A Symbolic Exploration of the Joint State Space and the Underlying Argumentation-based Reasoning Processes for Multiagent Planning

by

Yuqing Tang

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2012

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Simon Parsons

Date

Chair of Examining Committee

Theodore Brown

Date

Executive Officer

Simon Parsons

Sergei Artemov

Samir Chopra

Timothy J. Norman (University of Aberdeen)

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

A Symbolic Exploration of the Joint State Space and the Underlying Argumentation-based Reasoning Processes for Multiagent Planning

by

Yuqing Tang

Advisor: Simon Parsons

Coming up with coherent behaviors for a team of agents in a non-deterministic environment is a complex process. The problem is further complicated by information regarding the environment being defeasible — new information will disqualify the old information — while at the same time this information is distributed, uncertain and possibly inconsistent.

In this dissertation, I propose an approach based on symbolic model checking techniques to implicitly explore the state space of a system of agents to form a coherent set of joint behaviors for the agents with uncertainty captured as non-determinism in the state space. The exploration process is based on individual agents' defeasible factored information about the dynamic of the state space towards a set of possibly inconsistent goals. This process can be interpreted using an argumentation-based approach. The exploration of the state space and the argumentation-based reasoning processes are carried out by a sequence of logical operations in the logic of Quantified Boolean Formulae (QBFs) and its model representation — Binary Decision Diagrams (BDDs). All the algorithms to carry out these processes can be computed with a polynomial number of QBF operations in terms of the number of inputs and the number of boolean variables in the domain. Although general symbolic solving QBFs and BDDs is PSPACE complete, the practices

of QBF solvers and BDD implementations allow us to employ various techniques and application-specific heuristics to solve a number of applications. In summary, the approaches proposed in this dissertation provide a formal system and implementation techniques to integrate non-monotonic reasoning and non-deterministic planning into defeasible multiagent decentralized planning so that coherent behaviors can be formed for a system of agents in a very demanding setting where the information regarding the environment and goals regarding what to do are distributed, incomplete, possibly inconsistent and uncertain.

Acknowledgments

I need to deeply thank my mentor, Professor Simon Parsons, for his generous, patient and continuous help in both my research and financial support in my PhD study. Without him I won't be able to know the wonderful research area named Autonomous Agents and Multiagent Systems; without him, I won't be able to conduct research in the field and publish a good amount of papers in prestigious conferences. Thanks need to go to my committee members, Professor Sergei Artemov, Professor Samir Chopra and Professor Timothy J. Norman for their supervision of this research work. Part of this dissertation is based on joint work with Professor Norman for a research project within the International Task Alliance in Network and Information Science. Thanks need to go to Professor Elizabeth Sklar for her guidance and financial support on the research of agent-based simulation. Thanks need to go to Professor Victor Pan for his introducing me to the research of polynomial root finding and matrix eigen problems starting by a coursework and ending with many good quality publications with my name in the author list although my contribution is a minor one. The research experience in matrix computation encourages me to keep my eyes open and contributes indirectly to many ideas leading to this dissertation. I also need to thank Dr. Felipe Meneguzzi for his comments and suggestions on AI planning formalisms. The joint work with Dr. Meneguzzi on probabilistic HTN planning which can potentially be combined with the work in this dissertation to create a new probabilistic version of argumentation for planning. I also need to thank two of my best friends at the CUNY Graduate Center's computer science department, Mr. Wenbo Cao and Mr. Ou Liu, for a large amount of inspiring time we spent together discussing

statistical machine learning, algorithms and complexity analysis, and many brain teasers. This thank list is certainly too short — thanks also go to all the people who have offered their help to me. Most importantly, I own tremendous thanks to my parents, my sister and brother for their strong support during the time of my PhD study.

