProvided) \otimes \\ & \quad Trust^{\pi_4}_{Cus}(ToolsProvided) \Big) \end{aligned}$$

In order to satisfy $Bui_2MaterialsProvided$, there is no relevant agent in π_4, but the customer herself has a relevant capability f_3 for this purpose. Accordingly, we have:

$$Trust^{\pi_4}_{Cus}(Bui_2MaterialsProvided) = Trust_{Cus}(Cus, f_3) \times Trust^{\pi_4}_{Cus}(HaveMaterials)$$

For $Trust^{\pi_4}_{Cus}(HaveMaterials)$ the relevant commitment in π_4 is c_1, and therefore the retailer is the only relevant agent with service s_1. So we have:

$$\begin{aligned} Trust^{\pi_4}_{Cus}(HaveMaterials) &= Trust_{Cus}(Ret, s_1) \times Trust^{\pi_4}_{Cus}(\top) \\ &= 0.7 \times 1 \\ &= 0.7 \end{aligned}$$

Then, turning back to $Trust^{\pi_4}_{Cus}(Bui_2MaterialsProvided)$ we have:

$$\begin{aligned} Trust^{\pi_4}_{Cus}(Bui_2MaterialsProvided) &= Trust_{Cus}(Cus, f_2) \times Trust^{\pi_4}_{Cus}(HaveMaterials) \\ &= 1 \times 0.7 \\ &= 0.7 \end{aligned}$$

Similarly, in order to satisfy $ToolsProvided$, there is no relevant agent in π_4, but the customer herself has a relevant capability f_1 for this purpose. Accordingly, we have:

$$Trust^{\pi_4}_{Cus}(ToolsProvided) = Trust_{Cus}(Cus, f_1) \times Trust^{\pi_4}_{Cus}(HaveTools)$$

For $Trust^{\pi_4}_{Cus}(HaveTools)$ the relevant commitment in π_4 is c_2, and therefore the retailer is the only relevant agent with service s_2. So we have:

$$\begin{aligned} Trust^{\pi_4}_{Cus}(HaveTools) &= Trust_{Cus}(Ret, s_2) \times Trust^{\pi_4}_{Cus}(\top) \\ &= 0.6 \times 1 \\ &= 0.6 \end{aligned}$$

Then, turning back to $Trust^{\pi_4}_{Cus}(ToolsProvided)$ we have:

$$\begin{aligned} Trust^{\pi_4}_{Cus}(ToolsProvided) &= Trust_{Cus}(Cus, f_1) \times Trust^{\pi_4}_{Cus}(HaveTools) \\ &= 1 \times 0.6 \\ &= 0.6 \end{aligned}$$

Now we have $Trust^{\pi_4}_{Cus}(Bui_2MaterialsProvided)$ and $Trust^{\pi_4}_{Cus}(ToolsProvided)$ and we can turn back to $Trust^{\pi_4}_{Cus}(HaveFurniture)$.

$$\begin{aligned}
\mathit{Trust}^{\pi_4}_{Cus}(HaveFurniture) &= \mathit{Trust}_{Cus}(Bui_2, s_5) \times \\
&\quad \Big(\mathit{Trust}^{\pi_4}_{Cus}(Bui_2MaterialsProvided) \otimes \\
&\qquad \mathit{Trust}^{\pi_4}_{Cus}(ToolsProvided)\Big) \\
&= 0.2 \times \Big(0.7 \otimes 0.6\Big) \\
&= 0.084
\end{aligned}$$

Finally, since we have $\mathit{Trust}^{p_4}_{Cus}(HaveTools)$ and $\mathit{Trust}^{p_4}_{Cus}(HaveFurniture)$, now we can compute $\mathit{Benefit}_{Cus}(\pi_4)$:

$$\begin{aligned}
\mathit{Benefit}_{Cus}(\pi_4) &= \mathit{Benefit}_{Cus}(HaveMaterials) \times \mathit{Trust}^{p_4}_{Cus}(HaveMaterials) \\
&\quad +(\mathit{Benefit}_{Cus}(HaveTools) \times \mathit{Trust}^{p_4}_{Cus}(HaveTools) \\
&\quad +(\mathit{Benefit}_{Cus}(HaveFurniture) \times \mathit{Trust}^{p_4}_{Cus}(HaveFurniture) \\
&= 0 + 8 \times 0.6 + 15 \times 0.084 \\
&= 6.06
\end{aligned}$$

Since the cost of π_4 is 16 and the risk-discounted benefit is 6.06, we have that the overall expected value of the utility is $6.06 - 16 = -9.94$. In other words, since the customer's trust in the retailer is not that high, and the customer's trust in the second builder is also quite low, protocol π_4 is too risky to be considered acceptable.

Table 4.5 shows the results for all of the protocols that are generated for our running example. The first column shows the protocol identifiers. The second column shows protocol benefits as computed in Section 4.1 and the third column shows protocol benefits after application of risk discount. The fourth column shows protocol costs as before. Fifth column shows expected utility values (i.e., risk-discounted benefit minus cost). The last column shows expected ranks of protocols.

These result show that the rank of a protocol depends on both the utility of the

Table 4.5. Customer's evaluation of protocol's expected utility values.

Protocol	Benefit	Risk-Discounted Benefit	Cost	Expected Value for Utility	Expected Rank
π_1	15	13.5	12	1.5	1
π_2	15	8.4	7	1.4	2
π_3	15	8.4	8	0.4	3
π_4	23	6.06	16	-9.94	6
π_5	23	6.06	14	-7.94	5
π_6	23	6.06	13	-6.94	4
π_7	23	6.06	16	-9.94	6

protocol and the trustworthiness of the participating agents.

In our previous results (in Table 4.3, on page 63) where we do not consider risk, π_1 is the worst protocol in terms of utility, because of its high cost. But when we take risk into account, it is the best protocol in terms of expected value. This is mainly because of the high trustfulness of the merchant's service s_3, which is the only service utilized in π_1. For other protocols, the major factor that determine the rankings is the trustfulness of the builders. Since the customer's trust in the first builder is quite high, π_2 and π_3 are ranked second and third, respectively. On the other hand, the customer's trust in the second builder is considerably low. Accordingly, benefits of protocols $\pi_4 - \pi_7$ in which the second builder participates, are discounted significantly. Moreover, expected utility values of these protocols are negative, since their costs are also higher than the other protocols. Other than that, since π_2 and π_3 have the same risk discounted benefit, their final ordering is determined according to their costs where the cost of π_2 is lower than π_3 and consequently π_2 is ranked on top of π_3. The same situation applies also for $\pi_4 - \pi_7$.

These results show that even though a protocol has a potential to create a high utility for an agent, if others do not play their parts, the outcome might not be desired (as in protocols $\pi_4 - \pi_7$). On the other hand, a protocol that has a low utility may be preferable if the participants are trustworthy, which increase likelihood of receiving the benefit, once the protocol is executed.

4.3. Discussion

In this chapter we presented a method to evaluate and rank commitment protocols. In the first part we provided the equations to compute the utility of a commitment protocol from an agent's perspective. Our method utilizes subjective benefit and cost of a protocol to compute the utility of the protocol. In the second part we extended this formulation by incorporating trust of the agent in others. In this way we take the likelihood of protocol completion, which we represent as combination of trust values over conditions, into account while computing a protocol's utility. Our evaluations on our running example show that even if a protocol has a high utility value, it may not be the best option, if some parties in the protocol are not trusted.

In our ranking method we prefer to provide a single ranking metric (i.e., utility), which combines different factors, instead of providing several separate metrics, each considers only a single factor. We chose this direction, since it is more straightforward for agents to interpret and use a single ranking, instead of a set of separate rankings. However, in some cases, for an agent a single factor may be more important than the others. For example, an agent that had limited resources to cover the cost of a protocol might take only the cost ranking into account ignoring benefit, while selecting a protocol. Similarly, an agent may prefer to use overall trustfulness of a protocol independent from benefit or more generally independent from utility. For example, an agent that pursues a critical goal might prefer the most trustful protocol, even if the overall utility of the protocol was low. Our method can easily be used to handle such situations. In the former case, the agent can simply ignore the benefit and utility values and take only cost into account. In the later case, our method can be adapted to the situations by considering the benefit of every relevant condition as 1 and cost of every capability as 0.

5. ENACTMENT OF COMMITMENT PROTOCOLS

In the two previous chapters we presented methods to generate and rank commitment protocols. Following our agent process in Figure 1.1, the final step is enactment of one of the generated protocols. Once one of these protocols is enacted by the agents, the commitments of the protocol are created in the running multiagent system and agents start to interact with each other based on these commitments. However, before enactment of a protocol takes place, the agents should first agree on one of the generated protocols to use it for regulating their interaction.

Negotiation is the major method to reach agreement among agents in a multiagent system [30–33]. While negotiating, agents exchange offers in a context until an agreement is reached on one of these offers. In our case, the context of the negotiation is a commitment protocol. Hence, agents offer commitment protocols to each other, until an agreement is reached on one of them. Accordingly, here we adopt a well known monotonic concession procedure [34] to negotiate over commitment protocols. In our method, the leading agent offers the generated protocols to other agents one by one until an agreement on a protocol is reached. Intuitively, the first proposed protocol is the top ranked protocol from the leading agent's point of view. If all agents accept the proposed protocol, then it is enacted. On the other hand, if the proposed protocol is rejected, then the leading agent concedes and offers a protocol that has a lower rank than the rejected protocol.

An important part of negotiation is the deliberation processes that are used by agents in order to negotiate effectively. These are used (i) while deciding on the context of an offer that is going to be made by the agent, and (ii) while deciding on the reply of a received offer. While making these decisions, an agent may take different conditions into account depending on the context of the negotiation. Considering commitment protocols, some of these conditions are the benefit, the cost and the trustworthiness of the protocol, which we cover by defining ranking methods based on these conditions in the previous chapter. Another relevant condition is the *feasibility* of the commitments in a protocol. Commitments of a protocol (or in general a set of commitments) are feasible from an agent's perspective, if the agent is capable of fulfilling all of its responsibilities that emerge due to the commitments in the protocol.

Consider the customer in our running example, who may offer a commitment to the merchant to pay, if the merchant delivers the furniture. In this case, the customer should consider whether she can make the payment once the customer accepts the offer and delivers the furniture, before actually making the offer. Such deliberation is also essential for the merchant, who receives the offer. The merchant should also consider whether she can make the requested delivery in the offer, before accepting it. Accordingly, in the second part of this chapter we present a method to decide on the feasibility of a set of commitments. We first define the feasibility problem and then provide a method that utilizes constraint satisfaction techniques to decide on feasibility.

5.1. Negotiation

We present our monotonic concession procedure in Figure 5.1. This procedure can be executed by the leading agent to negotiate and reach agreement with other agents on the use of a commitment protocol to regulate a future interaction. The leading agent first selects one of the generated protocols, presumably the top ranked one from its perspective. Then, the leading agent proposes the commitments of this protocol to the corresponding debtors. However, before proposing the protocol the leading agent also checks its feasibility with respect to its own responsibilities. If all the debtors accept the proposed commitments, then an agreement is reached over the protocol. Otherwise, if at least one of the debtors rejects one of the proposed commitments, the leading agent concedes by discarding the current protocol and selecting another one from the generated set of protocols, which is ranked next. Then the leading agent offers the selected protocol's commitments to the corresponding debtors. This concession process continues until all agents agree on a protocol or reject all the generated protocols.

Details of our monotonic concession procedure we present in Figure 5.1 is as follows. The procedure requires the list of protocols sorted by a ranking in descending

```
Input: $\Pi$, list of protocols in descending order by rank
Input: $x$, identifier of the leading agent
 1: for all $\pi \in \Pi$ do
 2:   if IsFeasible($x, \pi$) then
 3:     $\mathcal{C} = \emptyset$
 4:     for all $y \in$ GetAgents($p$) do
 5:       $\mathcal{C}' \leftarrow$ GetCommitments($y, \pi$)
 6:       $resp \leftarrow$ Propose($y, \mathcal{C}'$)
 7:       if $resp = Accept$ then
 8:         $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}'$
 9:       else if $resp = Reject$ then
10:         for all $C(y, x, d, r) \in \mathcal{C}$ do
11:           InformCancel($y, C(y, x, d, r)$)
12:         end for
13:         goto 1        /* next protocol ... */
14:       end if
15:     end for
16:     return $\pi$
17:   end if
18: end for
19: return $null$
```

Figure 5.1. Monotonic concession procedure to reach agreement on a commitment protocol.

order (Π) and the identifier of the leading agent (x) as input. The procedure iterates over each feasible protocol π in Π until a protocol is accepted by all corresponding agents. For this purpose, the procedure first checks whether π is feasible from x's perspective using *IsFeasible* function, which we explain in the second part of this chapter. If π is feasible, then the procedure considers each agent y involved in π by proposing the commitments of π in which y is the debtor. In order to do that, the procedure first obtains the list of agents in π using the auxiliary function *GetAgents*. Then for each agent y, it obtains the set of commitments $\mathcal{C}'$ in which y is the debtor using the auxiliary function *GetCommitments* and offers $\mathcal{C}'$ to y using the function *Propose*. The response of y is kept by *resp*. If y accepts the offered commitments (i.e., *resp* = *Accept*), then the commitments are added to the list of accepted commitments $\mathcal{C}$. Finally, if all the debtors accept the offered commitments (i.e., there is agreement on the protocol), then π is returned.

Otherwise, if an offer for a set of commitments is rejected (i.e., $resp = Reject$), then π is discarded and x concedes to the next protocol that is ranked after π. Also note that, once a protocol is rejected, x notifies the debtors of the commitments in $\mathcal{C}$ (i.e., the debtors who already accepted to create a commitment in the context of π) about the cancellation of π via *InformCancel* function. Finally, if the agent offers all the protocols and none of them is accepted, then the procedure returns $Null$, which indicates that there is no agreement on any protocol.

Our negotiation procedure is a variant of the monotonic concession method proposed by Rosenschein and Zlotkin [34]. In their method, negotiation is not led by an agent and accordingly every agent can make an offer. Additionally, while replying offers, agents can also make counter-offers. Different than their procedure, in our case the offers are only made by the leading agent. This is more convenient in our case, since only the leading agent generates the protocols. Besides, the agents who received an offer reply only by accepting or rejecting the proposed commitments. Alternatively, our method can be extended to allow counter-offers. However, if this is the case, the leading agent has to reconsider the counter-offered protocol by taking support, evaluation and feasibility into account. Finally, although our negotiation method is based on the concession concept, this is not the only strategy that can be utilized by agents while negotiating. For example, as an alternative to concession, in the earlier phase of negotiation, an agent that is hoping to reach an agreement quickly may prefer to offer a protocol that is more beneficial to other agents (but less beneficial to itself). In general, various negotiation techniques can be applied at this point to decide on the protocol that will be offered [33].

5.2. Feasibility of Commitments

The method that we present in the previous section provides a way to negotiate over commitment protocols. However, a major challenge in this method is deciding effectively about the feasibility of a protocol before offering it to other agents. Basically, a protocol is feasible from an agent's perspective, if the agent is capable of fulfilling all of its responsibilities that occur due to the protocol's commitments. However, while considering feasibility of a protocol, an agent should not consider only that protocol's commitments. Instead, the agent should take all of its relevant commitments into account, which may exist because of concurrently executing protocols. This is necessary, because those commitments that exist in the concurrently executing protocols may affect each other.

Consider the retailer in our running example. In most of the generated protocols (Table 3.1 in page 42) the retailer has two commitments to the customer, one to deliver the building materials and one to deliver building tools. Moreover, in many realistic settings, the retailer would interact with many customers (and agents in other roles) at the same time in the context of different protocols and accordingly would be committed to them simultaneously to deliver materials and tools (and possibly to do other things). In both cases, the retailer should be able to fulfill all of her commitments simultaneously. In other words all the commitments of the retailer (independent from the protocols they belong) should be feasible. Otherwise, if the retailer's commitments were infeasible, she would inevitably violate some of her commitments.

5.2.1. Conflicts between Commitments

A fundamental indicator of infeasibility among an agent's commitments is a *conflict* between two commitments. Basically, two commitments are in conflict, if fulfillment of one commitment prevents the debtor of the commitment from fulfilling the other commitment (and vice verse). For example, assume that the retailer had only one set of tools. If the retailer would commit simultaneously to two different customers to deliver this set of tools, these commitments of the retailer would conflict (i.e., they would be infeasible). They would conflict, since the retailer could not deliver the same set of tools to two different customers and accordingly could not fulfill one commitment without violating the other. If the retailer would deliver the tools to the first customer to fulfill the corresponding commitment, then it would be impossible for the retailer to fulfill her commitment to the second customer. Similarly, if the retailer would deliver the tools to the second customer to fulfill the corresponding commitment, then it would be impossible for the retailer to fulfill her commitment to the first customer.

Accordingly, here we present two fundamental cases that define conflict situations between two commitments. We show these cases as if-then rules that can be used by agents as a reasoning mechanism to capture conflicts in their commitments. In these rules we use the following meta-predicates:

- $Inconsistent(u, u')$: The conditions u and u' are inconsistent with each other, hence at a given time either only u or only u' hold.
- $Consistent(u, u')$: The conditions u and u' are consistent with each other, hence u and u' may hold at the same time.
- $Conflict(c_i, c_j)$: The commitments c_i and c_j are in conflict.
- $ConditionalConflict(c_i, c_j)$: The commitments c_i and c_j are in conditional conflict.

We assume that consistency relations of conditions (i.e., *Inconsistent* and *Consistent* meta-predicates) are part of the agent's domain knowledge and shared among all agents. This can be achieved by using a domain ontology as it is customary in the literature [35]. Here, we assume that given a domain $\mathcal{D}$, if there exists an inconsistency between two conditions in $\mathcal{D}$ (e.g., $Inconsistent(u, u') \in \mathcal{D}$), then in $\mathcal{D}$ there is no statement about the consistency of these two conditions (e.g., $Inconsistent(u, u') \notin \mathcal{D}$) and vice verse.

Note that while capturing conflicts between commitments, states of the commitment should also be taken into account. Intuitively, only active and conditional commitments can be in conflict. On the other hand, an already fulfilled or violated commitment does not conflict with other commitments, since that commitment is in a terminal state.

For example, suppose that the retailer is committed to two customers Cus_1 and Cus_2 as follows: $c_1 = C(Ret, Cus_1, \top, Cus_1HaveTools)$ and $c_2 = C(Ret, Cus_2, \top, Cus_2HaveTools)$. Now consider two cases. In the first case, the antecedents of both commitments hold already (i.e., *top*), but neither $Cus_1HaveTool$ nor $Cus_1HaveTools$ holds. In this case, if the retailer had only a single set of tools to deliver, then the commitments would conflict, since fulfillment of one of the commitments would consume the tools and the retailer would not be able to fulfill the other commitment. In the second case, beside the antecedents of both commitments, $Cus_1HaveTool$ also holds initially. Accordingly, the first commitment is already fulfilled. In this case, the first commitment does not have any effect on the fulfillment of the second commitment, since it is already fulfilled. Therefore, the fulfillment or violation of the second commitment is independent from the first commitment. Accordingly, we cannot consider any conflict between theses two commitments.

Accordingly, the rules below apply only to those commitments, which are active or conditional. In order to differentiate active and conditional commitments, for readability, below we use $\top$ symbol as the antecedent of active commitments (i.e., the commitment's antecedent is already satisfied). We also use _ instead of agent identifiers, when the identity of an agent is irrelevant.

Definition 5.1. *(Commitment Conflict) Given a set of commitments $\mathcal{C}$ and a domain $\mathcal{D}$, if there are two active commitments $c_i = C(\_, \_, \top, u)$ and $c_j = C(\_, \_, \top, u')$ in $\mathcal{C}$ and the consequences of these commitments are inconsistent with respect to $\mathcal{D}$, then c_i and c_j are in* conflict.

$$\frac{C(\_, \_, \top, u) \in \mathcal{C} \text{ and } C(\_, \_, \top, u') \in \mathcal{C} \text{ and } Inconsistent(u, u') \in \mathcal{D}}{Conflict(C(\_, \_, \top, u), C(\_, \_, \top, u'))}$$

Assume that the retailer is committed to two customers Cus_1 and Cus_2 as follows: $c_1 = C(Ret, Cus_1, \top, Cus_1HaveTools)$ and $c_2 = C(Ret, Cus_2, \top, Cus_2HaveTools)$. In this situation, the retailer is committed to the first customer to deliver the tools (i.e., retailer should satisfy $Cus_1HaveTools$) because of her first commitment c_1. Similarly, because of her second commitment c_2 the retailer should deliver the tools also to the second customer (i.e., retailer should satisfy $Cus_2HaveTools$). Assuming that the retailer has only one set of tools in stock, it is the case that $Inconsistent(Cus_1Have\text{-}Tools, Cus_2HaveTools)$, since the retailer cannot deliver the same set of tools to two different customers. Accordingly, using the rule in Definition 5.1 we can conclude that

c_1 and c_2 conflict (i.e., $Conflict(c_i, c_j)$).

A conflict as specified in Definition 5.1 indicates an inevitable violation of at least one of the conflicting commitments. That is the debtor of the commitments is obligated to fulfill both commitments, since they are active (i.e., commitments' antecedents are already satisfied), but is unable to fulfill both in the given domain.

Note that in Definition 5.1 the debtors and the creditors of the commitments are actually irrelevant, since the commitments conflict anyway, if their consequences cannot be brought about simultaneously, independent from the corresponding agents.

Definition 5.2. *(Conditional Commitment Conflict) Given a set of commitments $\mathcal{C}$ and a domain $\mathcal{D}$, if there is an active commitment $c_i = C(\_, \_, \top, u)$ and a conditional commitment $c_j = C(\_, \_, \theta', u')$ in $\mathcal{C}$, and the consequences of these commitments are inconsistent with respect to $\mathcal{D}$, then c_i and c_j are in* conditional *conflict.*

$$\frac{C(\_, \_, \top, u) \in \mathcal{C} \; and \; C(\_, \_, \theta', u') \in \mathcal{C} \; and \; Inconsistent(u, u') \in \mathcal{D}}{ConditionalConflict(C(\_, \_, \top, u), C(\_, \_, \theta', u'))}$$

Given a set of commitments $\mathcal{C}$ and a domain $\mathcal{D}$, if there are two conditional commitment $c_i = C(\_, \_, \theta, u)$ and $c_j = C(\_, \_, \theta', u')$ in $\mathcal{C}$ and the consequences of these commitments are inconsistent and the antecedents of these commitments are consistent with respect to $\mathcal{D}$, then c_i and c_j are in conditional conflict.

$$\frac{\begin{array}{c} C(\_, \_, \theta, u) \in \mathcal{C} \; and \; C(\_, \_, \theta', u')\mathcal{C} \; and \; Inconsistent(r, r')\mathcal{D} \; and \\ \forall q, q' : q \in \theta \; and \; q' \in \theta' \; it \; is \; the \; case \; that \; Consistent(q, q') \in \mathcal{D} \end{array}}{ConditionalConflict(c_i, c_j)}$$

Different than the Definition 5.1, in Definition 5.2 one or both of the commitments are conditional. Accordingly, occurrence of a conflict between two such commitments depends on the satisfaction of the commitments' antecedents. For example, suppose that the retailer is committed to two customers Cus_1 and Cus_2 as follows: $c_1 = C(Ret, Cus_1, Cus_1ToolsPaid, Cus_1HaveTools)$ and $c_2 = C(Ret, Cus_2, Cus_2ToolsPaid, Cus_2HaveTools)$. In this situation, only if the first customer pays for the tools (i.e., first customer should satisfy $Cus_1ToolsPaid$), the retailer will be committed to the first customer to deliver the tools because of c_1. Similarly, only if the second customer pays for the tools (i.e., second customer should satisfy $Cus_2ToolsPaid$), the retailer will be committed to the second customer to deliver the tools because of c_2. In this case, even tough the consequences of c_1 and c_2 are inconsistent (assuming that the retailer has only one set of tools in stock), only if both customers make the corresponding payments and satisfy antecedents of c_1 and c_2 (i.e., both commitments turn to active and Definition 5.1 applies), c_1 and c_2 will conflict. Otherwise, if only one customer or none of them made the payment, commitments would not be in conflict, since at least one of the commitments would be conditional. Moreover, if the antecedents of the commitments are inconsistent as specified in the second part of Definition 5.2, then these commitments are not in conflict, since the antecedents cannot be satisfied concurrently. Accordingly, only one of these commitments can be active at a time and consequently, the debtor is under the obligation of one commitment at a time. In conclusion, a conditional commitment conflict indicates possibility of a commitment conflict between two commitments (as specified in Definition 5.1). But, occurrence of the actual conflict depends on the satisfiability of the commitments' antecedent.

5.2.2. From Conflicts to Feasibility

Although commitment conflicts that we defined in the previous section capture fundamental cases that cause to infeasibility among commitments, they are limited because of the following two reasons:

- Conflict definitions do not allow us to effectively represent and reason about commitments' *resource requirements.*
- Conflict definitions do not take *temporal constraints* over commitments, such as deadlines, into account.

In the real world, agents spend resources when performing their capabilities. For example, in order to be able to make a payment an agent should have enough money (i.e., resource). Hence, while considering feasibility of an agent's commitments, we should take the resources that are needed by the agent in order to fulfill its commitments into account. The conflict definitions in the previous section represent resource requirements abstractly over *consistent* and *inconsistent* meta-predicates. For example, if the merchant was committed to two customers to deliver some furniture and the merchant owned only one piece of furniture, then we would say that the corresponding conditions of these two commitments are inconsistent.

It is clear that this abstract representation is ineffective for practical cases. First of all, this representation requires revision of consistent and inconsistent meta-predicates for every change in the availability of resources, which is inefficient once we consider quantitative resource types such as money. Moreover, this representation allows us to capture conflicts (and respectively infeasibility) that occur only between pairs of commitments. However, an infeasibility may still occur among three commitments, even though there is no conflict when the commitments are considered in pairs. For example, the merchant might have two pieces of furniture and committed to three customers to deliver the furniture. In this case there would be no conflict between commitment pairs, but there would be an infeasibility among the three commitments. In order to capture such situations, we may try to extend our conflict definitions considering ternary relations. However, since an infeasibility may occur among any number of commitments, this approach fails.

Other than the resource requirements, in many practical settings commitments are also constrained by deadlines or other temporal constraints. Although in our previous work we show that it is possible to capture conflicts between two commitments that occur due to their temporal constraints, using event calculus formalism [36], this approach is also inefficient to determine feasibility of more than two commitments, as in the case of conflicts that occur because of commitments' resources requirements.

Accordingly, we extend our feasibility definition as follows: given a set of commitments of an agent (either debtor or creditor), the commitments are feasible from that agent's perspective, if the agent can satisfy all the conditions (either antecedent or consequent) that it is responsible for with respect to these commitments, respecting the resource requirements of conditions and the temporal constraints of the commitments. Therefore, in order to effectively decide on the feasibility of an agent's commitments, (i) we need a representation that can reflect resource requirements and temporal constraints of commitments, and (ii) we need a method to that can efficiently decide on feasibility, taking commitments' resource requirements and temporal constraints into account.

5.3. Extended Technical Framework

As we discussed above, in order to capture feasibility, in our framework we should represent temporal and resource constraints over commitments. Besides, different than the two previous chapters, here we should deal with the actual execution of the system to be able to capture when an agent takes an action and how resources are exchanged with respect to an action. Accordingly, first we extend our technical framework that we presented in Section 2.2 with new elements that we use while computing feasibility of an agents' commitments. Basically, we define a state based discrete execution framework in which each state is associated with an integer time value.

We start with temporal properties, which we use to specify a temporal constraint over the satisfaction of a conditions. Basically, we associate a time interval with a condition and if the condition is satisfied within this time interval, then we say that the property is satisfied. Otherwise we call the property as failed. Below, we use variables p, p_i, p' over properties and t, t_i, t' over time moments.

Definition 5.3. *(Temporal Property) $P(t_s, t_e, u)^\mu$ denotes a* temporal property*, in which t_s and t_e are two integers that define a time interval, u is a condition and μ is the state of the property, which can take one of the values True ($\top$), False ($\bot$) or Undetermined (∇).*

A property $P(t_s, t_e, \theta)^\mu$ states that the condition u should be satisfied in a moment between t_s and t_e, inclusively. Given a property, if property's condition is satisfied with respect to the specified temporal constraint, then the state of the property is *satisfied* ($\top$). If the condition is not satisfied with respect to the temporal constraint, then property's state is *failed* ($\perp$). Finally, if it is not possible to determine whether the property is satisfied or not, yet, then the property's state is *undetermined* (∇). Satisfied and failed states of properties are terminal states. For any property the initial moment of the property's time interval is less or equal to the last moment of the property's time interval. That is, if t_s is the initial moment and t_e is the last moment of the property's time interval, then $t_s < t_e$. Also note that in order to satisfy a property, the property's condition should be satisfied within the given time interval. For example, if there is a property, such that a furniture should be delivered to the customer between Tuesday and Thursday, then the property is satisfied only if the furniture is actually delivered at one of the specified days. Otherwise, if the furniture is delivered on Monday, even tough the condition is satisfied earlier than specified and it holds when the time interval starts, the property is not satisfied.

In most of the real world situations, some resources are required to satisfy certain conditions (e.g., money is required to make a payment). Such situations can be represented in an ontology by associating conditions to resources that are required to satisfy the condition. Such a representation can also be extended by including what happens to a resource when a condition is satisfied (e.g., consumed, transferred, etc.). Beside resource requirement, satisfaction of a condition usually does not happen instantly. That is in order to satisfy a condition an agent should perform some actions (i.e., utilize its capabilities), each take some amount of time. This can also be represented in an ontology by associating the condition with the required amount of time to satisfy it. In the rest of this chapter we assume that agents have a common domain knowledge about how resources are affected by satisfaction of conditions. They are also aware of their own resources and keep track of them. Besides, each agent knows how long it takes to satisfy a condition. For readability, we do no explicitly show these details, when they are clear.

Now, we define a commitment using temporal properties by replacing the antecedent from a set of conditions to a set of properties and the consequent from a condition to a property.

Definition 5.4. *(Commitment) $C(x, y, \mathcal{P}_{ant}, p_{con})^{\nu}$ denotes the* commitment *from the debtor agent x to the creditor agent y to satisfy the consequent property p_{con}, if every property in the antecedent $\mathcal{P}_{ant}$ is satisfied. ν is the state of the commitment, which can take one of the values Conditional (C), Active (A), Discharged (D), Violated (V) or Expired (E).*

A commitment is created in *conditional* state (C). If the antecedent is satisfied while the commitment is conditional, then the commitment is *active* (A) and if the antecedent is failed to be satisfied while the commitment is conditional, then the commitment is *expired* (E). If the consequent is failed to be satisfied while the commitment is active, then the commitment is *violated* (V) and if the consequent is satisfied, while the commitment is active, then the commitment is *fulfilled* (F). Expired, fulfilled and violated commitments are in terminal states. Given a commitment, we assume that there is a total order between each property in the antecedent and the consequent property. That is, for all antecedent properties in a commitment, if t^e_{ant} is the last moment of the time interval of antecedent property and t^s_{con} is the first moment of the time interval of the commitment's consequent property, then $t^e_{ant} < t^s_{con}$.

Since our main objective is to decide on the feasibility of an agent's commitments, we have to capture how the state of these commitments may evolve with respect to the progression of the multiagent system. That is we have to examine how the agents in the multiagent system may behave and accordingly how the state of the commitment may change. Accordingly, here we define a state based discrete execution framework. In this framework, state transitions occur due to events that happen in the multiagent system and affect the state of the commitments. We start with an *agent state* that specifies the states of conditions, properties and commitments in a given moment of an execution. Note that as we do in the previous chapter, we consider an execution state from an agent's perspective. That is the agent is aware of only a subset of conditions, properties and commitments in a system and it assesses the states of these elements with respect to its own beliefs. Hence, an agent state may not represent the whole state of the system and may even be different than the actual system state if the agent's beliefs are wrong. On the other hand, since our aim is to capture feasibility of a set of commitments from an agent's point of view, such a state definition suits well to our purposes. This approach is also more realistic than assuming complete knowledge about the system, since this is rarely the case in a multiagent system.

Definition 5.5. *(Agent State) An* agent state *η is a three-tuple $\eta = \langle \theta, \mathcal{P}, \mathcal{C} \rangle$ in which θ is the set of conditions, which hold in η, $\mathcal{P}$ is the set of properties and $\mathcal{C}$ is the set of commitments in η.*

Finally, we define an execution using a set of ordered agent states.

Definition 5.6. *(Execution) An* execution *e is an ordered set of agent states $e = \langle \eta_0, \eta_1, \eta_2, \ldots, \eta_n \rangle$ where each execution state η_i is associated with a unique moment starting from* 0 *up to n.*

An execution starts at moment 0 from an initial agent state η_0. In an execution, the moment t is exactly before $t + 1$. Hence, η_t is the agent state that is exactly before η_{t+1}. An execution ends up at moment n in the final agent state η_n where all commitments are in a terminal state. In the rest of the thesis we use θ_t, $\mathcal{P}_t$ and $\mathcal{C}_t$ to refer to the corresponding sets in the agent state η_t (e.g., $\theta \in \eta_t \equiv \theta_t$).

5.3.1. State Transition

The transition from an agent state η_{t-1} to η_t is a three step process. First, the condition that holds in η_t is determined by a function Δ^θ. Δ^θ is a domain dependent function that takes θ_{t-1} and arbitrary number of other parameters (e.g., agent percepts, events etc.) that represent how the real multiagent system evolves, in order to determine which conditions hold in η_t using a set of domain dependent rules. Then, those holding conditions are put into θ_t by the Δ^θ function. In the rest of this thesis we assume that such a function is defined and available with respect to the agent's domain knowledge [3]. After the assignment of the states of the conditions by θ_t a second function $\Delta^{\mathcal{P}}$, which is a function of $\mathcal{P}_{t-1}$ and θ_t, is executed to assign the states of the properties in $\mathcal{P}_t$. Finally, a third function $\Delta^{\mathcal{C}}$, which is a function of $\mathcal{C}_{t-1}$ and $\mathcal{P}_t$, is executed to assign the states of the commitments in $\mathcal{C}_t$. Below, we define the functions $\Delta^{\mathcal{P}}$ and $\Delta^{\mathcal{C}}$ in Definitions 5.7 and 5.8, respectively.

Definition 5.7. *(Assignment of Property States) The function $\Delta^{\mathcal{P}}(\mathcal{P}_{t-1}, \theta_t)$ assigns the states to the properties in $\mathcal{P}_t$ using the states of the properties in the previous agent state η_{t-1} given by $\mathcal{P}_{t-1}$ and the updated states of the conditions in θ_t. The $\Delta^{\mathcal{P}}$ function operates using the following rules:*

- *Undetermined: An undetermined property $P(t_s, t_e, u)^{\nabla}$ in $\mathcal{P}_{t-1}$ continues to stay in the undetermined state in $\mathcal{P}_t$, if either the current moment t is before the time interval of the property (i.e., $t < t_s$) or t is within the time interval of the property but the condition u is not satisfied. Formally:*

$$\frac{P(t_s, t_e, u)^{\nabla} \in \mathcal{P}_{t-1} \ and\ ((t < t_s)\ or\ (t_s \leq t \leq t_e\ and\ u^{t_i} \notin \theta_t))}{P(t_s, t_e, u)^{\nabla} \in \mathcal{P}_t}$$

- *True: An undetermined property $P(t_s, t_e, u)^{\nabla}$ in $\mathcal{P}_{t-1}$ is satisfied $\mathcal{P}_t$, if the current moment t is within the time interval of the property and u holds in the current state. Formally:*

$$\frac{P(t_s, t_e, u)^{\nabla} \in \mathcal{P}_{t-1}\ and\ t_s \leq t \leq t_e\ and\ u^{t_i} \in \theta_t\ and\ t = t_i}{P(t_s, t_e, u)^{\top} \in \mathcal{P}_t}$$

- *False: An undetermined property $P(t_s, t_e, u)^{\nabla}$ in $\mathcal{P}_{t-1}$ fails in $\mathcal{P}_t$, if the current moment t is later then the time interval of the property. This means that u is not satisfied within the time interval of the property. Formally:*

$$\frac{P(t_s, t_e, u)^{\nabla} \in \mathcal{P}_{t-1}\ and\ t_e < t}{P(t_s, t_e, u)^{\perp} \in \mathcal{P}_t}$$

- *Inertia: The following two rules state that once a property is in a terminal state (i.e., Satisfied or Failed) then this state is preserved.*

$$\frac{P(t_s, t_e, u)^{\top} \in \mathcal{P}_{t-1}}{P(t_s, t_e, u)^{\top} \in \mathcal{P}_t} \quad \frac{P(t_s, t_e, u)^{\perp} \in \mathcal{P}_{t-1}}{P(t_s, t_e, u)^{\perp} \in \mathcal{P}_t}$$

Definition 5.8. *(Assignment of Commitment States) The function $\Delta^{\mathcal{C}}(\mathcal{C}_{t-1}, \mathcal{P}_t)$ assigns the states of the commitments in $\mathcal{C}_t$ using the states of the commitments in the previous agent state η_{t-1} given by $\mathcal{C}_{t-1}$ and the updated states of the properties in $\mathcal{P}_t$. The $\Delta^{\mathcal{C}}$ function operates using the following rules. For readability, below we abuse our notation slightly with respect to denoting property states. We use the notation $p^{\top}$ to mean that the property p is in state $\top$. The same applies for the other two property states.*

- *Active: A conditional commitment becomes active, if every property in its antecedent is satisfied.*

$$\frac{C(x, y, \mathcal{P}, p_{cons})^{C} \in \mathcal{C}_{t-1} \textit{ and } \forall p_{ant} \in \mathcal{P} : p_{ant}^{\top} \in \mathcal{P}_t}{C(x, y, \mathcal{P}, p_{cons})^{A} \in \mathcal{C}_t}$$

- *Expired: A conditional commitment becomes expired, if there exists a failed property in its antecedent.*

$$\frac{C(x, y, \mathcal{P}, p_{cons})^{C} \in \mathcal{C}_{t-1} \textit{ and } \exists p_{ant}^{\perp} \in \mathcal{P}_t}{C(x, y, \mathcal{P}, p_{cons})^{E} \in \mathcal{C}_t}$$

- *Fulfilled: An active commitment becomes discharged, if its consequent property is satisfied.*

$$\frac{C(x, y, \mathcal{P}, p_{cons})^{A} \in \mathcal{C}_{t-1} \textit{ and } p_{cons}^{\top} \in \mathcal{P}_t}{C(x, y, \mathcal{P}, p_{cons})^{F} \in \mathcal{C}_t}$$

- *Violated: An active commitment becomes violated, if its consequent property is* *failed.*

$$\frac{C(x,y,\mathcal{P},p_{cons})^A \in \mathcal{C}_{t-1} \; and \, p_{cons}^{\perp} \in \mathcal{P}_t}{C(x,y,\mathcal{P},p_{cons})^V \in \mathcal{C}_t}$$

- *Inertia: A commitment that is either fulfilled, violated or expired cannot move to another state.*

$$\frac{C(x,y,\mathcal{P},p_{cons})^F \in \mathcal{C}_{t-1}}{C(x,y,\mathcal{P},p_{cons})^F \in \mathcal{C}_t} \quad \frac{C(x,y,\mathcal{P},p_{cons})^V \in \mathcal{C}_{t-1}}{C(x,y,\mathcal{P},p_{cons})^V \in \mathcal{C}_t} \quad \frac{C(x,y,\mathcal{P},p_{cons})^E \in \mathcal{C}_{t-1}}{C(x,y,\mathcal{P},p_{cons})^E \in \mathcal{C}_t}$$

Below we present two consistency properties of our framework. Lemmas 5.9 and 5.10 state that in any given agent state, every property and every commitment must have a state value and neither a property nor a commitment can be in two different states at the same time, respectively.

Lemma 5.9. *(State Uniqueness of Properties) If a property $P(t_s, t_e, u)^{\mu}$ is in $\mathcal{P}$ of an agent state η, then μ can have only of the values satisfied, failed or undetermined.* ■

Proof. Follows trivially from Definition 5.7. □

Lemma 5.10. *If a commitment (State uniqueness of commitments) $C(x,y,\mathcal{P},p_{con})^{\nu}$ is in $\mathcal{C}$ of an agent state η, then ν can have only of the values conditional, active, discharged, violated or expired.* ■

Proof. Follows trivially from Definition 5.8. □

5.3.2. Possible Executions

Here we define how an agent can examine progression of the system in our technical framework by using possible executions. A *possible execution* is any execution from a given agent state s_0 to a final agent state s_n, which is generated by manipulating the state of conditions and applying the state transition process that is defined in Section 2.2 at each intermediary agent state s_t, such as $0 < t < n$.

A possible execution is a hypothetical progression of the system based on the knowledge of its current agent state. The key issue while creating a possible execution is the manipulation of the condition states, which should be done based on the responsibilities of the agents as specified by the commitments. That is, if a condition is the consequent (antecedent) property of a commitment, then the debtor (creditor) of the commitment is responsible for the condition. When the responsible agent of a condition is the leading agent (i.e., the agent that examines the system), manipulation of the conditions should be based on the availability of resources to the leading agent that are required to satisfy these conditions. For instance, suppose that there is a condition about a payment of 10 Dollars, which is a responsibility of the leading agent. This condition can be set to satisfied in a state of a possible execution, only if the leading agent has at least 10 Dollars at that state. On the other hand, if the a condition's responsible agent is not the leading agent, then the state of the condition can be set to any valid value, since the behavior of the other agents are not known in advance.

Note that by systematically considering every possible combination of the conditions' states that are given in an agent state, we can generate every possible state that can be accessed from the given agent state. By repeating this process for every generated state, we can generate all possible executions that can initiate from the given agent state. For instance, considering our running example, suppose that in the current agent state η_0 the merchant is committed to the customer to deliver the furniture at state 2, if the merchant is paid in state 1. More precisely, we have $C(Mer, Cus, P(1, 1, FurniturePaid)^{\nabla}, P(2, 2, HaveFurniture)^{\nabla})^{C}$. Remember that the customer is the leading agent and suppose that the conditions *FurniturePaid* and *HaveFurniture* do not hold in the current state. Then there are three possible executions as follows:

- e_1 : Cus satisfies $FurniturePaid$ at 1 and makes the commitment active. Mer satisfies $HaveFurniture$ at 2 and makes the commitment fulfilled.
- e_2 : Cus satisfies $FurniturePaid$ at 1 and makes the commitment active. Mer

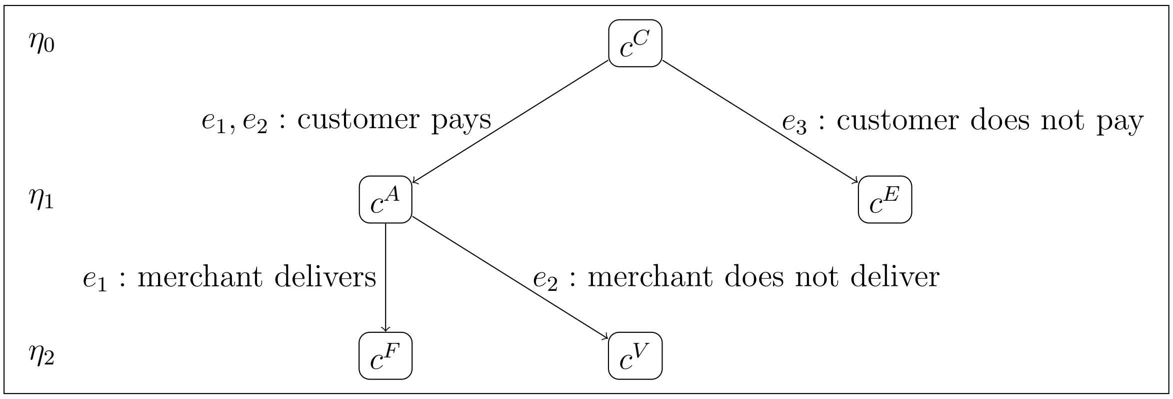
η_0
η_1
η_2
c^C
c^A
c^E
c^F
c^V
e_1, e_2 : customer pays
e_3 : customer does not pay
e_1 : merchant delivers
e_2 : merchant does not deliver

Figure 5.2. Possible executions.

does not satisfy $HaveFurniture$ at 2 and violates the commitment.

- e_3 : Cus fails to satisfy $FurniturePaid$ at 1 and the commitment expires.

Note that $HaveFurniture$ may or may not hold depending on the behavior of Mer. Hence, we should take both cases into account while considering the possible executions. On the other hand, if we know that Cus (i.e., the leading agent) does not have enough resources to satisfy $FurniturePaid$, then we should neither consider e_1 nor e_2 as a possible execution, since we know that $FurniturePaid$ can never be satisfied.

5.3.3. Feasibility of Commitments

Now we are ready to formally define feasibility of a set of commitments from an leading agent's perspective using the notion of possible executions.

Definition 5.11. *(Feasibility of Commitments) In a given agent state η_t, the set of commitments $\mathcal{C} \subseteq \mathcal{C}_t$ of an agent x, such that $\mathcal{C} = \{c \mid C(x, \_, \_, \_)^{A|C}$ or $C(\_, x, \_, \_)^{A|C}\}$ is* feasible, *if there exists a possible execution e that starts from η_t and there exists a state $\eta_{t'} \in e$, such that $t < t'$ and every commitment $c \in \mathcal{C}$ is in fulfilled state in $\mathcal{C}_{t'}$.*

Definition 5.11 states that if starting from the given state, there exists at least one possible execution in which every initially active or conditional commitment of the leading agent is fulfilled, then the commitments of the leading agent are feasible. Note that feasibility does not mean that the considered commitments are going to be definitely fulfilled by their debtors. A commitment—by its nature—may not be fulfilled, even if its debtor has enough resources to fulfill it. For instance, the merchant in the previous examples might simply decide not to deliver for some reason even if it had the furniture (i.e., even if its commitments were feasible) and would violate its commitment. Conversely, if the commitments of an leading agent are not feasible, then no matter what the agent does (or the other agents do), it is not possible to fulfill all of the leading agent's commitments (i.e., there does not exist a possible execution in which all the commitments are discharged). For instance, the merchant's commitments to a set of customers might require her to deliver ten pieces of furniture in total while the merchant had only five pieces of furniture. In this case, the merchant would be over-committed (i.e., the commitments would not be feasible) and there would be no way to fulfill them. To deal with this, a merchant may try to discover other ways to create resources through other agents services in the system by creating new commitments [26].