Contents

List of Definitions

List of Propositions

List of Algorithms

List of Examples

List of Figures

Chapter 1

Introduction

As of today, more and more computers in different forms are being used to aid us in almost every aspect of our life and work. These computers can be in the form of a desktop or a laptop computer, in the form of a mobile phone or a personal digital assistant, in the form of a robot, a vehicle that drives itself, or a space craft controlled by computers. The functions of these computers range from helping us store and manage schedules, to collecting information and generating reports for us, to controlling safety-critical devices (e.g. the space craft, nuclear plants, and autonomous driving vehicles). As computers are getting involved in human life more deeply and more broadly, the tasks they are handling for human are getting more and more complex. At the same time, the social aspects of these computers, arising from the social nature of their human owners and from the decomposition approach used in complex problem solving, play an even more important role in the system.

Autonomous Agents and Multiagent Systems (*AAMAS*) is a research area addressing the complexity and the social nature of these computer systems. Individual computer systems are referred to as intelligent autonomous agents. An *agent* is a computer system (or a physical entity controlled by a computer system, such as a robot) that is capable of acting independently on behalf of its user or owner. In other words, given its preset design objectives, an agent can decide what actions to take

and then execute them without being told explicitly what to do at any given moment. A *multiagent system* (*MAS*) is one that consists of a number of agents, which interact with one another, typically by exchanging messages through some computer network infrastructure [Wooldridge, 2002].

The most important issue of MAS research is dealing with the problem of ensuring that overall problem-solving is effective and coherent [Sycara, 1998]. The solutions to this issue can be broken down into agent specifications — specifying beliefs, goals and the desired properties of agents, planning — deciding how these goals might best be achieved, and coordination — coordinating a team of agents for effective and coherent team behaviors. This dissertation will contribute to the theoretical and algorithmic aspects in agent specifications, planning and coordination for multiagent systems in which the problem-solving behaviors are effective and coherent with respect to an argumentation theory.

1.1 Background

1.1.1 Agent theories and agent architectures

Agent theories and agent architectures are two key issues in AAMAS research. An *agent theory* is a specification for an agent. An *agent architecture* is a methodology for building agents that can satisfy an given agent theory.

Typically, an agent theory develops formalisms to represent the properties of agents [Wooldridge and Jennings, 1995]. An agent theory usually has a mental component which is concerned with agents' mental attitudes [Levesque, 1984; Bratman, 1990; Cohen and Levesque, 1990; Rao and Georgeff, 1991]. Wooldridge and Jennings [1995] classified these mental attitudes into *information attitude*s, such as beliefs and knowledge, and *pro-attitude*s, such as desire, intention, obligation, commitment, choice and so on. Overall, Wooldridge [1997] identified the following properties of an agent:

- autonomy: agents encapsulate some state (that is not accessible to other agents), and make

decisions what to do based on this state, without the direct intervention of humans or others;

- reactivity: agents are *situated* in an environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the internet, or perhaps many of these combined), are able to *perceive* this environment (through the use of potentially imperfect sensors), and are able to respond in a timely fashion to changes that occur in it;

- pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by *taking the initiative*;

- social ability: agents interact with other agents (and possibly humans) via some kind of *agent-communication language*, and typically have the ability to engage in social activities (such as cooperative problem solving or negotiation) in order to achieve their goals.

Agent architectures are concerned with how an agent theory can be implemented — how agents, can be divided into components and how these components can work together to perceive the environment, reason, and determine how to act. Many architectures have been proposed for individual agents: perhaps the most important are logic-based agents, Belief-Desire-Intention (BDI) agents, reactive/hybrid agents, and decision-theoretic planning agents. A logic-based agent [Genesereth and Nilsson, 1987] symbolically represents its knowledge about the external world and actions using a logical language, then uses an automatic theorem prover to conclude what actions to take given the goals. A Belief-Desire-Intention (BDI) deliberative agent [Rao and Georgeff, 1991], extends the logical agent with the concepts of belief, desire and intention, uses these concepts to model the selection of intentions of what to do, then uses mean-ends analysis [Fikes and Nilsson, 1971] to produce the actions which can achieve the selected intentions. A reactive agent [Brooks, 1986] reacts to the stimulus of the environment obeying a specification which governs what to do given the agent's raw sensor input from the environment. A hybrid agent [Ferguson, 1992] combines both the logical architecture and the reactive architecture to advocate long term behaviors with the logical architecture and short term reactive behaviors with the reactive architecture. A

decision-theoretic planning agent [Blythe, 1999] incorporates probability and utility measurements on the states, actions and their results, then tries to produce a policy by which the agent can maximize its expected utility (or some variant thereof).