Unfortunately, the formulation of feasibility problem in Definition 5.11 is NP-complete. We can show this by transforming the well known SequencingWithIntervals problem into a feasibility problem [37]. In fact, if we ignore resource requirements, our problem is identical to SequencingWithIntervals problem.

Theorem 5.12. *(Complexity) Let us call the decision problem we specify in Definition 5.11 as* Feasibility*. Then,* Feasibility *is NP-complete.*

Proof Sketch: We can show NP-completeness of Feasibility by transforming SequencingWithIntervals into Feasibility. The complete transformation is on page 152 in Appendix B.2. □

5.4. Computing Feasibility of Commitments

The easiest way to compute feasibility of a set of commitments is to enumerate every possible execution and check whether there exists a possible execution in which all the commitments are fulfilled. However, since computing feasibility of a set of commitments is an NP-complete problem, such a naive approach would fail even for considerably small set of commitments. Even though the NP-completeness result implies that there does not exist an efficient algorithm to solve every instance of this problem, if we use some details of this problem, such as the restrictions imposed by resource requirements and temporal constraints, to guide the enumeration process to omit non-promising executions, we can still solve many practical problems in acceptable time limits.

Accordingly, in this section, we propose a constraint satisfaction based method to compute feasibility of an agent's commitments. In our method we transform computation of commitments' feasibility into a constraint satisfaction problem and use constraint solving techniques to come up with a conclusion efficiently. A constraint satisfaction problem (CSP) consists of a set of variables and a set of constraints over the values of these variables [38]. Each variable has a domain of possible values. Constraints define restrictions over the possible values of the variables with respect to the assigned values to the other variables. An assignment is a state of a CSP, in which some or all of the variables have assigned values. An assignment is called consistent, if the assigned values of the variables do not violate any constraint and is called complete, if a value is assigned to every variable. An assignment that is consistent and complete is called a solution. Various techniques exist to solve CSPs [38] and some of them are implemented in CSP solvers elegantly [39]. Our aim is to use these constraint solving techniques in order to determine feasibility of an leading agent's commitments. In order to achieve this, we first create a CSP instance using the information embodied by the commitments in which the leading agent participates. Then we try to solve the created CSP instance using a constraint solver. Finally, depending on the existence of a solution to the created CSP instance, we decide on the feasibility of the leading agent's commitments.

```
Input: $\eta_t$, an agent state
Input: $x$, the leading agent
1: $\mathcal{N} \leftarrow$ CreateCSPInstance($\mathcal{C}_t, \theta_t, x, t$)
2: $soln \leftarrow$ ConstraintSolver($\mathcal{N}$)
3: if $soln = Failure$ then
4:     return $True$
5: else
6:     return $False$
7: end if
```

Figure 5.3. The CSP based CheckFeasibility algorithm to decide on the feasibility of a set of commitments.

We present this procedure as an algorithm in Figure 5.3. We call this algorithm CheckFeasibility. The algorithm takes an agent state η_t and the identifier of the leading agent as input and returns $True$, if the commitments in $\mathcal{C}_t$ in which the leading agent is either the debtor or creditor, can be fulfilled (i.e., feasible) and $False$, otherwise. The algorithm first creates the CSP instance $\mathcal{N}$ using the function *CreateCSPInstance*, which takes the set of commitments $\mathcal{C}_t$, the set of conditions θ_t, the leading agent x and the initial moment t as input. We define the details of how the CSP instance is created using this input below. Then the algorithm calls a constraint solver to solve the created CSP instance $\mathcal{N}$ and stores the result in *soln*. Our algorithm is independent from a specific constraint solver, hence any constraint solver can be adopted (assuming that the specific constraint solver is able to parse the created CSP instance). If *soln* is equal to the constant $Failure$, which indicates that there is no solution to $\mathcal{N}$, the algorithm concludes that the considered commitments are not feasible and returns $False$. On the other hand, if there is a solution of $\mathcal{N}$, then the algorithm concludes that the considered commitments are feasible and returns $True$.

Now we show how a CSP instance can be created for our CheckFeasibility algorithm in a two step process. Note that we do not present the formal details of this process here for readability. Full details of this process is available in Appendix A. The first step is to define the CSP variables and their domains. Since we define the state transitions of a commitment over its antecedent and consequent, feasibility of a set of commitments depend on the satisfiability of their antecedent and the consequent by the responsible agents (i.e., is there a possible execution in which all properties are satisfied and hence all commitments are fulfilled). Technically, a property can be satisfied by an agent, if the following conditions hold:

- The agent is capable of satisfying the condition of the property.
- Assuming that satisfaction of each condition requires a certain amount of time and the agent can execute one capability at a time, the time interval of the property is long enough to satisfy the condition.
- The agent has enough resources that are required to satisfy the condition of the property.

For the first case we assume that an agent participates in a commitment only if it has all the capabilities that are necessary to satisfy the properties that the agent is responsible for. For the second case, we have to take the temporal constraint of the property into account and show that the agent can satisfy the condition with respect to these constraints. Accordingly, our first two sets of CSP variables capture the moments at which the agent starts to use a capability to satisfy a property's condition and when the condition is actually satisfied, taking the time required to perform the corresponding capability. Considering our definition of possible executions, these moments correspond to the indices of the agent states in a possible execution.

More specifically, given a set of commitments $\mathcal{C}$, for each property p_i that is under the responsibility of the leading agent in the context of a commitment (either the antecedent or consequent) we create two variables ϕ_i^s and ϕ_i^e. A value that is assigned to a ϕ_i^s variable represents the moment that the agent starts to use a capability to satisfy the corresponding p_i's condition. On the other hand, a value that is assigned to a ϕ_i^e represents the moment that the agent finishes to use its capability and actually satisfies the corresponding p_i's condition. For example, suppose that there is a property $p_i = P(3, 7, u)^\nabla$ and $\tau^u = 2$ (i.e., two units of time is required to satisfy the condition u). In this case, the domain of ϕ_i^s is $[2, 6]$ and the domain of ϕ_i^e is $[3, 7]$. Hence, the leading agent can start to use a service as early as moment 2 (i.e., $\phi_i^s = 2$) and satisfy the condition in 3 (i.e., $\phi_i^e = 3$) utilizing moments 2 and 3 to satisfy u. On the other hand, the leading agent can start to use a service at moment 6 at the latest (i.e., $\phi_i^s = 6$) and satisfy the condition in 7 (i.e., $\phi_i^e = 7$) utilizing moments 6 and 7 to satisfy u.

ϕ^s and ϕ^e variables capture the temporal constraints and when properties are satisfied in a possible execution. However, in order to satisfy a property, the responsible agent should have the required resources at that moment as we specified above. Therefore, we should also keep track of the available resources to the agents and validate that the agent has enough resources while assigning a value to a ϕ^s variable. For this purpose, we consider the leading agent and the other agents separately as before. In order to keep track of the leading agent's resources we define a separate variable for each resource type (e.g., money) at each moment. More specifically, for each resource type r and moment t there is a variable ψ_t^r with the domain $[0, \infty]$, which represents the amount of resource r that is available to the leading agent at t. For instance, $\psi_3^{\$} = 10$ means that the leading agent has 10 Dollars at moment 3.

While considering the satisfaction times of the properties, we have to take into account the properties for which the other agents are responsible for, beside the properties that the leading agent is responsible for. Although, the leading agent does not know in advance when the other agents fulfill their responsibilities, we assume that such properties are always satisfied at the earliest moment of their time interval. In other words, given a property $p_i = P(t_s, t_e, u)^{\nabla}$ for which an agent other than the leading agent is responsible, we assume that this property is satisfied at t_s. Hence, we represent the acquired amount of resources as constant values. The idea of this assumption is that the best case to acquire a resource is to acquire it as early as possible. Note that this assumption does not affect the result of the CheckFeasibility algorithm, since (i) if a set of commitments were infeasible, even though the leading agent acquired every resource as early as possible, then the commitments would be infeasible in every other case (i.e., it does not matter when the resources were acquired), and (ii) if a set of commitments were indeed feasible, when the leading agent acquired every resource as early as possible, then it is not necessary to consider the other cases, since a single case is enough to decide on feasibility. A similar situation holds also for resources.

That is, while the leading agent has complete knowledge about its own resources, it usually does not have such a knowledge about the other agents resources. Hence, to decide on feasibility it is necessary to make assumptions also about the other agents' resources. In this context, we make the general assumption that the other agents have unlimited amount of resources and can satisfy any property that they are responsible for. Note that these assumptions are based on the fact that the leading agent does not have any knowledge about the behavior and resources of other agents. We discuss how additional knowledge (if available) about the other agents' behavior and resources can be used while deciding on feasibility in Section 5.5.

The second step while creating the CSP instance is to define the constraints over the variables in order to restrict their domains with respect to values assigned to other variables. In this way we can represent the relations between various variables. In our setting, both the values of ψ_t^r and ϕ_i^e variables depend on the assigned values to ϕ_i^s variables via constraints. For instance, suppose that there is a property that requires the leading agent to spend 10 Dollars and two time units are required to satisfy the property. If 3 is assigned to the corresponding ϕ_i^s variable, then 10 should be subtracted from the final value of the variable $\psi_3^{\$}$. On the other hand, value of ϕ_i^e variable can only be 4. In this context, we define a constraint for each ψ_t^r and ϕ_i^e variable, which reduces the domain of the these variable into a single value, once a value is assigned to every ϕ_i^s variable. Besides, since we assume that the leading agent can execute only one capability at a time, we define a set of constraints over the values of ϕ_i^s and ϕ_i^s variables to reflect this constraint.

The resulting CSP instance consists of a set of ϕ^s, ϕ^e and ψ^r variables. The constraints of the CSP instance restrict the values of ϕ^e and ψ with respect to the values assigned to ϕ^s variables in order to guarantee that the leading agent has enough resources to satisfy the considered properties while respecting the temporal constraints. The existence of a solution to such a CSP instance means that there is a possible execution in which all the properties that are included by the considered commitments as well as the temporal constraints and resource requirements of these properties are satisfied. Hence, every commitment is fulfilled in this possible execution, which means that the commitments are feasible. On the other hand, if there is no solution, then it is not possible to satisfy some of the properties (i.e., there is no such possible execution). Hence, it is also not possible to fulfill every commitment, which means that the commitments are not feasible.

Let us present an example to demonstrate how our formulation works in a practical case. Suppose that Mer has the following commitments and aims to check feasibility:

- $c_1 = C(Cus, Mer_1, \top, P(1, 3, Furniture_1Paid)^{\nabla})^A$
- $c_2 = C(Cus, Mer_2, \top, P(2, 5, Furniture_2Paid)^{\nabla})^A$
- $c_3 = C(Bank, Cus, \top, P(4, 6, LoanTaken)^{\nabla})^A$

First, we extract the properties from the commitments in $\mathcal{C}$ and associate them with the responsible agents. Hence we have the set of properties $\mathcal{P}$ as follows:

- $p_1 = P(1, 3, Furniture_1Paid)^{\nabla}$, Cus is responsible
- $p_2 = P(2, 5, Furniture_2Paid)^{\nabla}$, Cus is responsible
- $p_3 = P(4, 6, LoanTaken)^{\nabla}$, $Bank$ is responsible

With respect to the commitments in $\mathcal{C}$, Cus is responsible to bring about p_1 and p_2 and $Bank$ is responsible to bring about p_3. We present the time intervals of these properties in Figure 5.4. Suppose that Cus has 10 Dollars initially. Also suppose that Cus needs 10 Dollars in order to satisfy each property in order to satisfy each property Cus needs two time units. Finally, as we explained above, we assume that $Bank$ would satisfy p_3 at 4 and accordingly Cus receives 15 Dollars.

In this setting, we have the following two CSP variables for each corresponding property p_1 and p_2: for p_1 there are ϕ_1^s with the domain $[0, 2]$ and ϕ_1^e with the domain $[1, 3]$, and for p_2 there are ϕ_2^s with the domain $[1, 4]$ and ϕ_2^e with the domain $[2, 5]$. There are also seven $\psi_t^{\$}$ variables for each t such as $0 \leq t \leq 6$.

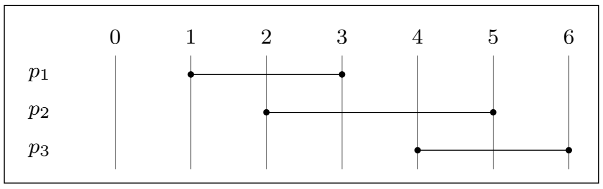

Figure 5.4. Time intervals of the properties in the CSP Example.

Let us consider one of possible solutions this CSP instance.

- $\phi_1^s = 1, \phi_1^e = 2, \phi_2^s = 4, \phi_2^e = 5$
- $\psi_0^\$ = 10, \psi_1^\$ = 0, \psi_2^\$ = 0, \psi_3^\$ = 0, \psi_4^\$ = 5, \psi_5^\$ = 5, \psi_6^\$ = 5$

Since Cus has initially 10 Dollars, $\psi_0^\$ = 10$. When 1 is assigned to ϕ_1^s the constraint over $\psi_1^\$$ applies and makes its value 0, since Cus spends 10 Dollars to satisfy p_1. Besides 2 is assigned to ϕ_1^e, since it take two time units to satisfy p_1. When 4 is assigned to ϕ_2^s, the value of the $\psi_4^\$$ is set to 5, taking into account the satisfaction of p_3 by $Bank$ (i.e., Cus receives 15 Dollars). Finally, 5 is assigned to ϕ_2^e, , since it take two time units to satisfy p_2.

In order to present a case where there is no solution (i.e., commitments are not feasible), assume that Cus has initially 0 Dollars. There are three values, $0, 1$ and 2 in the domain of ϕ_1^s. If 0 is assigned to ϕ_1^s, then the value of the $\psi_0^\$$ is restricted to -10 by the constraint that is defined over $\psi_0^\$$. However, the domain of $\psi_0^\$$ is $[0, \infty]$. Hence, it is not possible to assign -10 (or any other value) to $\psi_0^\$$, when 0 is assigned to ϕ_1^s. The same situation occurs when 1 or 2 is assigned to ϕ_1^s. In this case there is no consistent value to assign either to $\psi_1^\$$ or $\psi_2^\$$. Since there does not exist an assignment in which all ϕ_1^s, $\psi_0^\$$, $\psi_1^\$$ and $\psi_2^\$$ have consistent values due to the associated constraints, there is no solution to this CSP instance.

5.5. Feasibility and Analysis of Agent States

The leading agent can use the algorithm in Figure 5.3 to check feasibility of its commitments providing an agent state to the algorithm. This agent state can be the actual current state of the leading agent or a hypothetical state that the agent creates on purpose. In the former case, the leading agent already has some commitments and it aims to check feasibility of these commitments. In the later case, the leading agent creates a hypothetical state by modifying its actual state through adding new commitments to it. For example, the leading agent that is considering to offer a protocol that it generated as we defined in the previous chapters, should first check feasibility of the protocol's commitments. In such a case, the leading agent creates a hypothetical state by extending its current state (i.e., existing commitments) with the commitments that are going to be created when the protocol is enacted and then checks feasibility of its commitments using this hypothetical state. If the commitments in the hypothetical state are feasible, then it is safe to offer and enact the protocol. Otherwise, the leading agent should omit this protocol, since it would not be possible to fulfill all commitments, if the protocol was enacted.

On the other hand, the leading agent may also consider to remove some commitments while creating a hypothetical state to check feasibility. As we discussed before, agents have varying levels of trust in other agents with respect to their behavior about their commitments. Accordingly, while checking feasibility, the leading agent that does not trust in a particular agent, may consider to create a hypothetical agent state by removing the commitments in which that particular agent is the debtor. Discarding untrusted commitments may be critical to correctly decide on feasibility. Remember that our feasibility algorithm assumes that the other agents that participate in commitments always fulfill their commitments. Therefore, while computing feasibility, the algorithm takes the resources that the leading agent will acquire due to other agents' commitments into account. However, if the leading agent keeps a commitment in the agents state that is going to be used to check feasibility, even though it believes that the commitment is going to be violated by its debtor, then the decision about feasibility may not be correct. Hence, in order to decide correctly, the leading agent should create a hypothetical agent state by removing such a commitment from the actual agent state before checking feasibility. If the commitments of the leading agent are still feasible after removing that commitment, then the leading agent can conclude that its state is safe.

Table 5.1. Role and resource variations in commitments from the leading agent's point of view.

ID	Form	Example
α_1	$C(x, \_, \top, p)^A$	$C(Cus, Mer, \top, P(Mon, Wed, FurniturePaid)^\nabla)^A$
α_2	$C(\_, x, \top, p)^A$	$C(Mer, Cus, \top, P(Mon, Wed, HaveFurniture)^\nabla)^A$
α_3	$C(x, \_, \theta, p)^C$	$C(Cus, Mer, P(Mon, Wed, HaveFurniture)^\nabla,$ $P(Thu, Fri, FurniturePaid)^\nabla)^C$
α_4	$C(\_, x, \theta, p)^C$	$C(Mer, Cus, P(Mon, Wed, FurniturePaid)^\nabla,$ $P(Thu, Fri, HaveFurniture)^\nabla)^C$

In conclusion, the results of the feasibility algorithm may be interpreted in several ways by the leading agent, depending on the commitments the agent puts into the agent state that it provides to the algorithm. To examine this issue in detail, we first consider the possible active and conditional commitment forms in which the leading agent may participate. Then, using different combinations of these forms, we develop a set of realistic cases. We then discuss what type of conclusions the leading agent can come up with in these cases using the result of our algorithm.

In Table 5.1 we present the four possible commitment forms from the leading agents point of view. The commitments differ based on the role (i.e., debtor or creditor) of the leading agent and the property that it should satisfy. For readability we use the week days (e.g., Mon for Monday) in properties, instead of integer time values. With respect to the forms in Table 5.1 the leading agent acquires and spends resources as follows:

α_1 corresponds to an active commitment in which x is the debtor and should spend some resources to satisfy p in order to fulfill its commitment. In the example, Cus is committed to Mer to pay for furniture between Mon and Wed. Hence, Cus has to spend money to fulfill commitment.

α_2 corresponds to an active commitment in which x is the creditor and x expects to acquire some resources, if the commitment is fulfilled by its debtor by satisfying p. In the example, Mer is committed to Cus to deliver the furniture between Mon and Wed. Hence, Cus expects to acquire new resources (i.e., a piece of furniture), once the commitment is fulfilled.

α_3 is a conditional commitment in which x is the debtor and x should spend some resources to satisfy p in order to fulfill the commitment, if the antecedent of the commitment holds. Note that, when the antecedent of the commitment is satisfied (i.e., the commitment becomes active) x acquires some resources as result. In the example, Cus is committed to Mer to pay, if the furniture is delivered.

α_4 is a conditional commitment in which x is the creditor. x can make the commitment active by satisfying p, which requires x to spend some resources. When x makes the commitment active, then it expects to acquire some resources, if the commitment is fulfilled by its debtor by satisfying θ. In the example, Mer is committed to Cus to deliver the furniture, if Cus pays.

Below we present several realistic cases in our running example using different combinations of these commitment forms. After we define each case, we first present the results of our algorithm and then discuss how these results can be used by the leading agent to come up with conclusions about her commitments.

- *Case 1:* Suppose that Cus aims to check feasibility of her commitments given the following two commitments that match to the form α_1, to two different merchants for payment:

  (i) $c_1 : C(Cus, Mer_1, \top, P(Mon, Wed, Furniture_1Paid)^{\nabla})^{A}$

  (ii) $c_2 : C(Cus, Mer_2, \top, P(Mon, Wed, Furniture_2Paid)^{\nabla})^{A}$

  In this situation Cus should fulfill these two commitments using her own resources, since there is no commitment, which causes Cus to acquire any money. Let us consider two different situation depending on Cus's initial amount of money. Assume that Cus has initially enough money only to cover one of her payments. Then, given c_1 and c_2, our algorithm decides that these commitments are not feasible, since Mer does not have enough money to fulfill both c_1 and c_2. In this case, Cus concludes that she is going to violate at least one of c_1 or c_2 due to this result. Now, assume that Cus has initially enough money to make both payments. In this case, our algorithm decides that these commitments are feasible, since Cus has enough resources to fulfill both c_1 and c_2. Using this result, Cus can be sure that she is safe and can fulfill both of her commitments. Capturing feasibility of the leading agent's active commitments is particularly important, since these are the commitments that the agent must fulfill in any case. In this case, the leading agent should trust to its own resources (e.g., money) while fulfilling its commitments since it does not expect to acquire any new resources from other agents. Note that a cautious agent that does not trust to the other agents (since other agents may violate their commitments) may choose to consider only the commitments that match to α_1 while checking feasibility, even if it has commitments from the other agents, and hence may acquire resources as result of them. Then, if it turns out that its commitments are feasible, then the cautious agent can conclude that it is in a safe state, since it can fulfill all of its commitments using only its own resources, independent from the other agents' behavior.

- *Case 2:* Suppose that Cus aims to check feasibility of her commitments given the following two commitments that match to the form α_1 and the third commitment that match to the form α_2. In the third commitment, Emp represents the employer of Cus and $SalaryPaid$ represents payment of Cus's salary by Emp:

  (i) $c_1 : C(Cus, mer_1, \top, P(Mon, Tue, Furniture_1Paid)^{\nabla})^A$

  (ii) $c_2 : C(Cus, mer_2, \top, P(Wed, Thu, Furniture_2Paid)^{\nabla})^A$

  (iii) $c_3 : C(Emp, Cus, \top, P(Wed, Fri, SalaryPaid)^{\nabla})^A$

  In this case the leading agent, who aims to fulfill its own active commitments also expects other agents to fulfill their own active commitments. In this situation, Cus expects Emp to fulfill c_3 and therefore expects to acquire extra money (i.e., new resources). First consider this situation from a cautious leading agent's perspective. A cautious agent may assume that other agents fulfill their commitments as late as possible. Hence, if Cus is cautious, then she does not expect to get her salary before Friday. Therefore, she should generate a hypothetical state by modifying c_3, such that the interval of the consequent is only over Friday, to reflect this situation and should use this hypothetical state to check feasibility. Assume that Cus has initially enough money only to cover one of her payments. Then, given the hypothetical state, in which c_3 is modified as specified, our algorithm concludes that these commitments are not feasible, since Cus does not have enough money to fulfill both c_1 and c_2, even if c_3 is fulfilled on Friday. In this situation Cus concludes that, even if Emp fulfills her commitment and she acquires new resources, she will violate at least one of c_1 or c_2.

Secondly, consider a more venturesome approach where Cus chooses to assume that Emp fulfills her commitment as early as possible. Hence, Cus expects to acquire her salary at Wednesday and uses a hypothetical state that represents this situation by modifying c_3 accordingly. In this case, our algorithm returns that these commitments are feasible, since there exists a possible execution in which all commitments are fulfilled. For instance, if Emp fulfills c_3 on Wednesday, then Mer can fulfill both c_1 and c_2 using the money she acquires due to the fulfillment of c_3. According to this result, although it is possible to end up in a situation where Cus violates c_1 or c_2 (e.g., Emp pays on Friday), the venturesome Cus may still find this situation suitable, since it is still possible to fulfill both of her commitments.

- *Case 3:* Suppose that Cus aims to check feasibility of her commitments given the following two commitments that match to the forms α_1 and α_2:

  (i) $c_1 : C(Cus, Mer_1, \top, P(Mon, Tue, Furniture_1Paid)^{\nabla})^{A}$

  (ii) $c_2 : C(Cus, Mer_2, P(Mon, Tue, HaveFurniture_2)^{\nabla},$
  $P(Wed, Thu, Furniture_2Paid)^{\nabla})^{C}$

  Cases 1 and 2 consider only active commitments. However, in order to come up with more precise conclusions the leading agent should also take conditional commitments into account while examining feasibility of its commitments. This is because conditional commitments may become active and put the leading agent under obligation. Different than the previous cases, here c_2 is a conditional commitment with an antecedent property $P(Mon, Tue, HaveFurniture_2)^{\nabla}$. In this case Cus assumes that Mer_2 will satisfy this antecedent property and consequently c_2 will become active. Assume that Cus has initially enough money to only satisfy one of the payments specified above. Then, given these commitments, our algorithm returns that these commitments are not feasible, since Cus does not have enough money to fulfill both c_1 and c_2, once c_2 becomes active. Note that in the actual execution c_2 may never become active (i.e., expire) and accordingly Cus can fulfill c_1, which is the only active commitment. A cautious agent may try not to end up in such a situation, since it believes that the other agent will make the conditional commitment active. On the other hand, from a venturesome agent's point of view such situations may be acceptable. As a second situation, assume that Cus has initially enough money to handle both of the payments. In this situation, our algorithm returns that these commitment are feasible, since even if c_2 became active, Mer would have enough resources to fulfill both commitments. Hence, this would be a safe situation even for a cautious agent, since the agent could fulfill both commitments even if her conditional commitment became active, which depends on the other agent's actions.

- *Case 4:* Suppose that Cus aims to check feasibility of her commitments given the following two commitments that match to the forms α_1 and α_4:

  (i) $c_1 : C(Cus, Mer_1, \top, P(Mon, Wed, Furniture_1Paid)^{\nabla})^A$

  (ii) $c_2 : C(Mer, Cus, P(Mon, Wed, Furniture_2Paid)^{\nabla},$
  $$P(Thu, Fri, HaveFurniture_2)^{\nabla})^C$$

  There are also situations where the leading agent is the creditor of conditional commitments. That is the leading agent should satisfy the antecedent of such a commitment to make the commitment active spending certain resources, if it aims to acquire the resources due to fulfillment of this commitments. However, the spent resources to make such a commitment active, may prevent fulfillment of the leading agent's other commitments. In our example Cus aims to acquire a second furniture as result of the c_2's fulfillment. However, Cus should first make c_2 active by making a payment, which may prevent her to fulfill c_1 depending on the money available to her. Assume that Cus has initially enough money only to make one of the payments. If this is the case, our algorithm returns that these commitments are not feasible, since Cus has not enough money to discharge c_1 and to make c_2 active, concurrently. On the other hand, if Cus had enough money initially to make both payments, the situation would be different. If this was the case, our algorithm would return that these commitments are feasible. Note that there might be a third commitment also in which Cus is the debtor and the commitment requires the second furniture, which can only be acquired making c_2 active, in order to be fulfilled. If this was the case, it would be crucial for Cus to make c_2 active. However, as we pointed out above Cus should decide carefully, since making c_2 active might prevent her to fulfill c_1.

- *Case 5:* Suppose that Cus aims to check feasibility of her commitments given the following four commitments where $Bank$ represent a bank agent, $LoanTaken$ represents that the loan is given to Cus and $DebtPaid$ represents that Cus paid his debt:

  (i) $c_1 : C(Cus, Mer_1, \top, P(Mon, Tue, Furniture_1Paid)^{\nabla})^{A}$

  (ii) $c_2 : C(Cus, Mer_2, \top, P(Tue, Wed, Furniture_1Paid)^{\nabla})^{A}$

  (iii) $c_3 : C(Emp, Cus, \top, P(Wed, Thu, SalaryPaid)^{\nabla})^{A}$

  (iv) $c_4 : C(Mer, Bank, P(Mon, Tue, LoanTaken)^{\nabla}, P(Wed, Fri, DebtPaid)^{\nabla})^{A}$

  In the previous cases we examined commitments in which a certain type of resource (e.g., money) is required either by the antecedent or the consequent property of the commitment, but not both. However, it is possible to have commitments in which both the antecedent and the consequent properties require the same type of resource to be used. Hence, an agent that participates in such a commitment both spends and acquires the same type of resource. For instance, in c_4, Cus acquires some money (takes a loan) first when the commitment becomes active and later she spends again some money (pays her debt) to fulfill the commitment. Suppose that Cus has initially enough money to fulfill only one of c_1 or c_2. Given these commitments, our algorithm returns that these commitments are feasible. The reason is once c_4 becomes active, Cus acquires some extra money, which allows her to fulfill both c_1 and c_2. However, since c_4 has become active, Mer should also fulfill it by spending some more money. Although Cus has already spent all of her money to fulfill c_1 and c_2, she can still fulfill c_4 once c_3

is fulfilled by , which causes Cus to acquire enough money at Thu in the worst case. On the other hand, without considering c_4 our algorithm returns that these commitments are not feasible, since initially Mer has enough money to fulfill only one of c_1 and c_2. Once c_3 is fulfilled Cus acquires more money, but this happens only after one of c_1 or c_2 is already violated. Note that according to c_4, Cus should pay back more than she takes from the $Bank$ (e.g., interest), which may not look like beneficial initially. However, the result of our algorithm indicates that without c_4, Cus would violate either c_1 or c_2. Hence, it is indeed necessary for Cus to participate in c_4, in order to be able to fulfill all of her commitments.

Although we examined the above cases based on cautious and venturesome agent perspectives, these cases are actually general enough to be adopted by any type of agent. In general, a cautious agent would choose not to trust others. Hence, it may adopt an approach similar to the one in Case 1 or Case 2. On the other hand, a venturesome agent may chose to take more risks as in the other cases in order not to miss opportunities. Other than these two extremes, there could be a full spectrum of different agents. For example, while trusting a highly respectful merchant to honor her commitments, the customer might not trust another agent due to her disrespectful reputation. If the customer would compile her commitments for feasibility, she would add the first merchant's commitments to the collection, while ignoring the commitments of the second merchant. In some situations, an agent might have even deeper knowledge about the other agents (e.g., the agent might know other agents' resources) and might choose only a subset of the other agents' commitments that it would expect to be fulfilled and would ignore the rest of the commitments. In all of these cases, our approach would still work equally well since all we are requiring from the leading agent is to decide on the commitments that are going to be used to check feasibility.

5.6. Formal Properties of CheckFeasibility Algorithm

In this section, we show soundness and completeness of CheckFeasibility algorithm we present in Figure 5.3. In order to do that we first show the correspondence between feasibility (infeasibility) of a set of commitments and existence (non-existence) of a solution to the CSP instance that is created based on these commitments. Then, using this correspondence result we show soundness and completeness of our feasibility algorithm.

In the rest of this section, we consider the agent state $\eta_t = \langle \theta_t, \mathcal{P}_t, \mathcal{C}_t \rangle$. We assume that $\mathcal{C}_t^x$ represents the set of active and conditional commitments of $\mathcal{C}_t$ in which x participates (i.e., $\mathcal{C}_t^x = \{c \mid C(x, \_, \_, \_)^{A|C}$ or $C(\_, x, \_, \_)^{A|C}\} \subseteq \mathcal{C}_t$). We use e_κ to represent any (possible) execution from η_t to some agent state $\eta_{t'}$, such that $t < t'$. Also, we use $\mathcal{N}$ to refer to the CSP instance that is created from η_t applying the formal CSP definition as we specified in the previous section where the initial moment t_0 is equal to t. Finally, we assume that CHECKFEASIBILITY algorithm in Figure 5.3 uses a constraint solver that implements sound and complete constraint solving techniques to solve a given CSP instance [38]. More precisely, if the constraint solver returns a solution to the given CSP instance, then this solution is valid and if there exists a solution to the given CSP instance, then the constraint solver always returns a solution.

Lemma 5.13. *(Construction: Solution to Execution) Let κ be a solution to N. Then a possible execution e_κ can be constructed that corresponds to κ, such that every commitment in $\mathcal{C}_t^x$ is fulfilled at least in one state of e_κ.*

Proof Sketch: We can define a construction procedure as follows: Create an agent state $\eta t'$ for the initial moment t. If a ϕ_i^e variable is equal to t, update the state of the corresponding condition in θ_t of ηt. Update $\mathcal{P}_t$ and $\mathcal{C}_t$ applying $\Delta^{\mathcal{P}}$ and $\Delta^{\mathcal{C}}$, respectively. Repeat this procedure creating new agent states until a state $\eta_{t'}$ in which every commitment in $\mathcal{C}_{t'}$ is fulfilled. The complete construction procedure is in the corresponding proof on page 153 in Appendix B.2. □

Lemma 5.14. *(Construction: Execution to Solution) Let e_κ be possible execution, such that every commitment in $\mathcal{C}_t^x$ is fulfilled at least in one state of e_κ. Then a solution κ for N can be constructed that corresponds to e_κ.*

Proof Sketch: We can define a construction procedure as follows: Iterate over each state η_t in e_κ. For each η_t, if a property p_i turns from undetermined to satisfied, then set values of ϕ variables using t and by necessary information available via the domain knowledge. The complete construction procedure is in the corresponding proof on page 153 in Appendix B.2. $\Box$

Lemmas 5.13 and 5.14 show that we can construct a possible execution from a CSP solution and conversely a CSP solution from a possible execution, respectively. Now we can show the correspondence between feasibility (infeasibility) of a set of commitments and existence (non-existence) of a solution to the CSP instance.

Lemma 5.15. *(Correspondence: Feasible) Commitments in $\mathcal{C}_t^x$ are feasible if and only if there exists a solution κ to $\mathcal{N}$.*

Proof Sketch: Using Definition 5.11 and Lemma 5.13 it is trivial to show both directions. That is, commitments are feasible, then there exists a possible execution from which a solution can be constructed. On the other hand, if there exists a solution, a possible execution in which all of the commitments are fulfilled (i.e., feasible) can be constructed. The complete proof is on page 154 in Appendix B.2. $\Box$

Lemma 5.16. *(Correspondence: Infeasible) Commitments in $\mathcal{C}_t^x$ are not feasible if and only if there does not exist a solution κ to $\mathcal{N}$.*

Proof Sketch: Using Definition 5.11, Lemma 5.13 and Lemma 5.14 it is trivial to show both directions using proof by contradiction. That is we can show that contradictions occur, if a solution (exists/does not exist) when commitments are (infeasible/feasible). The complete proof is on page 154 in Appendix B.2. $\Box$

Lemmas 5.15 and 5.16 show the relation between feasibility of commitments and existence of a solution to the corresponding CSP instance. Now we can show soundness and completeness of CheckFeasibility algorithm in Figure 5.3. Intuitively, in order to call CheckFeasibility algorithm sound, the commitments should indeed be feasible, whenever the algorithm returns true, and otherwise the commitments should not be feasible, whenever the algorithms returns false.

Theorem 5.17. *(Soundness) Given the agent state η_t and the leading agent x, if* CheckFeasibility *algorithm (Figure 5.3) returns true, then the commitments in $\mathcal{C}_t^x$ are feasible and if* CheckFeasibility *algorithm returns false, then the commitments in $\mathcal{C}_t^x$ are not feasible.*

Proof Sketch: Using Lemmas 5.15 and 5.16 we can show both cases by creating contradictions. That is if the algorithm returns true when the commitments are not feasible, then there should be no solution to the CSP. But the algorithm returns true, only if there exists a solution to the CSP, which is a contradiction. We can use the same idea to show the second case. The complete proof is on page 155 in Appendix B.2. □

The CheckFeasibility algorithm is complete if it returns true, whenever the commitments are feasible and false otherwise.

Theorem 5.18. *(Completeness) Given an agent state η_t and an the leading agent x, if the commitments in $\mathcal{C}_t^x$ are feasible, then the* CheckFeasibility *algorithm (Figure 5.3) returns true, and if the commitments in $\mathcal{C}_t^x$ are not feasible, then* CheckFeasibility *algorithm returns false.*

Proof Sketch: Using Lemmas 5.15 and 5.16 we can show both cases by creating contradictions. That is if the commitments are feasible when the algorithm returns false, then there should be a solution to the CSP. But the algorithm returns false, only if there does not exist a solution to the CSP, which is a contradiction. We can use the same idea to show the second case. The complete proof is on page 156 in Appendix B.2. □

5.7. Computational Results

We conducted computational experiments to investigate performance and scalability of our feasibility algorithm (Figure 5.3). As we mentioned earlier, to the best of our knowledge there does not exist any large data set of commitment protocols for testing in the literature. Hence, we conducted our experiments on a data set we generated systematically as we explain below. In our experiments we used off-the-shelf JaCoP constraint solver which employs a depth-first search strategy. We performed our experiments on an Intel i7-2620 2.7 GHz processor with 2 GB memory running Ubuntu 11.04 Linux.

First, we explain our data generation method. Our aim is to test our feasibility algorithm on a hard instance in which we consider a time interval that is densely populated by a set of properties with variable time constraints, which are extracted from the leading agent's commitments. Random generation of such a data set is not difficult. However, if the entire data is generated arbitrarily, it becomes difficult to evaluate the results. Hence, we choose to generate the data systematically by using the following four parameters: (i) total number of properties, (ii) time required to satisfy a property's condition, (iii) number of resource types and the amount of each resource type required to satisfy a property's condition, and (iv) the amount of initial resources. As we discussed in our CSP formulation in Section 5.4, we assume the agents other than the leading agent satisfy the properties that they are responsible for at a certain moment. Accordingly, we do not represent these properties as variables, but embed resource changes that may occur due to them into the specification of initial resources.

In order to generate the properties, first we compute the total length of the time interval by multiplying the total number of properties with the time required to satisfy a property's condition, which we take equal for all properties. Then we create the first property by setting the first moment of its time interval to 0 and the last moment to the total length of the time interval. We generate the remaining properties one by one as follows. For each property, we first determine length of the new property's time interval by subtracting the time required to satisfy a property's condition from the previously generated property's time interval. Then to determine the new property's initial moment either (i) using the previously generated property's initial moment, or (ii) adding the time required to satisfy a property's condition to the previously generated property's initial moment. We use these two methods alternating for each consecutive property. In this way we create a set of feasible properties, which have overlapping time intervals, such that each property has a single slot to execute without conflicting with other properties. On the other hand, to generate an infeasible set of properties we simply set the amount of initial resources one less than the amount required to satisfy all the properties. Such a problem instance is hard to solve for a naive feasibility algorithm that searches for a solution arbitrarily. Hence, it allows us to examine how the underlying constraint solver performs comparing to a naive search, utilizing special techniques and heuristics, as we discuss in the rest of this section.

Beside the constraint propagation techniques, which are used by constraint satisfaction algorithms to reduce the size of the search space, a major factor that differs constraint satisfaction from a naive search algorithm, such as DFS, is the use of heuristics to guide the search process [38, 40]. These heuristics are mainly used for variable and value selection. Remember that, given a set of unassigned variables, a constraint solver assigns a single value to a single variable at each step while solving a CSP. Hence, at each step the constraint solver should first select a variable from the set of the unassigned variables and then the constraint solver should select a value to assign to the selected variable from its domain. The naive approach is to select both the variable and value randomly. This approach assumes that the structure of the problem's search space is totally unknown. But in many practical problem this is not the case (i.e., we have extra knowledge about the problem), and if certain variable and value selection heuristics are used, performance of the constraint solver can be improved significantly.

Accordingly, in order to find the best variable and value selection heuristics for our data, in the first part of our computational study, we performed a set of experiments. In these experiments we examined all combinations of variable and value selection heuristics as follows. For variable selection we considered three well-known heuristics: (i) *Smallest Domain* (SD), which selects the unassigned variable that has the least number of values in its domain, (ii) *Largest Minimum* (LM), which selects the unassigned variable that has the largest minimal value in its domain, and (iii) *Smallest Maximum* (SM), which selects the unassigned variable that has the smallest maximum

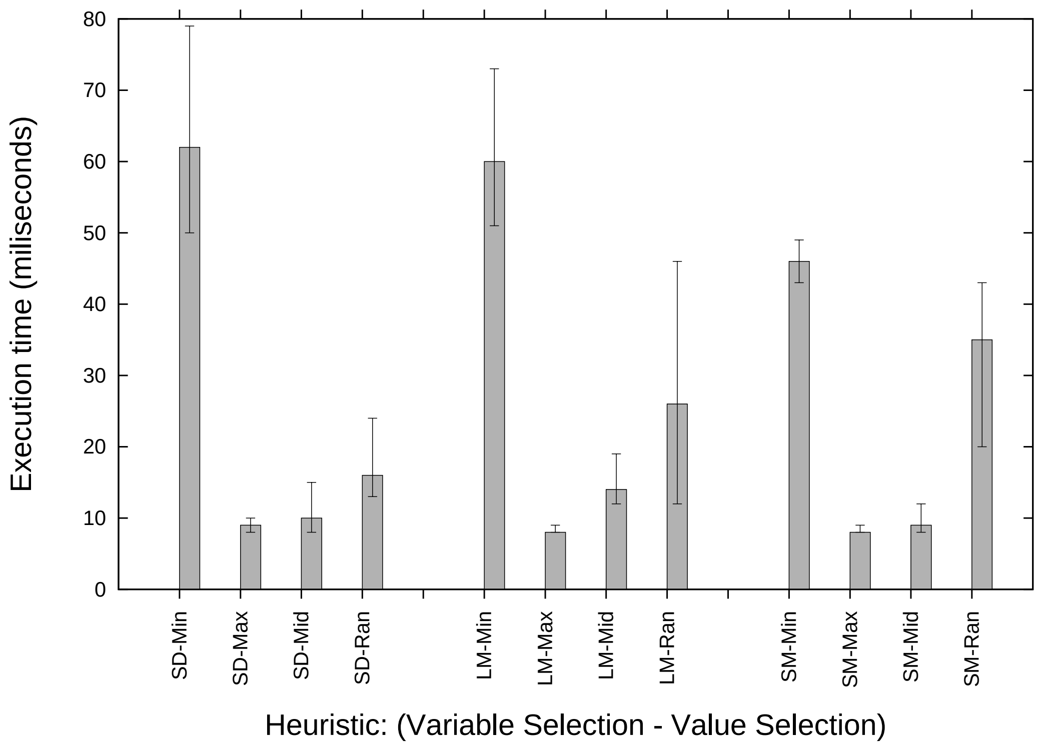

Figure 5.5. Comparison of feasibility algorithm's execution time using different combinations of variable and value selection heuristics, for 20 feasible properties.

value in its domain. SD is intuitive and works well in general, since it selects the variable that is most likely to cause a failure. Hence, it allows early pruning. LM and SM heuristics behave similarly. We chose them, because they reflect "do something as late as possible" behavior of an agent. For value selection we considered four heuristics: (i) *Minimum of Domain* (MIN), which assigns the minimum value of the selected variable's domain , (ii) *Maximum of Domain* (MAX), which assigns the maximum value of the selected variable's domain(iii) *Middle of Domain* (MID), which assigns the value in the middle of the selected variable's domain, and (iv) *Random* (RAN), which selects a value randomly from the selected variable's domain.

We present our results in Figures 5.5, 5.6, 5.7 and 5.8. In Figure 5.5 there are 20 feasible properties and each column shows the average execution time of our feasibility algorithm over five runs, for the combination of labeled variable and value selection

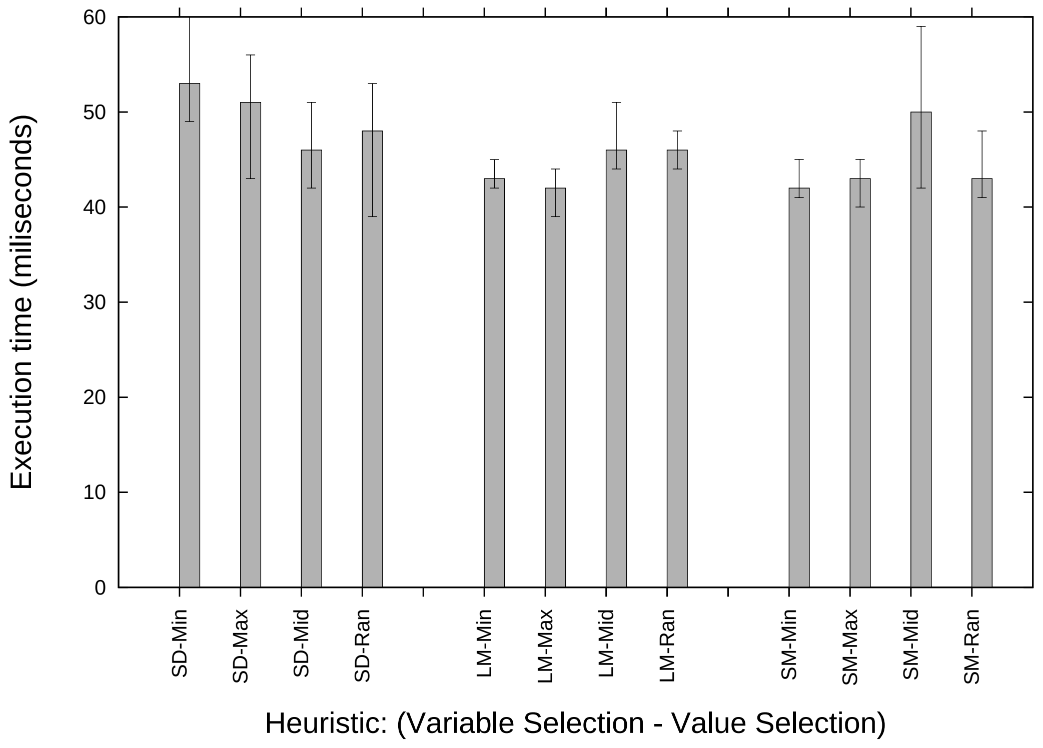

Figure 5.6. Comparison of feasibility algorithm's execution time using different combinations of variable and value selection heuristics, for 20 infeasible properties.

heuristics. The figure also shows minimum and maximum execution times we observed. Figure 5.6 shows the same results for 20 infeasible properties. Our results show that in the feasible case, execution time is significantly affected by the value selection heuristic. In this case, MAX heuristic for value selection produces the best results. MID also performs well, but it is slightly slower than MAX. MIN shows the worst performance and it is significantly slower than other heuristics. These results are valid for all variable selection heuristics. For SD heuristic, RAN performs better in average, but in general the difference between minimum and maximum execution times is large. MIN performs better in SM, but it is still significantly slower than others variable heuristics. Good performance of MAX and bad performance of MIN are expected result, since the structure of our data favors assignment of larger values. Accordingly, MAX can find a solution quickly with less number of backtracking, comparing to other variable heuristics. On the other hand, in the infeasible case, all combinations of

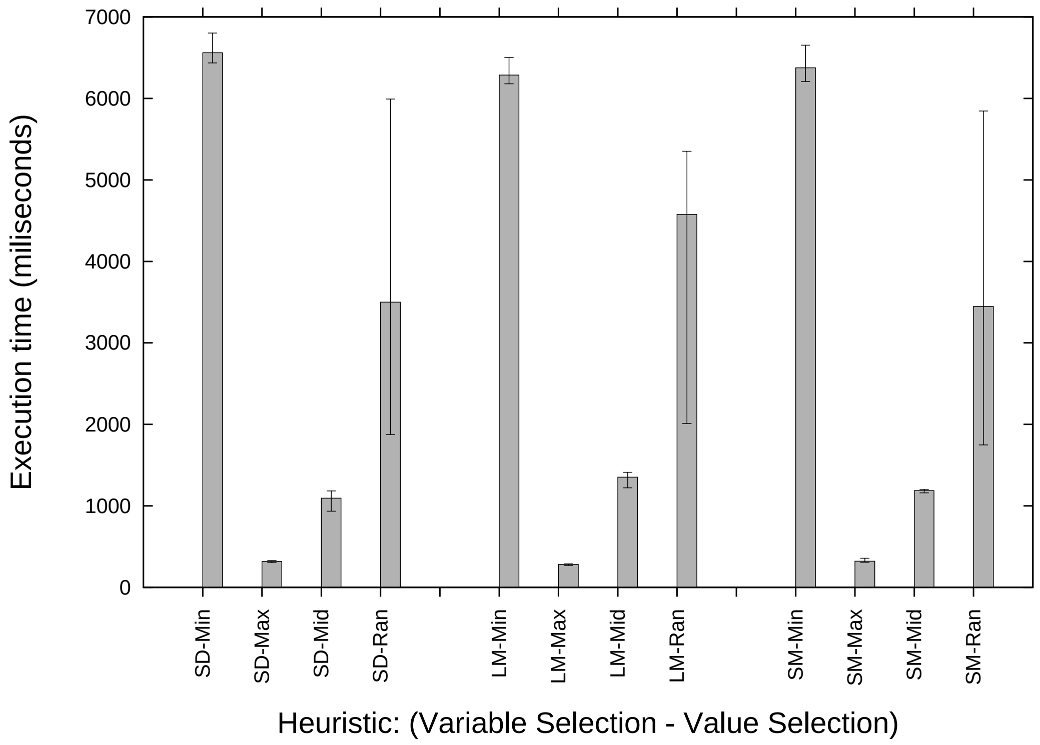

Figure 5.7. Comparison of feasibility algorithm's execution time using different combinations of variable and value selection heuristics, for 60 feasible properties.

variable and value selection heuristics perform similar results. This happens, since when the properties are infeasible, the constraint solver examines all possible assignments, independent from the heuristics.