1.1.2 Action theories and AI planning

In parallel to the agent theories and architectures, the AI literature contains a wide range of action theories and practical planning techniques. Action theories [Thielscher, 2000; McCarthy, 1963; McCarthy and Hayes, 1969; Kowalski and Sergot, 1986; Reiter, 2001; Vo and Foo, 2005] are concerned about specifying actions and drawing conclusions from the specifications. Practical planning techniques [Fikes and Nilsson, 1971; Chapman, 1987; McAllester and Rosenblitt, 1991; Soderland and Weld, 1991; Penberthy and Weld, 1992; Erol et al., 1994; Cimatti and Roveri, 2000; Cimatti et al., 2003] are concerned about constructing plans for achieving goals.

The development of action theory focuses on solving three fundamental problems: the frame problem [McCarthy and Hayes, 1969], the qualification problem [McCarthy, 1977] , the ramification problem [Lin and Reiter, 1994]. The frame problem refers to the need to specify what does not change from situation to situation as the result of actions. The qualification problem refers to the need to specify precisely the conditions under which the effects of an action can take place. The ramification problem refers to the need to specify precisely the indirect effects of an action derived from the dependencies among various aspects of the world. The difficulty of these three fundamental problems lies in the fact that the amount of frame specifications, qualification conditions and ramification effects, as well as their possible combinations, can be huge or even infinite. By definitions, it is also unavoidable to have inconsistent combinations on these frame specifications, qualification conditions and ramification effects. The kind of inconsistency can be very complicated. For example, a qualification condition might disqualify some action effects so as to disqualify some ramification effects; as a result, the disqualified ramification effects might re-establish some frame specifications of the action. This implies that solving these three fundamental problems together inherently requires non-monotonic reasoning [Vo and Foo, 2005].

AI Planning techniques are concerned with the capability of agents looking for a plan to achieve some desired state of the world in the long run. A plan is captured by a data structure over a set of actions of the agents so that the desired state of world can be achieved. Starting with STRIPS [Fikes and Nilsson, 1971], classical planning has used a state space model. The world is modeled as being in states with a factored representation — by a set of formulae in a logical language or by a collection of variables, and actions (operators) are available to change the world from states to states. The planning problem is then to look for a sequence of actions that can lead the agent towards the desired goal states from an initial state. Various representation languages [Fikes and Nilsson, 1971; Ghallab et al., 1998], algorithms [Fikes and Nilsson, 1971], and semantics [Lifschitz, 1987] have been studied within the state space setting.

Another planning setting is plan space planning [Chapman, 1987; McAllester and Rosenblitt, 1991; Soderland and Weld, 1991; Penberthy and Weld, 1992] in which the planning processes can be abstracted as refining a given partial plan into a detailed plan that can achieve the desired goals. A partial plan is usually composed of an ordering over the actions along with additional logical constraints over the actions. The representation of a partial plan essentially captures a collection of plans instead of one specific plan. A plan space planning process then starts with a general partial plan, then refines the plan incrementally into more detailed partial plan(s) until an executable plan (such as a sequence of actions or a detailed partial plan which will be refined during the execution) that can achieve the desired goals is reached.

While state space planning and plan space planning deals with deterministic actions — executing an action in a state results in a deterministic state — recent developments in planning consider non-deterministic or probabilistic domains and producing plans that can take into account sensing information during the planning. Two new trends are non-determinsitic planning [Cimatti and Roveri, 2000; Cimatti et al., 2003] and decision theoretical planning [Boutilier et al., 1999]. Non-determinsitic planning generates a state-action table for an agent to behave in a reactive manner in a non-deterministic state transition system towards a set of goal states (see Chapter 3 for details). Decision theoretic planning also generates a state-action table for an agent to behave in

a reactive manner but in environments that are stochastic and where the states or changes can be characterized by rewards. The goal of the state-action table is to produce behaviors that can maximize the expected overall rewards in the stochastic environment.