In order to study the performance of these heuristics in a larger data set, we increased the number of properties from 20 to 60 and repeat our experiments. Figures 5.7 and 5.8 show the results of these experiments. From these results we observe that if there are more properties, then the difference between MAX and MID is more significant. We repeat our experiments for different number of properties (other than 20 and 60) and we observe that MAX performs better in all cases where the difference is more significant for larger number of properties. We do not present these results, since they do not present any significant difference than the results we present for 20 and 60 properties.

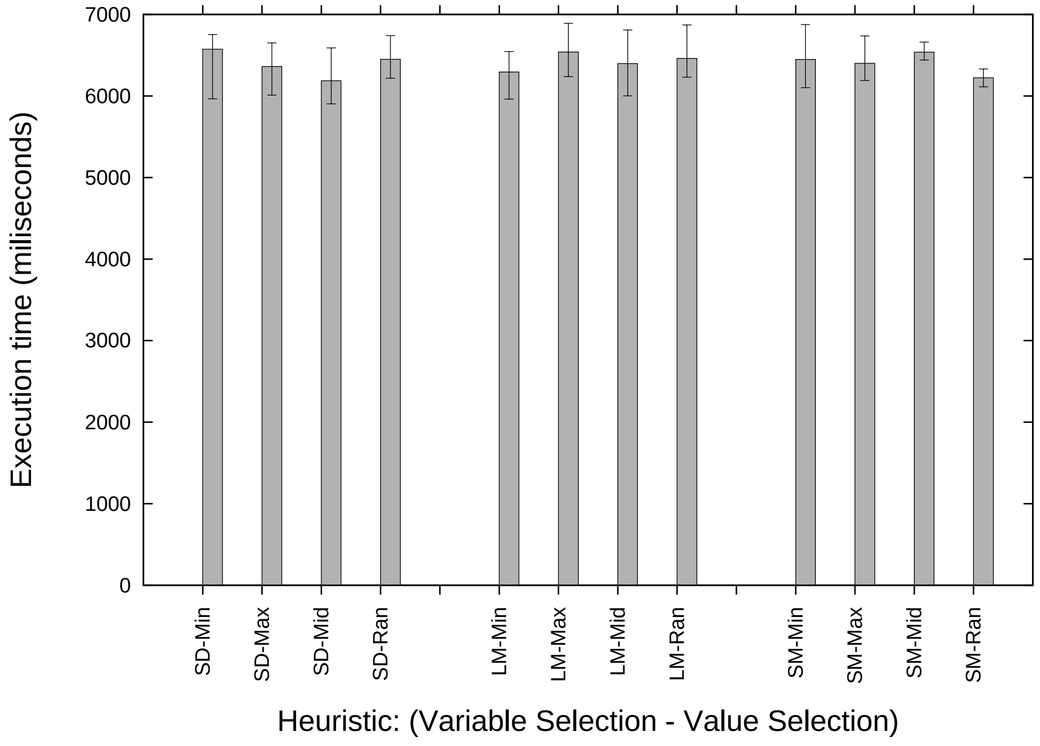

Figure 5.8. Comparison of feasibility algorithm's execution time using different combinations of variable and value selection heuristics, for 60 infeasible properties.

Comparison of various heuristics shows us the most appropriate variable and value selection heuristics for our data. In the second part of our experimental study, our aim is to evaluate the computational performance and the scalability of our feasibility algorithm, using these heuristic and show that it is applicable on practical cases. For this purpose we test our algorithms performance by executing it on a set of properties that vary in size. We also aim to show that modeling of feasibility problem as a CSP is advantageous over a naive method. For this purpose, we compared our algorithm's performance against a standard DFS algorithm that can be used to decide on the feasibility of a set of properties. In these experiments we measure the execution time of the algorithms while increasing the size of the property set from one to hundred.

This DFS algorithm uses the standard backtracking to decide on the feasibility as follows. Given a set of properties, the algorithm first creates an unassigned variable

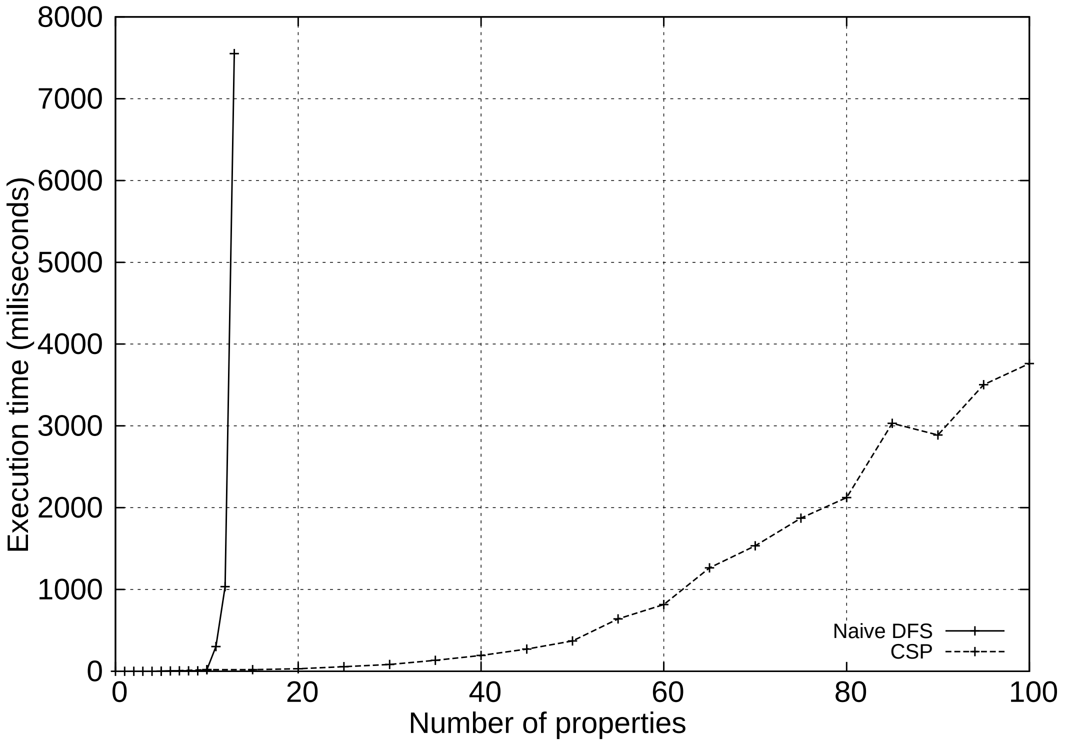

Figure 5.9. Execution times of our feasibility algorithm and the DFS algorithm for various size feasible property sets.

for each property. This variable represents when the leading agent starts to use its capability to satisfy the selected property. In other words it corresponds to ϕ^s variables in the CSP formulation. After that the algorithm repeats the following steps. First, it randomly selects an unassigned variable. Then, the algorithm randomly selects a value from the time interval of the property that corresponds to the selected variable, and assigns the selected value to the variable. Then it checks whether this assignment follows the resource requirements and the temporal constraints on the property (i.e., feasibility), by examining all variables that has an assigned value. If the new assignment is feasible, the algorithm selects another unassigned variable and repeats the steps above. Otherwise, it selects another value for assignment from the time interval of the property that corresponds to the variable and checks feasibility again. If none of the possible values of a variable is feasible, the algorithm backtracks to the previous variable. The algorithm terminates either when all variables have a valid value (i.e.,

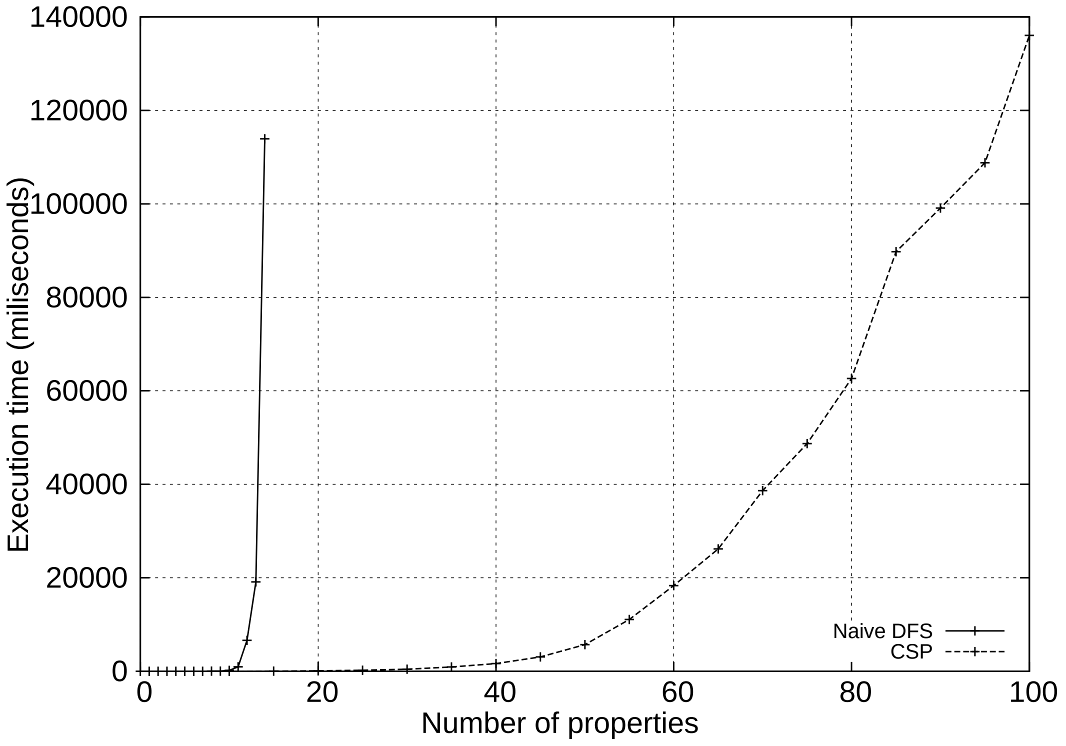

Figure 5.10. Execution times of our feasibility algorithm and the DFS algorithm for various size infeasible property sets.

properties are feasible) or when there are unassigned variables, but no value left to consider.

In Figure 5.9 we present the results of the two algorithms when the properties are feasible. The x-axis shows the number of properties and the y-axis shows the average execution time of the algorithms over five runs. The dashed line shows our feasibility algorithm and the straight line shows the DFS algorithm. The results show that the DFS algorithm's execution time increases exponentially as expected. Note that we do not measure the execution time of the DFS algorithm after 15 properties, since it does not terminate in a reasonable amount of time. On the other hand, our feasibility algorithm scales well and decides on the feasibility of 60 properties in less than a second.

In Figure 5.10 we present the results of the two algorithms when the properties are infeasible. The DFS algorithm's execution time increases exponentially as before. In addition to that, our feasibility algorithm's performance also worsens as expected, since it has to examine all alternative assignments to detect the infeasibility. However, CheckFeasibility decides on feasibility still in reasonable time even for 50 properties.

5.8. Discussion

In this chapter, first we first presented a method to negotiate over commitment protocols. Then, we studied feasibility of commitments, which shows whether an agent can fulfill a given set of commitments. In this context, we first extended our technical framework to show temporal constraints and resource requirements of commitments. Then we defined the feasibility problem and showed that deciding on it is NP-complete. With respect to this result, we formulated this problem as a CSP and used constraint satisfaction techniques to solve. We studied soundness and completeness of our method. Finally, we conducted computational experiments to examine execution performance of our algorithm. Although NP-completeness of this problem implies that it cannot be solved efficiently for all instances, we showed that our method performs significantly better than a naive method and applicable in many practical cases.

In our computational experiments we used the general purpose JaCoP constraint solver. Although JaCoP provides a wide range of constraint types, it does not provide specific heuristics designed for feasibility like problems. However, as pointed out in the constraint satisfaction literature, problem specific heuristics and their efficient implementations improve computational performance significantly [38]. In this context, arch consistency and constraint propagation techniques developed for scheduling and timetabling, which take temporal relations and resources into account, can be adopted for the efficient computation of commitment feasibility [41, 42].

Other than specialized heuristics, some simple pre-processing steps that are applied to the commitment data before computing feasibility, may also improve computational performance significantly. For instance, in most of the real world situations agents' commitments are not gathered in a single heavily overlapped cluster as we studied here. Instead they form small cluster, such that each cluster is separated by a gap. In such situations, each cluster can be solved separately based on their temporal orders, while propagating feasibility results and resource outcomes of earlier clusters to the later ones. As a result, since each cluster has fairly few commitments compared to the total number of commitments, computations can be done quickly.

6. DISCUSSION

6.1. Related Work

6.1.1. Commitments

Commitments have been first introduced to multiagent systems research by Castelfranchi and Singh by a series of conceptual papers [9, 12, 13]. In these papers they discuss that previous agent interaction approaches that are mostly based on the mental states of agents, such as KQML [43], are not adequate for agent interaction for various reasons, such as restriction of agents' autonomy, lack of formality and lack of support for heterogeneity. Then they introduce commitments as a more natural and powerful concept to model and reason about agent interaction. After the introduction of commitments, their formal models have been extensively studied in the literature and many reasoning methods have been proposed over these models.

Yolum and Singh propose one of the first and most influential formalism of commitments [8, 44, 45]. They represent a commitment as a domain independent meta-fluent in event calculus that binds two other abstract fluents as antecedent and consequent. Then, in order to capture how a commitment's state evolves over time, they define a set of commitment operations, such as create, discharge and cancel over abstract events. This representation allows them to show the lifecycle of a commitment in a domain independent manner. They also show that in order to use this model in a particular domain, it is enough to bind the abstract fluents of the commitments and the abstract events of the operations to domain dependent instances of fluents and events. Other than the commitment model, Yolum and Singh also introduce the concept of commitment protocol, which is a set of related commitments. They show how such protocols enable flexible specification of agent interaction by utilizing event calculus planning to generate all of the possible executions of a commitment protocol. That is given a commitment protocol, an initial state and an intended future state, they effectively generate a plan (i.e., a set of agent actions) that allows the agents to reach from the initial state to the desired future state.

This initial model is used by several researchers as a basis to study commitments further. Winikoff and colleagues extend Yolum and Singh's model in several directions [46]. They introduce desirable and undesirable states of a commitment protocol. An undesirable state is a state that is found undesirable at least by one agent. On the other hand, a desirable state is a state that is desired by at least one agent and it is not found undesirable by any agent. They discuss that it is necessary to identify especially the undesirable final (i.e., terminal) states in a protocol, since such states can be harmful to agents. However, they do not present a particular method for the identification of such state. Instead they state that undesirable final states should be identified by protocol designers and be avoided. Additionally, they also expanding Yolum and Singh's model by extending the discharge operation to cover some issues that are not considered in the initial model. For instance, Yolum and Singh's model does not handle the situation where a commitment about an already true fluent is created. To solve this issue, they introduced a new meta-predicate that causes a new commitment to be discharged immediately, if the consequent of the commitment already holds, when it is created.

Yolum and Singh's initial model is implicitly centralized and does not consider handling of parallel actions that are taken by different agents, which may cause to inconsistencies and race conditions in a multiagent system. Accordingly, Winikoff develops a distributed version of Yolum and Singh's model [20]. In this distributed version, each agent keeps its own copy of state (i.e., commitments and fluents). When an action is taken by an agent, the agent informs others about this action by sending a message and others update their state accordingly. The key point of this distributed version is the state update processes applied by agents when an action is taken and when a message is received. In the former case, the operations of the centralized model are used by adding extra message components to inform other agents. In the later case, the operations are redefined to be triggered as a result of incoming messages. Winikoff shows that if agents' own state copies are monotonic (i.e., if a fluent starts to hold, then it holds forever), then this distributed model is implicitly synchronized and therefore do not cause to race conditions. Winikoff justifies monotonicity assumption by showing that many practical situations can be represented in a monotonic model.

In a later study, Chopra and Singh propose a more general model that can handle nonmonotonic operations on commitments, such as cancellation and delegation, and still avoid inconsistencies [24]. Their model is also based on exchange of informative messages between agents. However, they achieve consistency by formalizing a set of alignment principles in their commitment model. One of these principles is notification, which ensures that two agent states are compared when both or neither have received vital information, such as commitment state changes. Another important principle is the definition of priority rules among operators, which prevents misalignment in the case of concurrent operations by different agents on the same commitment. They formalize these principles as a set of reasoning rules over operators and constraints on agents' behavior and show that their formalization prevents misalignment.

Based on the formalism of Yolum and Singh, in a series of papers Torroni and colleagues develop a monitoring framework for commitments using $\mathcal{S}$CIFF abductive logic programming proof-procedure [47–49]. Their main motivation is to develop a formal and operational framework that efficiently monitors commitments and verifies compliance of agents' actions with the protocols that they participate in. Since they are interested in run-time monitoring, they improve the basic commitment framework with temporal constraints over commitments. In this context they consider two types of constraints; (i) achievement of a condition within a time interval, and (ii) maintenance of a condition for the duration of a time interval. Although these kind of temporal constraints over commitments have been studied by other researchers [50–52], Torroni and colleagues provide a concrete operational framework that can handle these constraints at run-time. For this purpose they utilize an event driven implementation of event calculus called reactive event calculus, which is based on maximum validity interval concept of cached event calculus introduced by Chittaro and Montanari [53]. Using reactive event calculus they eliminate the necessity of backward reasoning at the occurrence of each event and accordingly they are able to do reasoning efficiently at run-time.

Beside event calculus, other techniques are also used to model commitments. Fornora and Colombetti develop an operational specification of commitments over speech acts [50, 54]. They focus on the lifecycle of an individual commitment, which is defined as a software object. Components of a commitment (e.g., debtor, consequent, state, etc.) are represented as data fields of the commitment object. In order to change the state of the commitment, they define a set of methods, which correspond to standard commitment operations, such as discharge and cancel. Finally, they define a commitment's antecedent and consequent as a temporal propositions. In their definition, a temporal proposition is a statement enclosed by a time interval that can be assigned to true, false or undefined. They use a commitment object to specify operational semantics of speech acts. For this purpose they map each speech act to one or more commitment operation. For instance, *inform*, which is an assertive speech act, is mapped to the creation of a commitment about the statement that is the subject of the inform act. Although this study is one of the earliest that recognize temporal aspect of commitments, this is not studied in detail by the authors, other than providing a definition.

Temporal logic is also used for modeling commitments [55]. Verdicchio and Colombetti [56] use a variant of CTL* with past-directed modalities for this purpose. In this context they define the past-directed operators *previous* and *since*, which correspond to the opposite of standard next and until operators, respectively. In their model, they represent a commitment as a first-order predicate. In order to represent the operations over commitments they define a set of axiomatic rules. For instance, discharge operation is defined as an axiom that states if there exists a commitment and the commitment's consequent holds eventually on all paths (i.e., in every possible evolution of the underlying system), then the commitment is fulfilled. Mallya, Yolum and Singh also provide a commitment model using CTL [52]. In this model they use two meta-predicates, *breached* and *satisfied* over a commitment, to show that the commitment is violated and fulfilled, respectively. Other than defining an expressive model of commitment, a major contribution of the authors is to show when a commitment is resolvable considering the temporal relations between its antecedent and consequent. They consider all of the 13 possible temporal relations as defined by Allen [57] and show that five out of 13 relations cannot be resolved. In all of these five non-resolvable relations, the time interval of the antecedent ends after the time interval of the consequent ends. Hence, it is not possible to decide on satisfaction of the antecedent within the time interval of the consequent. Following the resolvability results presented by Mallya, Yolum and Singh, we restrict the temporal relations between a commitment's antecedent and consequent commitment only to the resolvable relations, while considering feasibility.

In this thesis, we followed the basic commitment model of Yolum and Singh [8], in the generation (Chapter 3) and ranking (Chapter 4) of commitment protocols. This model provides a concise representation of commitments supporting practical reasoning that we need. On the other hand, while considering feasibility, we extended our framework to capture temporal constraints over commitments. In this extension we are mainly influenced by the commitment model that is developed by Torroni and colleagues [48]. This model provides an elegant representation of temporal properties over commitments. However, except for the use of the temporal properties our reasoning method is completely different than the one used by Torroni and colleagues (i.e., $\mathcal{S}$CIFF abductive logic proof procedure). For reasoning, we used possible executions in the abstract level and constraint satisfaction in the operational level to decide on commitments' feasibility. Finally, while studying feasibility, we extended the commitment framework with physical resources beside temporal properties. Although resource related topics have been extensively studied in many scientific areas for a long time [58], they have not been investigated in the context of commitments. Hence our consideration of physical resources is novel.

Different than most of the models we discussed above, in this thesis we do not consider logical implication while defining commitments' antecedent and consequent conditions. We choose to omit implication, since the aim of this thesis is not to provide a new logical commitment model. Instead, our aim is to develop general methods for generating, ranking and enacting commitment protocols. Hence, we do not consider implication to keep our methods general enough, to allow them to be used by any commitment based multiagent system, independent from the underlying commitment model.

Some of the commitment models we present above allow definition of nested commitment [8, 46]. A nested commitment is a commitment in which the antecedent or the consequent is another commitment. For example, a merchant commits to a customer to deliver furniture, if the customer commits to pay (i.e., $C(Mer, Cus, C(Cus, Mer, \top, PayFurniture), HaveFurniture)$). Nested commitments have not been studied in detail and both their conceptual meaning (e.g., what does committing to commit means, or can an agent commit to violate a commitment) and formalization (e.g., is there a limitation of nesting) are open research questions. Accordingly, in this thesis we do not consider nested commitments and left out them as future work.