When planning meets multiagent systems, the problems of distributed information and coordination arise. The problem of distributed information requires solving the problems of information integration and inferences. The problem of coordination requires solving the problems of action interactions and communication. The problems related to distributed information are usually being investigated by looking at the characteristics of interaction situations (scenarios) [Jennings, 1996]. The problem of coordination is usually investigated by looking at the interactions between actions [Decker and Lesser, 1995]: the choices of action combinations that can affect the overall outcomes, the order of the actions, and the resources and time constraints over action combinations and their orders.

Various approaches have proposed to solve these problems such as TÆMS [Decker and Li, 2000], GPGP [Durfee and Lesser, 1991] and DEC-POMDPs [Bernstein et al., 2000; Petrik and Zilberstein, 2009]. TÆMS[Decker and Li, 2000] is the data structure used by GPGP [Durfee and Lesser, 1991]. TÆMS is a blackboard data structure to model the global constraints, such as resources, time, utilities and so on. Such a data structure is shared by all the agents. [Lesser et al., 2004] uses a goal hierarchy (an HTN style knowledge) and partial ordering planning to guide the propagation of the shared constraints recorded in the TÆMS data structure as to solve the underlying complicated constraint optimization problems. DEC-POMDPs [Bernstein et al., 2000; Petrik and Zilberstein, 2009] extend decision theoretical planning of single agent [Boutilier et al., 1999] to multi-agent planning. To the best of my knowledge, the DEC-POMDPs approach does not consider communication among the agents. Instead, the approach models interactions among the agents the same as the interactions between the agents and the environment. It models both types of interactions as an observation problem in the environment using Decentralized Partial Observable Markov Decisions Processes (DEC-POMDPS).

1.1.3 Argumentation theory

Argumentation-based reasoning mechanisms can be viewed as a mathematical abstraction of the dialectical principles which are extracted from the study of human argumentation (Walton and Krabbe [1995], Hamblin [1970], and Toulmin [1958]; Toulmin et al. [1984]). The input to the reasoning scheme is knowledge represented in a formal language and sometimes an additional notion of preference over the knowledge, which can be represented inside or outside the formal language; its output is a characterization of the arguments and their conclusions. The two key components of an argumentation system are the process to construct arguments and the process to analyze the conflict pattern between these arguments. In the argument construction process, reasoning from the knowledge database to conclusions is recorded into a syntactic structure called an argument structure; then in the conflict analysis process, the conflicts between arguments will be captured and analyzed to characterize the acceptability of arguments and their conclusions. A set of arguments is *conflict-free* if there are no conflicts between any two arguments within this set. A set of arguments is collectively acceptable if any argument which defeats an argument in the set is defeated by an argument in this set. The notion of collective acceptability exerts a form of stability that can resist to external challenges. The most interesting property of argumentation based reasoning is the concept of "external stability" [Dung, 1995] through which a set of coherent beliefs and reasons is characterized by the relations between the arguments "internally" supporting the beliefs and reasons and the arguments "externally" supporting the contradictory beliefs and reasons. The concept of "external stability" is proved to correspond to the solution concept of N-person game [Dung, 1995]. This makes argumentation-based reasoning an attractive inference engine to organize the information on actions and goals in multiagent systems.

For reasoning about action theories, [Vo and Foo, 2005] proposed an argumentation theory that could solve the three fundamental problems in action theories: the qualification problem, the ramification problem and the frame problem. In [Vo and Foo, 2005], the assumption based argumentation framework [Bondarenko et al., 1997] (see Section 7.4.3) is used to carry out defeasible

reasoning over an action theory. The action theory is represented in a formal language is proposed with explicit symbols to represent the applicability of the qualification and frame conditions, and explicit symbols to represent time and duration. [Vo and Foo, 2005] then applied preferred extension semantics (see Section 7.4.1) to characterize the plausible outcome of the defeasible action theory.

For reasoning about how to act using argumentation theory, Atkinson and Bench-Capon [2007] proposed a presumptive argumentation system for practical reasoning based on argumentation schemes [Walton, 1996]. In [Atkinson and Bench-Capon, 2007], an alternative state transition system is used to underpin the argumentation system. States are explicitly referred to in the language used in the argumentation schemes [Walton, 1996]. Argumentation schemes are a set of distinct structures that can be instantiated to match against knowledge and belief input to generate argumentation. It consists of argument schemes, which are a set of abstract structures (or templates) that capture the reasoning (either deductive or non-deductive reasoning) within an argument, and critical questions (defeat schemes), which are a set of abstract structures that capture defeat relations between arguments. Valuation of state aspects is incorporated into the language, and a qualitative approach of promoting and demoting values can be associated with the actions. However, the formalism doesn't handle non-deterministic state transitions. It doesn't explicitly account for how argumentation-based practical reasoning can be propagated through the state transition system, and it doesn't provide algorithms matching the argument schemes against the input and how to argue and make decisions in the alternative state transition system.

As far as planning is concerned, García et al. [2008] proposed an argumentation formalism for partial order planning (a form of plan space planning [Chapman, 1987]). The argumentation is based on defeasible logic programming [Garcia and Simari, 2004]. Defeasible rules are used to simulate the applicability and outcomes of the action operators (similar to that of Chapter 3), and to simulate the operations to add an action and promote and demote the order of an action. The planning begins with an empty sequence of action. Then a dialectical semantics, which is equivalent to the grounded semantics, is used to drive undefeated arguments for a plan. The actions in this formalism are deterministic, and the planning algorithms output a sequence of actions (or a partial

order of actions) that can achieve the goals with respect to the defeasible action theory.

1.2 Approach

This dissertation addresses the following challenges in constructing multiagent systems:

1. The acquired knowledge (either from human or from machine learning algorithms) about the agents' actions and the inter-agent interactions are distributive, inconsistent and uncertain.

2. The pre-set goals of the agents, as well as the intermediate generated goals during the execution, are also distributive, inconsistent and uncertain.

3. Given the above challenges, it is a complex process to plan coherent long-term behaviors with the necessary coordination for a system of agents leading to the pre-set objectives of these agents.

This dissertation will propose an argumentation-based reasoning mechanism implemented in symbolic model checking techniques to specify multiagent behaviors, explore the state space of these behaviors, plan a coherent joint policy that can lead to a mutually agreed understanding of their goals along with formal tools to analyze and coordinate the interaction of these agents.

This dissertation will focus on the aspects of distributive, inconsistent and uncertain information in multiagent systems and on how to coordinate the planning processes that are carried out by individual agents as well as on how to coordinate the resulting joint plans. Symbolic model checking techniques are used to implicitly explore the multiagent space and the reasoning processes behind the planning processes so that the problem of exploring all possible interactions among the agents, and the problem of exploring all possible paths in the multiagent state space, can be reduced to symbolically representing and manipulating the models of a set of logical formulae.

In this dissertation, individual agents combine relevant information coming from different agents into a coherent state transition model of the whole system. The combination is backed by arguments constructed from the information, and underpinned by argumentation-based reasoning. The

implicit arguments keep track of related information and the reasons why and how the states transform and how the agents control the state transitions. On the other hand, a set of joint goals are also being combined from a possibly inconsistent set of goals of individual agents underpinned by the argumentation-based reasoning semantics. The agents can then explore the joint state space to produce plans that can orchestrate coherent joint behaviors towards the joint goals.