6.1.2. Generation

Chopra *et al.* also considered the idea of support as we have done in generating commitment protocols [59]. In their definition, a goal is supported by an agent in three cases: (i) the agent itself is capable to achieve the goal, (ii) another agent is committed to satisfy the goal condition and the agent supports the antecedent of the commitment, and (iii) the agent has a commitment to another agent where the antecedent of the commitment is the goal of the agent. In our definition of support we incorporate the first two cases. However, in the second case we define an extra condition, which checks whether the other agent is capable to satisfy the consequent. Besides, we do not consider the third case, since it does not establish an obligation on the other agent. Hence, the other agent (i.e., creditor) may have no incentive to bring about the antecedent of the commitment, which is the goal of the agent. Suppose that the goal of the merchant is to be paid and the merchant is committed to the customer to deliver the furniture, if the customer pays for it. According to their support definition, merchant's goal to be paid is supported by this protocol. But, if delivery of the furniture is not a goal of the customer, this commitment does not guarantee the payment. The aim of Chopra *et al.* is to develop a method to verify that an agent can achieve its goals by enacting a commitment protocol. For this purpose they develop a framework over a set of semantic rules that define the support concept. Then they feed this framework with an existing protocol and a set of goals and check whether the goals are supported by the given protocol. In this thesis we do not consider verification in that sense. Instead, we focus on the generation of new commitment protocols that support a set of goals.

Marengo *et al.* discuss control of events in the context of commitments [60]. Basically, an agent has control over an event, if the agent itself can initiate the event or another agent that is capable to initiate the event is committed to do so. Basically, their events and our agent capabilities (and services) correspond to the same concept. Hence, our definition of support and their definition of control is closely related. The main difference is the use of these two definitions. They use control to verify existing commitment protocols executions where does not make any difference to consider an antecedent as a precondition or incentive. Hence, they do not consider incentive explicitly as we do in our support definition. In our case, we use support to generate new commitment protocols. With respect to that we should considered incentive explicitly in our support definition while creating commitments. On the other hand incentive is implicit to their control definition. They also discuss the notion of safety, which is related to our feasibility concept. However, their notion is more abstract. They define a commitment as safe for its debtor, if the debtor controls the antecedent and can avoid the situations in which the commitment becomes active, or if the debtor controls the consequent and therefore able to fulfill the commitment when it becomes active. We do not consider the case where the debtor prevents the commitment to become active for its safety. Such a situation is not quite practical, since a debtor normally creates a commitment expecting (and intending) it to become active. Otherwise, it would not be meaningful to create the commitment in the first place. Moreover, we take resource constraints into account, which is not considered by Marengo *et al.* for safety.

Telang, Meneguzzi and Singh use hierarchical task networks (HTN) planning over a commitment model to generate plans that allow agents to enact commitment protocols [61, 62]. For this purpose first they formalize commitments and goals as an HTN planning domain. To formalize commitments first they define a set of predicates that represent the state of a commitment. Then in order represent state transitions by manipulating these predicates, they define a set of HTN operators that correspond to the commitment operations. They use the same approach also to formalize goals. They combine these two separate definitions by specifying a set of HTN methods, which show how a goal can be achieved by creating and manipulating commitments. Finally, to generate a plan that achieves a set of agent goals in a practical scenario, they add the domain dependent operators that represent the dynamics of the scenario. In this way the planning process takes commitments into account and creates a plan including actions on commitments, such as creating and discharging a commitment, to achieve the given goals. Although our protocol generation method and their planning method are based on the same principles, there are two major differences between these two methods. First, our aim is to generate a set of commitment protocols without considering execution of these protocols. Once one of these protocols is enacted agents are free to execute it in any way that they see fit. On the other hand, their approach generates a complete plan that dictates all the actions that should be performed by the agents. Accordingly, these two methods complement each other. Second, their approach is centralized. They assume that there is a central planner that knows all the agents, their goals and their capabilities and create plans with respect to this knowledge. On the other hand, we assume a decentralized system and generate a commitment protocol from an agent's perspective. Each assumption has its own advantages and disadvantages. While their method is more efficient in close well known domains, our method can be used without assuming complete knowledge of the system.

Gerard and Singh propose an approach for refactoring commitment protocols [63]. They argue that in most of the multiagent systems, the agents' design and the interaction protocols are strongly coupled. Hence, once the requirements of agents change, the protocols should also be updated and the same issue occurs when the requirements of a protocol changes. To overcome this drawback, they provide a method that is based on the interceptor software pattern [64]. For this purpose they first define a set of interceptors (additional executable elements to actual agent code) that mediate message flow between an agent and a protocol. Then they define a set of refactorings that work in cooperation with the predefined interceptors. As result once either the agent's or the protocol's requirements change, if certain refactorings are applied to the component to handle the new requirements, then the other component can continue to work as it is using interceptors. Refactoring of protocols is an efficient method to create new protocols from existing protocols. While generating commitments, we assume that there does not exist any protocol prior to the generation. But in most of the cases there exist protocols that can be adjusted to support a given set of goals. Hence, refactoring is a promising future direction for efficient generation of commitment protocols

6.1.3. Ranking

To the best of our knowledge, there is no quantitative ranking method for commitment protocols similar to the one that we propose in this thesis. The only relevant work on the ranking of commitment protocols is the planning method developed by Telang, Menguzzi and Singh [61, 62]. Since they use HTN planning, which computes utilities of plans as part of the planning process, the are able to use these values to rank their plans and corresponding protocols. However, their utility formulation takes only the costs of actions and do not consider trust as in our ranking method.

There are several studies to analyze qualitative properties of a commitment protocol. Yolum provides a framework for this purpose [15]. To achieve that Yolum examined possible runs of a protocol, which are defined over protocol states. The first analyzed property is *effectiveness*, which is defined over the progression of a protocol. Basically, if for each state (except the final states) there exists a transition to another state, then the protocol progresses. Moreover, in order to conclude that a protocol is effective, there should be no cycles beside progress. The second property is *consistency.* If a protocol is progressive and does not have a state that includes conflicting propositions, then the protocol is considered to be consistent. Beside defining these properties, Yolum also provide a set of algorithms to compute both effectiveness and consistency of a commitment protocol. Finally, Yolum considers robustness of a commitment protocol identifying two classes of protocols, recoverable and fault-tolerant. Recoverable protocols are the ones that can be repaired by at least one agent taking certain actions. On the other hand, if every agent can recover a protocol, then it is considered as fault-tolerant.

Johnson, McBurney and Parsons study similarity of dialogue game protocols, such as persuasion and negotiation [65]. Although their study does not consider commitments as we do here, they show some interesting equalities to compare protocols. They start from the basic idea of syntactic equality and then extend it to semantic and structural equivalence. For instance, they consider bisimulation in which two protocols are equal, if any state transition achievable in one protocol is also achievable in the protocol. Other equalities are length and final state equality.

6.1.4. Enactment

Normative concepts such as obligation, prohibition and permission are studied in the multiagent systems [66,67]. Normative concepts are closely related to commitments, since both norms and commitments represent the regulative aspects of multiagent systems. However, while commitments focus on agents' interaction, normative concept focus on individual agents. In general, a normative concept is associated with a specific action in the context of a role. For example, consider a program committee role in a multiagent system that correspond to a conference organization. An agent that enacts the program committee role is obligated to review papers that are assigned to her. Accordingly she is permitted to access to those assigned papers. On the other hand, she is prohibited to access any paper that is not assigned to her.

Conflicts between normative concepts is studied by Vasconcelos, Kollingbaum and Norman [68]. Basically, if an agent is both prohibited and obligated (or permitted) to take some action, then a conflict occur between these norms. Moreover, they also associate some temporal and resource constraints over norms, which are taken into account while considering a conflict. However, their method suffers from issues similar to that we previously identified with respect to our commitment conflict definition. First of all, their definition does not take into account the conflicts that can be captured if more than two norms are considered together. Consider the following (rather odd) example: suppose that the courier is obligated to deliver some furniture to the customer either on a business day or at the weekend. Also suppose that there are two prohibitions that apply to the courier while delivering: (i) the courier is prohibited to deliver in business days, and (ii) the courier is prohibited at the weekend. In this example, even though there is no conflict between the obligation and the individual prohibitions, clearly the courier cannot fulfill her obligation. Moreover, the authors also do not consider the conflicts that may occur between two obligations because of resource shortages. Normative concepts and their conflicts also studied in different areas such as philosophy and law [69, 70]. However, these studies are usually not computational or restricted to very specific logical system.

Singh discusses semantics of dialectical and practical commitments [71]. In this study, Singh provides a unified linear temporal logic based semantics for dialectical and practical commitments. In this context, Singh provides a set of reasoning postulates that are related to our conflict definitions. These are named "consistency" and "strong consistency". Consistency states that an agent cannot commit to false, and strong consistency states that an agent cannot commit to a negation of previously committed condition. Especially, the strong consistency postulate closely corresponds to our commitment conflict relation. Assuming that we can use negation and the meta-predicate "Inconsistent" interchangeably, both Singh's strong consistency postulate and our conflict definition can be used to identify conflicts. However, in Singh's semantics, the strong consistency postulate acts as a constraint and prohibits existence of such (conflicting) commitments. In our case, we do not consider such a prohibition, instead our aim is to detect such situations and delegate treatment of the conflict to the underlying multiagent system.

Goal conflicts are also studied in multiagent systems community. Thangarajah, Padgham and Winikoff study goal conflicts with respect to agent plans [72]. According to their definition, if achievement of a goal causes a holding condition to cease, which is required for the execution of a plan to achieve another goal, then there is an interference between these goals. To capture such situations they define preparatory effects and dependency links. They also show how parallel plans can be scheduled without causing interference by using dependency links. van Riemsdijk, Dastani and Meyer study the same problem in the context of logical agent programming languages [73]. They examined various semantics for such languages and how inconsistencies in agent's goal bases are handled in these semantics. They also show equivalence of their semantics to some other semantics, such as default semantics, in the literature.

Although various types of conflicts have been studied in multiagent systems research, to the best of our knowledge, our work is the first one in the formalization and computation of feasibility of a commitment protocol. On the other hand, there are some studies about the verification of various properties of commitment protocols. Venkatraman and Singh develop a method based on CTL and model checking to verify agents' run-time compliance with commitment protocols (i.e., agents behave according to the protocols) [14]. In their method each agent creates a local model using its observations, which are the messages exchanged among agents. Then the agent uses this model to reason about the state of its own and also about the state of other agents commitments. Accordingly the agent can verify compliance of its own and also others' with respect to the commitment protocol. El-Menshawy *et al.* propose a new CTL based logic to model commitments and develop a verification method extending the MCMAS [74] model checker to verify properties of commitment protocols, such as reachability, safety and liveness [75]. Telang and Singh define business patterns using commitments and show how these patterns can be used to design agent and also verify interaction protocols [76]. Different than feasibility, these methods aim to verify general properties of a commitment protocol. Hence, they do not consider individual agents and their resources that they have while executing the protocol. With respect to these differences, verification of a protocols general properties and feasibility are complementary.

6.1.5. Other Topics

In the early days of multiagent systems research, Speech Act theory [77] was widely used as a base by the research community to define agent communication mechanism. In speech act theory, communication is a form of action represented by various speech acts, such as assertives, directives and commissives. Earlier approaches for agent communication define speech acts using agents' mental states, such as beliefs, desires and intentions [78]. Knowledge Query and Manipulation Language [43] and Foundation for Intelligent Physical Agents (FIPA) Agent Communication Language [79] are two major examples of this approach. Although, mentalistic approaches provide a more suitable ACL semantics than conventional approaches such as finite state machines, there is a major criticism that they are not applicable to open multiagent systems, since in these systems mental states of agents are private and cannot be accessed from outside [9, 17]. Hence, participating agents are not be able to reason about the meaning of the communication, since they cannot observe the mental states of other agents. Even if participating agents reveal their mental state by their own will, one agent cannot use this information to interpret the communication with other agents, since other agents may always lie about their mental states. Other than this, more philosophical critics also state that communication is a public phenomenon and private mental states of agents neglect the public nature of communication.

Policies have been widely studied in computer science literature. These policies have been extremely powerful and are applied in many large systems today. The policies generally do no consider resource constraints, since the domains they apply to do not need them. Another important line of research is policy languages and frameworks. Among these, REI [80], Ponder [81], OWL-POLAR [82] and KaOS [83] are most widely known frameworks. These frameworks contain various, sophisticated tools such as policy editors or policy execution environments. They support a number of deontic concepts such as promises, prohibitions or permit. For example, they specify whether a person is permitted to enter a room or prohibited to enter a room. If the same person is both permitted and prohibited to enter a room, they can identify this as a policy conflict and apply various methods (such as precedence) to resolve conflicts. However, generally they do not consider time-bounded resource usage for these concepts. For example, it is not possible to state that certain amount of money is demanded to be allowed to enter a room by tonight. Resource requirements for given time intervals has been an important theme of this thesis. Deciding whether a given policy is feasible in terms of necessary resources is important. While our focus has been on commitments, the approach we have developed can be applied to deontic concepts, such as permissions. In the case of permissions, we can still detect infeasible situations, but since we are dealing with permissions, rather than promises, the infeasibility can be handled differently. For example, if again at most one person is allowed to enter a room and that two people have been permitted to enter, this can lead to an infeasible execution. However, just because these two people are permitted does not mean, they will actually go ahead and attempt to enter. Hence, contrary to our application here, even though we detect an infeasibility, we would not necessarily signal an exception.

Kafalı, Günay and Yolum develop a commitment-based method to capture violations of privacy-policies in social networks [84, 85]. They use commitments to model privacy policy agreements between the parties and employ model checking on these policies to capture violations. They show that privacy breaches that cannot be detected in traditional systems can be captured using this approach. Furthermore, they also develop a prediction tool that incorporates with semantic Web technologies to capture potential privacy breaches that may occur in the future depending on the evolution of the social network.

Commitment-based multiagent systems are applies to solve various practical problems. Singh, Chopra and Desai propose a commitment-based service oriented architecture, which replaces invocation-based service objects with engagement-based autonomous business services [86]. To achieve this, they develop a set of transaction patterns over commitments. These patterns reflect business requirements of various common business transactions and provide basic building blocks to develop complex interactions. First benefits of this commitment-based approach is flexible enactment of transactions using commitments. In other words, a transaction is considered as correct as long as commitments that describe the transaction are not violated. Hence, participants can execute their operational choices in a flexible manner within the boundaries of their commitments. Second benefit of their approach is the ease of specification and composition of business processes, since commitments specify business requirements explicitly and they can be interpreted and composed by stakeholders.

Desai *et al.* provide a commitment-based solution to the formalization of foreign exchange market protocols [87]. They argue that the current specifications for foreign exchange markets are informal and have unclear business semantics. They show how commitment-based protocols with well-formed semantics can be used to overcome these issues. To achieve that they first build a core set of foreign exchange interaction protocols using commitments. Then they compose these protocols to come up with variety of different foreign exchange processes. They show that rigorous specification and verification of the protocols via commitments overcome many issues that emerge in the previous systems.

6.2. Conclusions

In this thesis we claim that predefined commitment protocols are not adequate for large-scale, dynamic open multiagent systems, because of the variety of agents, changes in agent preferences and changes in environment. Accordingly, we argue that in order to regulate their interactions while pursuing their goals, agents in a multiagent system should be able to generate their own commitment protocols, taking the current context of the multiagent system into account. In order to achieve that, we developed a three-phase agent process in which an agent first generates a set of commitment protocols based on its goals, capabilities and other agents' services. Then, the generated commitment protocols are ranked from the leading agent's perspective based on the utility and trust metrics. Finally, the agent negotiates over selected feasible protocols with other participating agents in order to reach agreement on the protocol for enactment. Below we summarize our main contributions:

- We developed two algorithms for efficient generation of commitment protocols. While the first algorithm uses the depth-first traversal strategy, the second algorithm employs a divide and conquer strategy. Both algorithms are based on the definition of support for goals. Using this definition they generate all minimal commitment protocols that support an agent's goals with respect to its capabilities and other agents' services. We showed that both algorithms are sound and complete. Besides we performed computational experiments to analyze the algorithms performance. We observe that the algorithm that uses the divide and conquer strategy outperforms the algorithm that is based on the depth-first traversal strategy.

- We developed a method that combines utility and trust to compare and rank commitment protocols. We defined utility of a protocol by subtracting the total cost of the capabilities that the agent should perform to satisfy its responsibilities in the protocol, from the total benefit that the agent expects due to the fulfillment of the commitments. When computing benefit we also considered the agent's trust in others about fulfilling their responsibilities. Analyzing our method in a case study we showed that taking trust into account may change the ranking results significantly. We also observed that a protocol's cost by itself is an important indicator for usefulness, since cost is a bound on the worst case.
- We developed a method that utilizes constraint satisfaction techniques to decide on the feasibility of commitment protocols considering the resources available to an agent and the temporal constraints on commitments. We first showed the limitations of conflict relations, which are frequently used in the literature to capture undesired situations in multiagent systems. Then to overcome these limitations, we defined feasibility of commitments and show that deciding on feasibility is NP-complete. In order to decide on the feasibility of commitments, we developed an algorithm that transforms feasibility problem into a constraint satisfaction problem and applies constraint satisfaction techniques to come up with a solution. With respect to that, we provided a detailed definition for creating a CSP instance from a given set of commitments. We proved that our algorithm is sound and complete. Besides we performed computational experiments to analyze the performance of our algorithm. We observed that, our algorithm can decide on feasibility in a reasonable amount of time for fairly large problem instances. We also conducted a comparative study of various variable and value selection heuristic that can be used while deciding feasibility in our algorithm.

6.3. Future Directions

Our research opens up interesting directions for further research. In many occasions, agents' goals cannot be represented with a single condition as we have done here. An interesting direction is to extend our goal representation. For example, we can interpret a set of goals with respect to the disjunctive semantic (i.e., it is enough to achieve one of the given goals) or we may use the combination of conjunctive and disjunctive semantics. We can also integrate logical operators such as negation and implication into our framework. All this extensions will enrich our framework and allow us to handle more complex scenarios.

We have not examined the performance of our commitment protocol generation algorithms, when a particular goal is not supported. Our algorithms detect such a situation only after considering every capability and service for that goal, which may take significant time depending on the number capabilities and services. In order to eliminate this drawback, we may enhance our algorithms with pruning techniques that are frequently used in other domains. For example, the forward checking and arc-consistency techniques are used in constraint satisfaction to prune the search space by detecting inconsistencies in advance. Such pruning techniques would improve our algorithms performance in large problems.

There are a number of extensions considering ranking. One direction is considering variations to the benefit calculation that reflect other concerns about protocol executions. For example, an agent that aims minimal loss in any protocol can first compute all subsets of a protocol, then calculate each subset's benefit, and require that the benefit be positive for all subsets. Another interesting direction is to explore to robustness of our proposed ranking metrics, when the agent's domain knowledge is not entirely correct. Finally, another direction is development of methods to incorporate rankings from different metrics. It would be interesting to compare the performance of our ranking method with other ranking methods.

Our feasibility algorithm can be improved in several directions. In the implementation of our algorithm we use JaCoP constraint solver. Although JaCoP is a state of the art constraint solver and its performance is adequate in most situations, computational performance of our implementation can be improved using a specialized constraint solver that use fine tuned heuristics to feasibility. Another interesting direction is the integration of a concrete trust model into our approach to manage the choices of the leading agent to select other agent's commitments, while creating an agent state to check for feasibility. Then, it would be interesting to observe the actual execution of the system and examine the consequences of the leading agent's choices. For instance we can observe whether a cautious agent avoids violation of its commitments, or misses opportunities due to its behavior. Another future direction is extending the CSP model by introducing different type of temporal properties such as maintenance (i.e., maintain a condition for a period of time), beside our achievement oriented temporal property.

APPENDIX A: CSP DEFINITION

Below we formalize the creation of the CSP instance from the input of Check-Feasibility algorithm, which includes a set of commitments $\mathcal{C}$, a set of conditions θ, the leading agent x and the initial moment t_0. Besides, we assume that the domain knowledge (e.g., available resources, resource requirements of conditions, etc.) of x is also accessible while creating the CSP instance.

Given $\mathcal{C}$, we first extract the properties that are included by commitments in $\mathcal{C}$. We also crate a mapping $\Gamma : \mathcal{P} \to \mathcal{X}$, which associates each property in $\mathcal{P}$ to an agent in $\mathcal{X}$ with respect to the agents' responsibilities. Then, from the agent's domain knowledge we extract the mapping $\Omega : \mathcal{R} \to \mathcal{Z}$ that associates every resource type r from the set of resource types $\mathcal{R}$ to a non-negative integer value (from the set $\mathcal{Z}$), which indicates the initial amount of r available to x at t_0. Then, assuming that $|\mathcal{P}| = n$ and using $\mathcal{P} = \{p_i : P(t_s, t_e, u)^{\nabla}$ and $1 \leq i \leq n\}$ and the initial moment t_0, we create the time interval $\mathcal{T}$ that starts at t_0 and ends at $Max(\{t_e : 1 \leq i \leq n\})$. Finally, using the agent's domain knowledge and the properties for which the leading agent is not responsible, we create a constant δ_t^r for each resource type $r \in \mathcal{R}$ and moment $t \in \mathcal{T}$, which shows the amount of resource from type r the leading agent acquires at time t, because of satisfaction of those properties.

Now we are ready to define the variables and constraints of the CSP instance. First we explain ϕ^s and ϕ^e variables. For every property $p_i \in \mathcal{P}$ such that $p_i = P(t_s, t_e, u)^{\nabla}$ and $\Gamma(p_i) = x$ there is a CSP variable ϕ_i^s in Φ^s and a CSP variable ϕ_i^e in Φ^e. Domains of these variables are determined as follows. Below, τ^u is part of the x's domain knowledge and represents the time units required to satisfy u.:

- Given a property $p_i = P(t_s, t_e, u)^{\nabla}$ and τ^u, the domain of the corresponding ϕ_i^s variable is $[Max(t_0, t_s - \tau^u + 1), Max(t_0, t_e - \tau^u + 1)]$. That is if it takes τ units to satisfy the consequent of the property using a capability, then the agent can initiate this capability as early as moment $t_s - \tau^u + 1$ in order to satisfy the condition at the first moment of p_i's temporal constraint t_s. Similarly, if it takes τ units to satisfy the consequent of the property using a capability, then the agent can initiate this capability as most at moment $t_e - \tau^u + 1$ in order to satisfy the condition at the last moment of p_i's temporal constraint t_e.

- Given a property $p_i = P(t_s, t_e, u)^\nabla$, the domain of the corresponding ϕ_i^e variable is $[t_s, t_e]$. This case is straightforward. The condition of the p_i should be satisfied only at a moment within p_i's time interval, the domain of the corresponding ϕ_i^e is identical to p_i's time interval.