As the number of coherent combinations of the information regarding the joint state space can grow exponentially with the amount of information input, symbolic model checking techniques are employed to implicitly explore the state space and the combinations of information that specify the state space. The specific symbolic model checking technique I employ is Binary Decision Diagrams (BDDs). BDDs [Bryant, 1986] are a system of composite hashed data structures that compactly represent the models of formal language as decision diagrams. With BDDs, most logical operations can be performed within a polynomial number of BDD operations in terms of the sizes of the BDDs that involve in the operations. The symbolic encoding of the information allows sets of states and their changes be explored simultaneously with a sequence of BDD operations. This is the case because the implementation of BDDs caches a fair amount of logical results with hash tables. These shared hashed data structures also reflect the fact that similar logical structures are shared across different states, state-actions, and state transitions.

The BDD-based implicit exploration of multiple agents' state space, as well as the set of coherent information combinations that produce the state spaces and the changes in the space, enables us to compute a policy that allows the agents to direct the external world towards its desired outcomes. To ground all these algorithmic procedures in combining information about the world models and goals and composing agent behaviors, I have also developed semantics for these algorithms and combination procedures in terms of argumentation-based reasoning. BDD-based symbolic techniques enable us to make an implicit exploration of the space of coherent extensions of arguments that can be produced from the pieces of information that are held by individual agents. This essentially produces arguments regarding the external world dynamics and goals. The inconsistency in the information regarding the world dynamics and goals is then captured by concepts of defeat in the

argumentation theory. With the semantics of multiple preferred acceptable argument extensions, the algorithms based on symbolic techniques reorganize the information into multiple possibilities and plan policies (compactly encoded in BDDs) that can handle all these possibilities.

1.3 Thesis Contributions

This dissertation has following major contributions:

1. **Symbolic techniques for multiagent state space — the model, decentralized planning and interaction analysis** [Chapter 4]

   Symbolic techniques are introduced to explore the multiagent state space and analyze the interaction among the agents in this state space. Planning solution concepts for a single agent are extended into multiagent systems. In the same formalism, inter-agent interaction and coordination in multiagent state space can be analyzed efficiently using symbolic techniques.

2. **Defeasible factored action theory and its symbolic implementation** [Chapter 5]

   A defeasible factored action theory is proposed to specify the state space for individual agents and a multiagent system as a whole. Symbolic model checking techniques are employed to enable the underlying defeasible reasoning mechanism to explore all possible combinations of these information for all possible states simultaneously in polynomial number of symbolic model checking operations. All these efforts lead to mutual agreement among the agents on the most plausible joint understanding of the multiagent state space.

3. **Non-deterministic decentralized multiagent planning and coordination on defeasible factored information** [Chapter 6]

   A decentralized planning system is proposed to make use of the non-monotonic defeasible factored information, and this system is able to plan for multiple inconsistent goals simultaneously through symbolic model checking techniques. A multiagent coordination mechanism

is also created to execute the planned policy with the capability to resolve conflicting prescriptions of actions with respect to a power relationship among the agents.

4. **Argumentation theory for defeasible reasoning about actions, planning and execution of planning** [Chapter 7]

   The defeasible factor action theory reasoning procedures and the non-deterministic planning procedures are re-formalized to be underpinned by an argumentation theory. To the best of the author's knowledge, this is the first work which can provide a well-defined argumentation-based reasoning semantics for non-deterministic single-agent planning as well as non-deterministic multiagent planning.

5. **Introducing formalisms and techniques from other areas to AI planning, argumentation, and MAS research**

   Previously symbolic model checking techniques have been proposed in AI planning [Cimatti et al., 2003] and agent program verification [Wooldridge et al., 2006]. This dissertation advances further to bridge symbolic checking techniques into information algebra [Kohlas, 2003a] which can provide a well-defined mathematical framework for understanding multi-agent behaviors [see Chapter 2]. Symbolic techniques (Quantified Boolean Formulae and Binary Decision Diagrams in particular) are employed to enable the implicit exploration of a large number of argumentation for the planning processes all together simultaneously [see Chapter 7].