Now we define the following constraints over the variables in Φ^s and Φ^e. For every ϕ_i^s and ϕ_j^e pair, ϕ_j^e is equal to one less of the sum of ϕ_i^s's value and time required to satisfy the condition of p_i (i.e., τ^u),

$$\forall \phi_i^s \in \Phi^s, \phi_j^e \in \Phi^e : \text{ if } i = j \text{ and } \phi_i^s = t, \text{ then } \phi_i^e = t + \tau^u - 1.$$

For every ϕ_i^s and ϕ_j^s pair, values of ϕ_i^s and ϕ_j^s are not equal to each other, unless i is equal to j,

$$\forall \phi_i^s \in \Phi^s, \phi_j^s \in \Phi^s : \text{ if } i \neq j, \text{ then } \phi_i^s \neq \phi_j^s.$$

For every ϕ_i^s, ϕ_i^e and ϕ_j^s triple, ϕ_i^e is less than ϕ_j^s, if ϕ_i^s is less than ϕ_j^s,

$$\forall \phi_i^s \in \Phi^s, \phi_i^e \in \Phi^e, \phi_j^s \in \Phi^s : \text{ if } \phi_i^s < \phi_j^s, \text{ then } \phi_i^e < \phi_j^s.$$

The first constraint restricts the value of ϕ_i^e to a single value according to the value assigned to the corresponding ϕ_i^s and the time required to satisfy the condition of the corresponding p_i. The second and third constraints are considered together in order not to allow two capabilities to be used by the leading agent in parallel (i.e., the leading agent can satisfy one property at a time).

Now we explain ψ variables. For every moment $t \in \mathcal{T}$ and for every $r \in \mathcal{R}$ there is a variable ψ_t^r over the domain $[0,\infty]$ in Ψ and the following constraint applies to every such variable:

$$\forall \psi_r^t \in \Psi : \psi_r^t = \begin{cases} \delta_{p_i}^r + \delta_t^r + \psi_r^{t-1} & \text{if } \exists \phi_i^s \in \Phi^S : \phi_i^s = t \\ \delta_t^r + \psi_r^{t-1} & \text{otherwise} \end{cases}$$

where $\delta_{p_i}^r$ is the amount of resource from type r the leading agent should spend at time t to satisfy p_i, which is part of x's domain knowledge. Note that if $t = t_0$, then $\mathcal{R}_{t-1}^r$ is replaced with $\Omega(r)$. This constraint reduces the valid values of each ψ_t^r to a single value, according to value of the corresponding ϕ_i^s variable, the amount of resource acquired from other agents (i.e., δ_t^r) and the available resource in the previous moment (i.e., ψ_r^{t-1}).

APPENDIX B: PROOFS

B.1. Proofs of Chapter 3: Generation of Commitment Protocols

Proof of Lemma 3.2 (on Page 27): We can show this by induction. Base cases: (i) If u is $\top$ then it holds trivially and accordingly $G_x(u)$ is achieved. (ii) If x has a capability $F_x(\top, u)$, then u can be satisfied and accordingly $G_x(u)$ can be achieved. This is true, since the precondition of the capability holds trivially as specified in the base case (i.e., the capability can be used in every situation). (iii) If $C(y, x, \top, u)$ is in π and x believes that y provides $S_x(y, \top, u)$, then u can be satisfied and accordingly $G_x(u)$ can be achieved. This is true, since we assume that every commitment in π that can possibly be fulfilled is eventually fulfilled. Here y has the service to fulfill the commitment. Therefore the commitment is eventually fulfilled.

Inductive cases: (i) If x has a capability $F_x(\theta, u)$, where θ is not trivial (i.e., θ includes at least one condition), and for each q in θ it is the case that $x, \pi \Vdash G_x(q)$, then u can be satisfied and accordingly $G_x(u)$ can be achieved. This is true since by induction we know that every q in θ can be satisfied and accordingly $F_x(\theta, u)$ can be used. (ii) If $C(y, x, \theta \cup \{w\}, u)$ is in π and x believes that y provides $S_x(y, \theta, u)$ and for each q in θ it is the case that $x, \pi \Vdash G_x(q)$ and finally for w it is the case that $x, \pi \Vdash G_x(w)$, then u can be satisfied and accordingly $G_x(u)$ can be achieved. This is true since by induction we know that every q in θ and also w can be satisfied. As the result, $S_x(y, \theta, u)$ can be utilized and $C(y, x, \theta \cup \{w\}, u)$ can be fulfilled. $\square$

Proof of Theorem 3.5 (on Page 32): We use proof by contradiction. To find a contradiction, suppose that given the input as in Theorem 3.5 ProtocolBased algorithm returns a protocol π, but x does not support $\mathcal{G}$ with respect to π. According to Definition 3.1 and consequently 3.3, this happens only if at least one of the following conditions hold:

- A goal in $\mathcal{G}$ is not considered by the protocol π. This condition does not hold for any π generated by ProtocolBased algorithm, since unless every goal in $\mathcal{G}$ is considered, the algorithm does not reach the base case and hence does not return a protocol (i.e., condition in Line 1 never holds).
- There is no capability in $x.\mathcal{F}$ and no service in $x.\mathcal{B}$ to support a goal in $\mathcal{G}$, but a protocol π is generated by ProtocolBased algorithm. This condition does not hold for ProtocolBased algorithm, since if this is the case, then the algorithm returns an empty set as a result, indicating that it is not possible to support $\mathcal{G}$) (i.e., Π is initiated to $\emptyset$ and loops in Lines 6 and 16 do not execute). Note that independent from the availability of a service to support a goal, the algorithm behaves in the same manner (i.e., returns empty set) if there does not exist an incentive for the provision of the service (i.e., loop in Line 17 does not execute).
- A precondition or an incentive required for a capability or service is not considered by the protocol π. This condition does not hold for any π generated by ProtocolBased algorithm, since the algorithm adds every precondition (Lines 10 and 21) and incentive (Line 25) as a goal to the goal queue, if the goal is not already considered and as we show in the first case, every such goal is considered by the algorithm.

Since, none of these conditions hold for ProtocolBased algorithm, the algorithm does not generate π, such that x does not support $\mathcal{G}$ with respect to π, which contradicts with our initial assumption. □

Proof of Lemma 3.6 (on Page 33). We can show that ProtocolBased algorithm generates only minimal protocols using proof by contradiction. To find a contradiction, suppose that ProtocolBased generates a protocol π that is not minimal. Note that ProtocolBased algorithm generates a commitment only if there is a goal in the goal queue $\mathcal{G}_p$. Hence, ProtocolBased algorithm may generate a non-minimal protocol only if an irrelevant goal, which is not required to support the initially given goals, is added into $\mathcal{G}_p$. With respect to the Definition 3.1, ProtocolBased algorithm adds a goal into $\mathcal{G}_p$ only if there is a precondition of a required capability (Line 10), a precondition of a required service (Line 21) or an incentive required for the provision of a service(Line 25). Moreover, a goal is added into $\mathcal{G}_p$ only if that goal is not already supported by the current protocol, thanks to the corresponding guarding statements in the algorithm. Therefore ProtocolBased algorithm does not add an irrelevant goal to $\mathcal{G}_p$ and accordingly does not generate a non-minimal protocol, which contradicts with our initial assumption. Therefore, ProtocolBased algorithm generates only minimal protocols.

Note that an important assumption that we make while considering minimality is the non-existence of cycles among capabilities' and services' preconditions. That is, for capabilities, if $F_x(q, u)$ is in $\mathcal{F}$, then $F_x(u, q)$ is not in $\mathcal{F}$, and we assume the same for services. □

Proof of Theorem 3.7 (on Page 33): Because of Lemma 3.6 we know that Protocol-Based algorithm generates only minimal protocols with respect to Definitions 3.4. Hence, we should only show that ProtocolBased algorithm generates every protocol in $\Pi^{min}_{x,\mathcal{G}}$. We can show this by examining how the algorithm decides on which capabilities and services to use to support a goal. First, suppose that there is a single goal to support. In this case, the algorithm iterates over every capability and service and generates an alternative protocol for every matching capability and service. Hence, if there is a single goal, the algorithm generates every minimal protocol to support it. Now, suppose that the algorithm generates every minimal protocol for n goals. We can show that the algorithm generates every minimal protocol also for $n + 1$ goals using induction. Simply, if the algorithm generates every minimal protocol to supports all of the n goals, then in order to generate every minimal protocol that supports the $n + 1^{th}$ goal and all of the previous n goals, it is enough to add the $n + 1^{th}$ goal to the pending goal queue ($\mathcal{G}_p$) of every protocol that supports all of the n goals and make a recursive call considering each protocol and the corresponding $\mathcal{G}_p$. □

Proof of Theorem 3.8 (on Page 34): We use proof by contradiction. Below we assume that the auxiliary functions (e.g, *Enqueue*) always terminate. Also remember that

$x.\mathcal{F}$ and $x.\mathcal{B}$ are finite and constant. To find a contradiction, suppose that given the input as in Theorem 3.8, PROTOCOLBASED algorithm does not terminate. Failure of termination occurs for PROTOCOLBASED algorithm, if one of the following conditions hold:

- The pending goal queue $\mathcal{G}_p$ does not get empty and accordingly the algorithm continues to make recursive calls without reaching the base case in Line 1. First of all, since $\mathcal{G}$, $x.\mathcal{F}$ and $x.\mathcal{B}$ are finite, the number of goals that $\mathcal{G}_p$ contains is also finite and accordingly by processing each goal one by one, at some point $\mathcal{G}_p$ becomes empty and the base case is reached. However, we should also show that the algorithm does not add a goal g' to $\mathcal{G}_p$ more than once causing a cycle. The algorithm does not add g' to $\mathcal{G}_p$ more than once, since each *Enqueue* operation over $\mathcal{G}_p$ in Line 10, 21 and 25 is guarded by *IsConsidered* function that allows g' to be added $\mathcal{G}_p$ only if it is not already considered (i.e., not in $\mathcal{G}_s$ and not in $\mathcal{G}_p$). Besides, in order to be added to $\mathcal{G}_p$, g' should not be equal to the goal g that the algorithm is currently considering. Hence, each goal g' is added to $\mathcal{G}_p$ only once for each protocol, which prevents a cycle to occur. As a result, this condition does not hold for PROTOCOLBASED algorithm.
- Loop in Line 8 iterates forever. Since $x.\mathcal{F}$ is finite and constant, and for each f in $x.\mathcal{F}$ there is a finite number of preconditions, the loop eventually terminates. As a result, this condition does not hold for PROTOCOLBASED algorithm.
- Loop in Line 6 iterates forever. Since $x.\mathcal{F}$ is finite and constant, the iteration condition of the loop fails to hold eventually. Hence, if the loop in Line 8 terminates and every recursive call (Line 14) that is made within the body of the loop returns (terminates) eventually, then the loop also terminates eventually. We know that loop in Line 8 eventually terminates (second item). Besides, we know that every recursive call terminates eventually (first item). As a result, this condition does not hold for PROTOCOLBASED algorithm.
- Loop in Line 19 iterates forever. Since $x.\mathcal{B}$ is finite and constant, and for each s in $x.\mathcal{B}$ there is a finite number of preconditions, the loop eventually terminates. As a result, this condition does not hold for PROTOCOLBASED algorithm.
- Loop in Line 17 iterates forever. Since $x.\mathcal{B}$ is finite and constant, the iteration

condition of the loop fails to hold eventually. Hence, if the loop in Line 19 terminates and every recursive call (Line 29) that is made within the body of the loop returns (terminates) eventually, then the loop also terminates eventually. We know that loop in Line 19 eventually terminates (fourth item). Besides, we know that every recursive call terminates eventually (first item). As a result, this condition does not hold for ProtocolBased algorithm.

- Loop in Line 16 iterates forever. Since $x.\mathcal{B}$ is finite and constant, the iteration condition of the loop fails to hold eventually. Hence, if the loop in Line 17 terminates eventually, then the loop also terminates eventually. We know that loop in Line 17 eventually terminates (fifth item). As a result, this condition does not hold for ProtocolBased algorithm.

Since, none of these condition holds, ProtocolBased algorithm terminates, which contradicts with our initial assumption. □

Proof of Theorem 3.9 (on Page 38): We use proof by contradiction to show soundness of GoalBased algorithm. To find a contradiction, suppose that given the input as in Theorem 3.9, GoalBased algorithm returns a protocol π, but x does not support $\mathcal{G}$ with respect to π. According to Definition 3.1 and consequently 3.3, this happens only if at least one of the following conditions hold:

- A goal in $\mathcal{G}$ is not considered by the protocol π. This condition cannot hold for any π generated by GoalBased algorithm, since unless every goal in $\mathcal{G}$ is considered, the algorithm does not reach the base case and hence does not return a protocol (i.e., condition in Line 1 never holds). Besides, if the algorithm returns a set of previously generated protocols for a goal using the mapping $\mathcal{M}$ (without reaching the base case), consideration of all the goals is still guaranteed for these protocols, since a protocol is added to the mapping only if the algorithm reaches to the base case.
- There is no capability in $x.\mathcal{F}$ and no service in $x.\mathcal{B}$ to support a goal in $\mathcal{G}$, but a protocol π is generated by GoalBased algorithm. This condition cannot hold

for GoalBased algorithm, since if this is the case, then algorithm returns empty set as result, indicating that it is not possible to support $\mathcal{G}$) (i.e., Π is initiated to $\emptyset$ and loops in Lines 13 and 22 do not execute). Note that independent from the availability of a service to support a goal, the algorithm behaves in the same manner (i.e., returns empty set) if there does not exist an incentive for the provision of the service (i.e., loop in Line 23 does not execute). Finally, note that in this case it is not possible to have a mapping in $\mathcal{M}$ for the considered goal, since the algorithm never reaches the base case.

- A precondition or an incentive required for a capability or service is not considered by the protocol π. This condition cannot hold for any π generated by GoalBased algorithm, since the algorithm adds every precondition (Lines 17 and 27) and incentive (Line 31) as a goal to the goal queue and as we show in the first case, every such goal is considered by the algorithm.

Since, none of these condition holds for GoalBased algorithm, the algorithm does not generate π, such that x does not support $\mathcal{G}$ with respect to π, which contradicts with our initial assumption. $\square$

Proof of Lemma 3.10 (on Page 39). We can show that GoalBased algorithm generates only minimal protocols using proof by contradiction. To find a contradiction, suppose that GoalBased generates a protocol π that is not minimal. Note that GoalBased algorithm generates a commitment only if there is a goal in the goal queue $\mathcal{G}_p$. Hence, GoalBased algorithm may generate a non-minimal protocol only if an irrelevant goal, which is not required to support the initially given goals, is added into $\mathcal{G}_p$. With respect to the Definition 3.1, GoalBased algorithm adds a goal into $\mathcal{G}_p$ only if there is a precondition of a required capability (Line 17), a precondition of a required service (Line 27) or an incentive required for the provision of a service(Line 31). Therefore GoalBased algorithm does not add an irrelevant goal to $\mathcal{G}_p$. We should also show that a goal is considered only once in the context of each protocol. But this does not hold for GoalBased algorithm, since each sub-protocol of a protocol may consider the same goal. However, when merging such sub-protocols, the *Merge* function detects and filters out non-minimal protocols. Hence, the $Merge$ function guarantees to consider each goal only once and preserves minimality. Accordingly GoalBased algorithm does not generate a non-minimal protocol, which contradicts with our initial assumption. Therefore, GoalBased algorithm generates only minimal protocols.

Note that an important assumption that we make while considering minimality is the non-existence of cycles among capabilities' and services' preconditions. That is, for capabilities, if $F_x(q, u)$ is in $\mathcal{F}$, then $F_x(u, q)$ is not in $\mathcal{F}$, and we assume the same for services. $\square$

Proof of Theorem 3.11 (on Page 40): Because of Lemma 3.6 we know that GoalBased algorithm generates only minimal protocols with respect to Definitions 3.4. Hence, we should only show that GoalBased algorithm generates every protocol in $\Pi^{min}_{x,\mathcal{G}}$. We can show this by examining how the algorithm decides on which capabilities and services to use to support a goal. First, suppose that there is a single goal to support. In this case, the algorithm iterates over every capability and service and generates an alternative protocol for every matching capability and service. Hence, if there is a single goal, the algorithm generates every minimal protocol to support it. Now, suppose that the algorithm generates every minimal protocol for n goals. We can show that the algorithm generates every minimal protocol also for $n + 1$ goals using induction. Simply, if the algorithm generates every minimal protocol to supports all of the n goals, then in order to generate every minimal protocol that supports the $n + 1^{th}$ goal and all of the previous n goals, it is enough to add the $n + 1^{th}$ goal to the pending goal queue ($\mathcal{G}_p$) of every protocol that supports all of the n goals and make a recursive call considering each protocol and the corresponding $\mathcal{G}_p$. $\square$

Proof of Theorem 3.12 (on Page 40): We use proof by contradiction. Below we assume that the auxiliary functions (e.g, *Enqueue*) always terminate. Also remember that $x.\mathcal{F}$ and $x.\mathcal{B}$ are finite and constant. To find a contradiction, suppose that given the input as defined in Theorem 3.12, GoalBased algorithm does not terminate. Failure of termination occurs for GoalBased algorithm, if one of the following conditions hold:

- The pending goal queue $\mathcal{G}_p$ does not become empty and accordingly the algorithm continues to make recursive calls without reaching the base case in Line 1. First of all, since $\mathcal{G}$, $x.\mathcal{F}$ and $x.\mathcal{B}$ are finite, the number of goals that $\mathcal{G}_p$ contains is also finite and accordingly by processing each goal one by one, at some point $\mathcal{G}_p$ gets empty and the base case is reached. However, we should also show that the algorithm does not add a goal g' to $\mathcal{G}_p$ more than once causing a cycle. The algorithm does not add g' to $\mathcal{G}_p$ more than once, since each *Enqueue* operation over $\mathcal{G}_p$ in Line 17, 27 and 31 is guarded by an equality condition that allows g' to be added $\mathcal{G}_p$ only if it is not equal to the goal g that the algorithm is currently considering. Hence, each goal g' is added to $\mathcal{G}_p$ only once for each protocol, which prevents a cycle to occur. As result, this condition does not hold for GoalBased algorithm.
- Loop in Line 15 iterates forever. Since $x.\mathcal{F}$ is finite and constant, and for each f in $x.\mathcal{F}$ there is a finite and constant number of preconditions, the loop eventually terminates. As a result, this condition does not hold for GoalBased algorithm.
- Loop in Line 13 iterates forever. Since $x.\mathcal{F}$ is finite and constant, the iteration condition of the loop fails to hold eventually. Hence, if the loop in Line 15 terminates and every recursive call (Line 20) that is made within the body of the loop returns (terminates) eventually, then the loop also terminates eventually. We know that loop in Line 15 eventually terminates (second item). Besides, we know that every recursive call terminates eventually (first item). As a result, this condition does not hold for GoalBased algorithm.
- Loop in Line 25 iterates forever. Since $x.\mathcal{B}$ is finite and constant, and for each s in $x.\mathcal{B}$ there is a finite number of preconditions, the loop eventually terminates. As a result, this condition does not hold for GoalBased algorithm.
- Loop in Line 34 iterates forever. Since $x.\mathcal{F}$ and $x.\mathcal{B}$ are finite and constant, the number of protocols in Π' is finite and the iteration condition of the loop fails to hold eventually. Accordingly the loop eventually terminates. As a result, this condition does not hold for GoalBased algorithm.
- Loop in Line 23 iterates forever. Since $x.\mathcal{B}$ is finite and constant, the iteration condition of the loop fails to hold eventually. Hence, if the loops in Lines 25 and 34

terminate and every recursive call (Line 33) that is made within the body of the loop returns (terminates) eventually, then the loop also terminates eventually. We know that loops in Lines 25 and 34 eventually terminate (fourth and fifth items). Besides, we know that every recursive call terminates eventually (first item). As a result, this condition does not hold for GoalBased algorithm.

- Loop in Line 22 iterates forever. Since $x.\mathcal{B}$ is finite and constant, the iteration condition of the loop fails to hold eventually. Hence, if the loop in Line 23 terminates eventually, then the loop also terminates eventually. We know that loop in Line 23 eventually terminates (sixth item). As a result, this condition does not hold for GoalBased algorithm.

Since, none of these conditions hold for GoalBased algorithm it terminates, which contradicts with our initial assumption. $\square$

B.2. Proofs of Chapter 5: Enactment of Commitment Protocols

Proof of Theorem 5.12 (on Page 94): We can show that Feasibility is NP-complete by transforming SequencingWithIntervals, which is known to be NP-complete [37], to Feasibility. Definition of SequencingWithIntervals is as follows: Given a set of tasks T and for each $t \in T$, a length $l(t) \in Z^+$, a release time $r(t) \in Z_0^+$, and a deadline $d(t) \in Z^+$, is there any schedule for T that satisfies the release time and deadline constraints, while performing one task at a time?

Given this definition of SequencingWithIntervals it is straightforward to transform SequencingWithIntervals to Feasibility as follows.

(i) For each task $t \in T$, create an active commitment $c_t \in \mathcal{C}$ from agent x to y where the consequent of the commitment is the property $P(r(t), d(t), t)^{\nabla}$.
(ii) For each $t \in T$, create a capability f_t, which can be performed to satisfy t.
(iii) For each $t \in T$, set in the domain knowledge of x that performing f_t takes $l(t)$ units of time and f_t does not require any resources.

The resulting commitments in $\mathcal{C}$ are feasible if and only if there is a valid schedule for T, as specified in the definition of SequencingWithIntervals.

Assuming that creation of a commitment, creation of a capability and setting the domain knowledge takes constant time, complexity of this transformation is $\mathcal{O}(|T|)$. Hence, it can be done in polynomial time. The transformation is sound. That is, for any instance of SequencingWithIntervals transformed into Feasibility, commitments in $\mathcal{C}$ are feasible if and only if there is a valid schedule for T. The transformation is also complete. That is, the procedure transforms any given SequencingWithIntervals instance into a Feasibility instance. In conclusion, since we can transform from SequencingWithIntervals to Feasibility in polynomial time in a sound and complete manner, and SequencingWithIntervals is NP-complete, Feasibility is also NP-complete. $\square$

Proof of Lemma 5.13 (on Page 110): For every given solution κ to the CSP instance $\mathcal{I}$ we can construct the corresponding possible execution e_κ, in which every commitment is fulfilled, using the following procedure.

- *Initialization:* Set s_{t-1} as the first agent state in execution e_κ, populate $\mathcal{P}_t$ and $\mathcal{C}_t$, set variable n to t and jump to Step 2.
- *Step 1:* Set the variable n to $n+1$.
- *Step 2:* Create a new agent state s_n and add it to $e_\mathcal{S}$.
- *Step 3:* If there exists a CSP variable $\phi_i^e \in \kappa$ that is equal to n, then add the condition u_i of the corresponding p_i to θ_n.
- *Step 4:* Apply $\Delta^\mathcal{P}$ and $\Delta^\mathcal{C}$ to populate $\mathcal{P}_n$ and $\mathcal{C}_n$, respectively.
- *Step 5:* If ever commitment in $\mathcal{C}_n$ is not fulfilled jump to Step 1. Otherwise, end procedure.

$\square$

Proof of Lemma 5.14 (on Page 110): For every given possible execution e_κ in which every commitment is fulfilled, we can construct the corresponding solution κ to the CSP instance $\mathcal{N}$ using the following procedure:

- *Step 1:* Set the variable n to t.
- *Step 2:* For each property $p_i \in \mathcal{P}_n$, if p_i is undetermined in $\mathcal{P}_{n-1}$ and it holds in $\mathcal{P}_n$, or n is equal to t and p_i holds in $\mathcal{P}_n$, then set ϕ_i^e to n and ϕ_i^e to $n - d$ where d is the time required to satisfy p_i (which is extracted from domain knowledge).
- *Step 3:* If ever property in $\mathcal{P}_n$ is not satisfied jump to Step 1.