In action theories, many formalisms have been proposed for single agents, including default logic [Reiter, 1980, 2001], circumscription [McCarthy, 1987], defeasible reasoning and argumentation [Simari et al., 2004; García et al., 2008; Vo and Foo, 2005], and so on (see [Vo and Foo, 2005] for a comprehensive survey of action theories). To the best of my knowledge, very little work has been done on applying non-monotonic action theories for multiagent systems and on building non-deterministic planning for non-monotonic action theories. This dissertation will fill in the gap by constructing a defeasible action theory for multiagent systems.

The work present in this dissertation echos the mental attitudes in agent theories and the hybrid agent architectures but in a defeasible reasoning manner. The defeasible specifications (Chapter 5) implements a concept of information attitudes (i.e. beliefs on the situation, beliefs on the actions and their outcomes in the environment). Chapter 6 presents algorithms that can reason about a form of pro-mental attitudes in the agent theory [Wooldridge and Jennings, 1995] — inconsistent goals within individual agents and distributed across agents — by propagating these goals through dynamics of the world specified by the agents' action theories. From the agent architecture point of view [Wooldridge and Jennings, 1995], the defeasible multiagent action theory acts as a building block for a deliberative agent to reason about the actions, their applicable conditions and their outcomes. On the other hand, the multiagent policy produced by non-deterministic planning algorithms in Chapter 6 can be viewed as producing reactive rules for the agents. The the underlying meaning of competing specifications resemble the competing rules that are used in reactive agent architectures and hybrid architectures, but instead of having competing rules specifying which actions to take, in this defeasible action theory the specification identifies which changes in the world are reasonable given the agents' beliefs. Based on reasoning with the defeasible action theory, planning algorithms will then generate the reactive rules in the form of a policy to control the agents towards a mutually agreed plausible view of their long term goals.

On argumentation theory for actions and planning, the defeasible action theory proposed in this dissertation can be viewed as an automatic reasoning mechanism to pick up the reasoning that can match the argument schemes and critical questions proposed in [Atkinson and Bench-Capon, 2007] without specifying them explicitly as argument schemes. The defeats among the arguments are also automatically picked up based on the inconsistency in the underlying model theory instead of specifying the critical questions explicitly as in [Atkinson and Bench-Capon, 2007]. One future direction of this work is to apply the argumentation schemes as structural heuristics to direct the automatic symbolic model checking based argumentation to improve the efficiency.

Regarding the argumentation approach to action theories, Vo and Foo [2005] only considered one preferred extension. This dissertation takes into account all the preferred extensions that can

be the outcome of the argumentation on the action theories, and translates the multiple preferred extensions in non-deterministism of the state space underlying the action theories. The planning algorithms (Chapter 6) makes another major difference between this work and that of [Vo and Foo, 2005]. [Vo and Foo, 2005] is mainly concerned about defeasible reasoning on actions not about planning. Also [Vo and Foo, 2005] has the argumentation based on a very expressive language, the deductive system needed by assumption based argumentation framework [Bondarenko et al., 1997], which will prevent it from being an implementable tool and so making it only a tool for theoretical analysis. In contrast, the fact that all algorithms in this dissertation are bounded by polynomial number of BDD operations makes my approach tractable if the sizes of the encountered BDDs are tractable (see Section 2.2.1 for a discussion of BDD complexity), and thus useful as the basis of practical tools.

As far as planning is concerned, the actions in the formalism of [García et al., 2008] are deterministic, and the planning algorithms output a sequence of actions (or a partial order of actions) that can achieve the goals with respect to the defeasible action theory. The work in this dissertation differs from that of [García et al., 2008] in the following two aspects:

- argumentation formalisms are provided to explicitly capture reasoning about one step actions, and policies as well as execution structures so as to capture more essential aspects of an action theory and its applications in a planning process, and
- multiple preferred extensions in argumentation semantics are translated into the non-determinism of the action theory (which can be interpreted into Kripke structures), and the planning algorithms and the argumentation framework are able to reason about this non-deterministism.

1.4 Outline

The rest of this dissertation will be organized as follows.