□

Note that in the constructive procedures of Lemma 5.13 and 5.14's proofs, we do not mention about the ψ variables for clarity. However, in the proof of Lemma 5.13, these variables can be handled in Step 3 by updating θ with respect to resource changes and in the proof of Lemma 5.14, the values of ψ variables can be set in a straightforward manner using domain knowledge in Step 2 along with ϕ variable.

Proof of Lemma 5.15 (on Page 111): We first show the case where the commitments in $\mathcal{C}_t^x$ are feasible. If the commitments in $\mathcal{C}_t^x$ are feasible, then by Definition 5.11 there exists a possible execution e_κ, which is an execution from the agent state η_t to an agent state $\eta_{t'}$, such that $t < t'$ and every commitment in $\mathcal{C}_t^x$ is in fulfilled state in $\mathcal{C}_{t'}$. Moreover, with respect to the Lemma 5.14, we know that if there is an execution η_κ, then there exists a corresponding solution κ to $\mathcal{N}$.

Now we show the second case where there exists a solution κ to the CSP instance $\mathcal{N}$. With respect to the Lemma 5.13, we know that if there exists a solution κ to the CSP instance $\mathcal{N}$, then there exists a corresponding possible execution e_κ, which is an execution from the agent state η_t to an agent state $\eta_{t'}$, such that $t < t'$ and every commitment in $\mathcal{C}_t^x$ is in fulfilled state in $\mathcal{C}_f$. Hence, by Definition 5.11, the commitments in $\mathcal{C}_t^x$ are feasible. □

Proof of Lemma 5.16 (on Page 111): We first show the case where the commitments in $\mathcal{C}_t^x$ are not feasible using proof by contradiction. To find a contradiction, suppose that the commitments in $\mathcal{C}_t^x$ are not feasible, but there exists a solution κ to the CSP instance $\mathcal{N}$. If there is a solution κ to the CSP instance $\mathcal{N}$, then with respect to the Lemma 5.13, we know that there is a corresponding possible execution e_κ, which is an execution from the agent state η_t to an agent state $\eta_{t'}$, such that $t < t'$ and every commitment in $\mathcal{C}_t^x$ is in fulfilled state in $\mathcal{C}_f$. Hence, the commitment in $\mathcal{C}_t^x$ are feasible with respect to Definition 5.11, which contradicts with our initial assumption.

Now we prove the second case where there is no solution κ to the CSP instance $\mathcal{N}$ using proof by contradiction. To find a contradiction, suppose that there is no solution κ to the CSP instance $\mathcal{N}$, but the commitments in $\mathcal{C}_t^x$ are feasible. If the commitments in $\mathcal{C}_t^x$ are feasible, then there is an execution e_κ, which is from the agent state η_t to an agent state $\eta_{t'}$, such that $t < t'$ and every commitment in $\mathcal{C}_t^x$ is in fulfilled state in $\mathcal{C}_{t'}$. With respect to the Lemma 5.14, we know that if there exists an execution e_κ, then there exists a corresponding solution κ to $\mathcal{N}$, which contradicts with our initial assumption. □

Proof of Theorem 5.17 (on Page 112): First remember that we assume that we use a sound and complete constraint solver. We use proof by contradiction. We first show the case where the feasibility algorithm (Figure 5.3) returns true. To find a contradiction, suppose that given the agent state η_t and the leading agent x, the algorithm returns true, but the commitments in $\mathcal{C}_t^x$ are not feasible. With respect to Lemma 5.16, we know that if the commitments in $\mathcal{C}_t^x$ are not feasible, then there is no solution κ to the CSP instance $\mathcal{N}$. If there is no solution κ to $\mathcal{I}$, then the constraint solver that is used by the algorithm returns no solution. If the constraint solver returns no solution, then the algorithm returns false, which contradicts with our initial assumption.

Now we show the case in which the algorithm returns false. To find a contradiction, suppose that given the agent state s_t and the leading agent x, the feasibility algorithm returns false, but the commitments in $\mathcal{C}_t^x$ are feasible. With respect to

Lemma 5.15, we know that if the commitments in $\mathcal{C}_t^x$ are feasible, then there is a solution κ to the CSP instance $\mathcal{N}$. If there is a solution κ to $\mathcal{N}$, then the constraint solver that is used by the algorithm returns a solution. If the constraint solver returns a solution, then the algorithm returns true, which contradicts with our initial assumption. $\square$

Proof of Theorem 5.18 (on Page 112): First remember that we assume that we use a sound and complete constraint solver. We use proof by contradiction. We first prove the case where the commitments in $\mathcal{C}_t^x$ are feasible. To find a contradiction, suppose that given the agent state η_t and the leading agent x, the commitments in $\mathcal{C}_t^x$ are feasible, but the feasibility algorithm (Figure 5.3) returns false. The algorithm returns false, only if the constraint solver returns no solution, which means that there is no solution κ to the CSP instance $\mathcal{N}$. With respect to Lemma 5.16, we know that if there is no solution κ to $\mathcal{N}$, then the commitments in $\mathcal{C}_t^x$ are not feasible, which contradicts with our initial assumption.

Now we show the second case where the commitments in $\mathcal{C}_t^x$ are not feasible. To find a contradiction, suppose that given the agent state η_t and the leading agent x, the commitments in $\mathcal{C}_t^x$ are not feasible, but the feasibility algorithm returns true. The algorithm returns true, only if the constraint solver returns a solution, which means that there is a solution κ to the CSP instance $\mathcal{N}$. With respect to Lemma 5.15, we know that if there is a solution κ to $\mathcal{N}$, then the commitments in $\mathcal{C}_t^x$ are feasible, which contradicts with our initial assumption. $\square$

REFERENCES

1. Hilbert, M. and P. López, "The World's Technological Capacity to Store, Communicate, and Compute Information", *Science*, Vol. 332, No. 60, pp. 60–65, 2011.

2. CISCO, "CISCO Visual Networking Index: Forecast and Methodology, 2012–2017", White Paper, May 2013.

3. Russell, S. J. and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2^{nd} edn., Pearson Education, Upper Saddle River, NJ, USA, 2003.

4. Singh, M. P. and M. N. Huhns, *Service-Oriented Computing: Semantics, Processes, Agents*, John Wiley & Sons, New York, NY, USA, 2005.

5. Coulouris, G., J. Dollimore and T. Kindberg, *Distributed Systems: Concepts and Design*, 4^{th} edn., Addison-Wesley, New York, NY, USA, 2005.

6. Tananbaum, A. S., *Computer Networks*, 4^{th} edn., Prentice Hall, Upper Saddle River, NJ, USA, 2002.

7. Fielding, R. T., J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, IETF, June 1999.

8. Yolum, P. and M. P. Singh, "Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments", *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pp. 527–534, 2002.

9. Singh, M. P., "Agent Communication Languages: Rethinking the Principles", *Computer*, Vol. 31, No. 12, pp. 40–47, 1998.

10. Chopra, A. K., A. Artikis, J. Bentahar, M. Colombetti, F. Dignum, N. Fornara,

A. J. I. Jones, M. P. Singh and P. Yolum, "Research Directions in Agent Communication", *ACM Transactions on Intelligent Systems and Technology*, Vol. 4, No. 2, pp. 20:1–20:23, 2013.

11. Jennings, N. R., K. Sycara and M. Wooldridge, "A Roadmap of Agent Research and Development", *Autonomous Agents and Multi-Agent Systems*, Vol. 1, No. 1, pp. 7–38, 1998.

12. Castelfranchi, C., "Commitments: From Individual Intentions to Groups and Organizations", *Proceedings of the First International Conference on Multiagent Systems*, ICMAS, pp. 41–48, 1995.

13. Singh, M. P., "An Ontology for Commitments in Multiagent Systems: Toward a Unification of Normative Concepts", *Artificial Intelligence and Law*, Vol. 7, No. 1, pp. 97–113, 1999.

14. Venkatraman, M. and M. P. Singh, "Verifying Compliance with Commitment Protocols", *Autonomous Agents and Multi-Agent Systems*, Vol. 2, No. 3, pp. 217–236, 1999.

15. Yolum, P., "Design Time Analysis of Multiagent Protocols", *Data and Knowledge Engineering*, Vol. 63, No. 1, pp. 137–154, 2007.

16. Mallya, A. U. and M. P. Singh, "An Algebra for Commitment Protocols", *Autonomous Agents and Multi-Agent Systems*, Vol. 14, No. 2, pp. 143–163, 2007.

17. Chopra, A. and M. P. Singh, "Agent Communication", G. Weiss (Editor), *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, 2^{nd} edn., Chap. 3, pp. 101–141, MIT Press, 2013.

18. Nau, D., M. Ghallab and P. Traverso, *Automated Planning: Theory & Practice*, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2004.

19. Desai, N., A. U. Mallya, A. K. Chopra and M. P. Singh, "Interaction Protocols as Design Abstractions for Business Processes", *IEEE Transactions on Software Engineering*, Vol. 31, No. 12, pp. 1015–1027, 2005.

20. Winikoff, M., "Implementing Flexible and Robust Agent Interactions using Distributed Commitment Machines", *Multiagent and Grid Systems*, Vol. 2, No. 4, pp. 365–381, 2006.

21. Colombetti, M., "A Commitment-based Approach to Agent Speech Acts and Conversations", *Proceedings of Workshop on Agent Languages and Communication Policies*, pp. 21–29, 2000.

22. Torroni, P., F. Chesani, P. Mello and M. Montali, "Social Commitments in Time: Satisfied or Compensated", M. Baldoni, J. Bentahar, M. van Riemsdijk and J. Lloyd (Editors), *Declarative Agent Languages and Technologies VII*, Vol. 5948 of *Lecture Notes in Computer Science*, pp. 228–243, Springer Berlin Heidelberg, 2010.

23. Darwiche, A. and J. Pearl, "On the Logic of Iterated Belief Revision", *Artificial Intelligence*, Vol. 89, No. 1–2, pp. 1–29, 1997.

24. Chopra, A. K. and M. P. Singh, "Multiagent Commitment Alignment", *Proceedings of the Eight International Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pp. 937–944, 2009.

25. Kafalı, O. and P. Torroni, "Exception Diagnosis in Multiagent Contract Executions", *Annals of Mathematics and Artificial Intelligence*, Vol. 64, No. 1, pp. 73–107, 2012.

26. Günay, A. and P. Yolum, "Service Matchmaking Revisited: An Approach Based on Model Checking", *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol. 8, No. 4, pp. 292–309, 2010.

27. Sensoy, M., J. Zhang, P. Yolum and R. Cohen, “Poyraz: Context-Aware Service Selection under Deception”, *Computational Intelligence*, Vol. 25, No. 4, pp. 335–366, 2009.

28. Sabater, J. and C. Sierra, “REGRET: Reputation in Gregarious Societies”, *Proceedings of the Fifth International Conference on Autonomous Agents*, pp. 194–195, 2001.

29. Teacy, W. T., J. Patel, N. R. Jennings and M. Luck, “TRAVOS: Trust and Reputation in the Context of Inaccurate Information Sources”, *Autonomous Agents and Multi-Agent Systems*, Vol. 12, No. 2, pp. 183–198, 2006.

30. Jennings, N. R., P. Faratin, A. R. Lomuscio, S. Parsons, M. J. Wooldridge and C. Sierra, “Automated Negotiation: Prospects, Methods and Challenges”, *Group Decision and Negotiation*, Vol. 10, No. 2, pp. 199–215, 2001.

31. Kraus, S., *Strategic Negotiation in Multiagent Environments*, MIT Press, Cambridge, MA, USA, 2001.

32. Aydoğan, R. and P. Yolum, “Learning Opponent’s Preferences for Effective Negotiation: an Approach Based on Concept Learning”, *Autonomous Agents and Multi-Agent Systems*, Vol. 24, No. 1, pp. 104–140, 2012.

33. Jonker, C. M., K. V. Hindriks, P. Wiggers and J. Broekens, “Negotiating Agents”, *AI Magazine*, Vol. 33, No. 3, pp. 79–91, 2012.

34. Rosenschein, J. S. and G. Zlotkin, *Rules of Encounter*, MIT Press, Cambridge, MA, USA, 1994.

35. Gruber, T. R., “Toward Principles for the Design of Ontologies used for Knowledge Sharing”, *International Journal of Human-Computer Studies*, Vol. 43, No. 5-6, pp. 907–928, 1995.

36. Günay, A. and P. Yolum, “Detecting Conflicts in Commitments”, C. Sakama, S. Sardina, W. Vasconcelos and M. Winikoff (Editors), *Declarative Agent Languages and Technologies IX*, Vol. 7169 of *Lecture Notes in Artificial Intelligence*, pp. 51–66, Springer Berlin Heidelberg, 2011.

37. Garey, M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Fremann and Company, New York, NY, USA, 1979.

38. Apt, K. R., *Principles of Constraint Programming*, Cambridge University Press, Cambridge, UK, 2003.

39. Kuchcinski, K., “Constraints-driven Scheduling and Resource Assignment”, *ACM Transactions on Design Automation of Electronic Systems*, Vol. 8, No. 3, pp. 355–383, 2003.

40. Brailsford, S. C., C. N. Potts and B. M. Smith, “Constraint Satisfaction Problems: Algorithms and Applications”, *European Journal of Operational Research*, Vol. 119, No. 3, pp. 557–581, 1999.

41. Baptiste, P. and C. Le Pape, “A Theoretical and Experimental Comparison of Constraint Propagation Techniques for Disjunctive Scheduling”, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, IJCAI, pp. 600–606, 1995.

42. Lajos, G., “Complete University Modular Timetabling using Constraint Logic Programming”, E. Burke and P. Ross (Editors), *Practice and Theory of Automated Timetabling*, Vol. 1153 of *Lecture Notes in Computer Science*, pp. 146–161, Springer Berlin Heidelberg, 1996.

43. Finin, T., R. Fritzson, D. McKay and R. McEntire, “KQML as an Agent Communication Language”, *Proceedings of the Third International Conference on Information and Knowledge Management*, pp. 456–463, 1994.

44. Yolum, P. and M. P. Singh, “Reasoning about Commitments in the Event Calculus: An Approach for Specifying and Executing Protocols”, *Annals of Mathematics and Artificial Intelligence*, Vol. 42, No. 1-3, pp. 227–253, 2004.

45. Yolum, P. and M. P. Singh, “Commitment Machines”, J.-J. C. Meyer and M. Tambe (Editors), *Intelligent Agents VIII*, Vol. 2333 of *Lecture Notes in Computer Science*, pp. 235–247, Springer Berlin Heidelberg, 2002.

46. Winikoff, M., W. Liu and J. Harland, “Enhancing Commitment Machines”, J. Leite, A. Omicini, P. Torroni and P. Yolum (Editors), *Declarative Agent Languages and Technologies II*, Vol. 3476 of *Lecture Notes in Computer Science*, pp. 198–220, Springer Berlin Heidelberg, 2005.

47. Alberti, M., F. Chesani, M. Gavanelli, E. Lamma, P. Mello and P. Torroni, “Verifiable Agent Interaction in Abductive Logic Programming: The SCIFF Framework”, *ACM Transactions on Computational Logic*, Vol. 9, No. 4, pp. 29:1–29:43, 2008.

48. Chesani, F., P. Mello, M. Montali and P. Torroni, “Representing and Monitoring Social Commitments using the Event Calculus”, *Autonomous Agents and Multi-Agent Systems*, Vol. 27, No. 1, pp. 85–130, 2013.

49. Chesani, F., P. Mello, M. Montali and P. Torroni, “Commitment Tracking via the Reactive Event Calculus”, *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence*, IJCAI, pp. 91–96, 2009.

50. Fornara, N. and M. Colombetti, “Operational Specification of a Commitment-based Agent Communication Language”, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pp. 536–542, 2002.

51. Mallya, A. U. and M. N. Huhns, “Commitments Among Agents”, *IEEE Internet Computing*, Vol. 7, No. 4, pp. 90–93, 2003.

52. Mallya, A. U., P. Yolum and M. P. Singh, "Resolving Commitments among Autonomous Agents", F. Dignum (Editor), *Advances in Agent Communication*, Vol. 2922 of *Lecture Notes in Computer Science*, pp. 166–182, Springer Berlin Heidelberg, 2004.

53. Chittaro, L. and A. Montanari, "Efficient Temporal Reasoning in the Cached Event Calculus", *Computational Intelligence*, Vol. 12, No. 3, pp. 359–382, 1996.

54. Fornara, N. and M. Colombetti, "Defining Interaction Protocols using a Commitment-based Agent Communication Language", *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pp. 520–527, 2003.

55. Emerson, E. A., "Temporal and Modal Logic", *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pp. 995–1072, MIT Press, 1990.

56. Verdicchio, M. and M. Colombetti, "A Logical Model of Social Commitment for Agent Communication", *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pp. 528–535, 2003.

57. Allen, J. F., "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM*, Vol. 26, No. 11, pp. 832–843, 1983.

58. Ibaraki, T. and N. Katoh, *Resource Allocation Problems: Algorithmic Approaches*, MIT Press, Cambridge, MA, USA, 1988.

59. Chopra, A. K., F. Dalpiaz, P. Giorgini and J. Mylopoulos, "Reasoning About Agents and Protocols via Goals and Commitments", *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pp. 457–464, 2010.

60. Marengo, E., M. Baldoni, C. Baroglio, A. K. Chopra, V. Patti and M. P. Singh, "Commitments with Regulations: Reasoning about Safety and Control in REG-

ULA", *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pp. 467–474, 2011.

61. Telang, P. R., F. Meneguzzi and M. P. Singh, "Hierarchical Planning about Goals and Commitments", *Proceedings of the Twelfth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pp. 877–884, 2013.

62. Meneguzzi, F., P. R. Telang and M. P. Singh, "A First-Order Formalization of Commitments and Goals for Planning", *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, AAAI, p. In Press, 2013.

63. Gerard, S. N. and M. P. Singh, "Evolving Protocols and Agents in Multiagent Systems", *Proceedings of the Twelfth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pp. 997–1004, 2013.

64. Gamma, E., R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, New York, NY, USA, 1995.

65. Johnson, M. W., P. McBurney and S. Parsons, "When Are Two Protocols the Same?", M.-P. Huget (Editor), *Communication in Multiagent Systems*, Vol. 2650 of *Lecture Notes in Computer Science*, pp. 253–268, Springer Berlin Heidelberg, 2003.

66. Castelfranchi, C., F. Dignum, C. M. Jonker and J. Treur, "Deliberative Normative Agents: Principles and Architecture", *Sixth International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages*, ATAL, pp. 364–378, 2000.

67. Boella, G. and L. van der Torre, "Regulative and Constitutive Norms in Normative Multiagent Systems", *Proceedings of the Ninth International Conference on the Principles of Knowledge Representation and Reasoning*, KR, pp. 255–265, 2004.

68. Vasconcelos, W. W., M. J. Kollingbaum and T. J. Norman, "Normative Conflict

Resolution in Multi-agent systems", *Autonomous Agents and Multi-Agent Systems*, Vol. 19, No. 2, pp. 124–152, 2009.

69. Sartor, G., "The Structure of Norm Conditions and Nonmonotonic Reasoning in Law", *Proceedings of the Third International Conference on Artificial Intelligence and Law*, ICAIL, pp. 155–164, 1991.

70. Elhaga, A., J. Breukerb and P. Brouwerc, "On the Formal Analysis of Normative Conflicts", *Information & Communications Technology Law*, Vol. 9, No. 3, pp. 207–217, 2000.

71. Singh, M. P., "Semantical Considerations on Dialectical and Practical Commitments", *Proceedings of the Twenty-third National Conference on Artificial Intelligence*, AAAI, pp. 176–181, 2008.

72. Thangarajah, J., L. Padgham and M. Winikoff, "Detecting & Avoiding Interference Between Goals in Intelligent Agents", *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, IJCAI, pp. 721–726, 2003.

73. van Riemsdijk, M. B., M. Dastani and J.-J. C. Meyer, "Goals in Conflict: Semantic Foundations of Goals in Agent Programming", *Autonomous Agents and Multi-Agent Systems*, Vol. 18, No. 3, pp. 471–500, 2009.

74. Lomuscio, A., H. Qu and F. Raimondi, "MCMAS: A Model Checker for the Verification of Multi-Agent Systems", A. Bouajjani and O. Maler (Editors), *Computer Aided Verification*, Vol. 5643 of *Lecture Notes in Computer Science*, pp. 682–688, Springer Berlin Heidelberg, 2009.

75. El-Menshawy, M., J. Bentahar, H. Qu and R. Dssouli, "On the Verification of Social Commitments and Time", *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems*, AAMAS, pp. 483–490, 2011.

76. Telang, P. R. and M. P. Singh, "Specifying and Verifying Cross-Organizational

Business Models: An Agent-Oriented Approach", *IEEE Transactions on Services Computing*, Vol. 5, No. 3, pp. 305–318, 2011.

77. Searle, J. R., *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press, Cambridge, UK, 1969.

78. Rao, A. S. and M. P. Georgeff, "BDI Agents: From Theory to Practice", *Proceedings of the First International Conference on Multiagent Systems*, pp. 312–319, 1995.

79. FIPA-TC-C, *Agent Communication Language, FIPA 2000 Specification*, Technical Report, Foundation for Intelligent Physical Agents, 2000.

80. Kagal, L., T. Finin and A. Joshi, "A Policy Based Approach to Security for the Semantic Web", D. Fensel, K. Sycara and J. Mylopoulos (Editors), *The Semantic Web - ISWC 2003*, Vol. 2870 of *Lecture Notes in Computer Science*, pp. 402–418, Springer Berlin Heidelberg, 2003.

81. Damianou, N., N. Dulay, E. Lupu and M. Sloman, "The Ponder Policy Specification Language", M. Sloman, E. Lupu and J. Lobo (Editors), *Policies for Distributed Systems and Networks*, Vol. 1995 of *Lecture Notes in Computer Science*, pp. 18–38, Springer Berlin Heidelberg, 2001.

82. Sensoy, M., T. J. Norman, W. W. Vasconcelos and K. Sycara, "OWL-POLAR: A Framework for Semantic Policy Representation and Reasoning", *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol. 12–13, pp. 148–160, 2012.

83. Bradshaw, J. M., S. Dutfield, P. Benoit and J. D. Woolley, "KAoS: Toward an Industrial-strength Open Agent Architecture", J. M. Bradshaw (Editor), *Software agents*, pp. 375–418, MIT Press, 1997.

84. Kafalı, Ö., A. Günay and P. Yolum, "$\mathcal{PROTOSS}$: A Run Time Tool for Detect-

ing $\mathcal{PR}$ivacy vi$\mathcal{O}$la$\mathcal{T}$ions in $\mathcal{O}$nline $\mathcal{S}$ocial network$\mathcal{S}$", *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pp. 429–433, 2012.

85. Kafalı, Ö., A. Günay and P. Yolum, "Detecting and Predicting Privacy Violations in Online Social Networks with $\mathcal{PROTOSS}$", *Distributed and Parallel Databases*, 2013, In Press.

86. Singh, M. P., A. K. Chopra and N. Desai, "Commitment-Based Service-Oriented Architecture", *IEEE Computer*, Vol. 42, No. 11, pp. 72–79, 2009.

87. Desai, N., A. K. Chopra, M. Arrott, B. Specht and M. P. Singh, "Engineering Foreign Exchange Processes via Commitment Protocols", *IEEE International Conference on Services Computing (SCC)*, pp. 514–521, 2007.