Chapter 2 and Chapter 3 will re-iterate the related formalisms and techniques that have already

been proposed in the literature. They will be adapted in a form that leads to the extensions in this dissertation. Chapter 2 provides the basic components of the information representation for the rest of this dissertation. Chapter 2 starts with Section 2.1 on the language of Quantified Boolean Formulae (QBFs), and Section 2.2 on Order Reduced Binary Decision Diagrams (ORBDDs or simply BDDs) that encode models of QBFs and implement all the logical operations of QBFs. Chapter 2 also covers a finite domain predicate language as a front-end language for a subset of QBFs/BDDs that can represent the application domains compactly (in Section 2.4). In Section 2.5, an algebra of information is introduced to explicitly characterize the information manipulations, such as projection, transportation, re-interpretation, that are useful for multiagent system modeling. Thereafter, in Chapter 3, the single agent state space and its factored representation is introduced. Along with the single agent state space, non-deterministic planning solution concepts and algorithms for a single agent are also introduced.

The contributions of this dissertation start with Chapter 4. In this chapter, the single agent state space is extended into multiagent agent state space along with its representation. Using the QBF/BDD representation, a formal analysis of partial views and projected views of the agents are provided. The single agent planning algorithms are extended into a decentralized planning algorithm based on individual agents' partial views of the world. The work in this chapter is derived from my previously publications [Tang and Parsons, 2005; Tang et al., 2009]. In Chapter 5, the information regarding the multiagent state space is further modularized into a form of factored defeasible information. An ad-hoc approach is used to combine the factored information distributed across agents into a coherent model of a multiagent state space. The ad-hoc approach is based on the principle of looking for maximally consistent combinations of information, and mutually inconsistent combinations are taken as alternative dynamics of the state space. At the same time, existing non-deterministic planning algorithms are extended to handle the combination of the inconsistent factored information. The work in this chapter is derived from my previous publications [Tang and Parsons, 2005; Tang et al., 2010b; Tang, 2010]. In Chapter 6, a decentralized planning system is proposed to make use of the non-monotonic defeasible factored information, and this system

is able to plan for multiple inconsistent goals simultaneously through symbolic model checking techniques. A multiagent coordination mechanism is also created to execute the planned policy with the capability to resolve conflicting prescriptions of actions with respect to a power relationship among the agents. In Chapter 7, the ad-hoc approach of combining factored information is backed by a formal theory of argumentation-based reasoning. Argumentation-based reasoning captures the reasons and counter-reasoning for the combinations and the propagation of these reasons through the dynamics of state space to form a coherent extensions for the dynamics of the state space and the plans that can be produced for the state space. The work in this chapter is derived from my previous publications [Tang and Parsons, 2005; Tang et al., 2010b].

In the end, Chapter 8 discusses related work, and Chapter 9 concludes this dissertation and outlines future directions.

Chapter 2

Information Representation and Manipulation

This chapter gives the technical background of this dissertation. It covers *Quantified Boolean Formulae* (QBFs), *Binary Decision Diagrams* (BDDs), *Information Algebras* and the QBF/BDD encoding of a *finite domain predicate language*. The language of QBFs is a standard propositional language extended with quantifications over propositional variables. A Binary Decision Diagram (BDD) is a rooted directed acyclic graph with two terminal nodes labeled by boolean constants TRUE (1) and FALSE (0) respectively. A path from the root of a QBF to a terminal node (TRUE or FALSE) encodes a truth assignment to the propositional variables of the QBF (this assignment makes the QBF true or false respectively). The finite domain predicate language is a front-end language to represent and manipulate the information that can be expressed in QBFs/BDDs. The information algebra is used in this dissertation as a meta mathematical tool to talk about how the information are expressed and transformed.

2.1 Quantified Boolean Formulae

A propositional language $\mathcal{L}$ based on a set of proposition symbols $\mathcal{P}$ with quantifications can be defined by allowing standard connectives $\wedge, \vee, \rightarrow, \neg$ and quantifiers $\exists, \forall$ over the proposition variables. The resulting language is a logic of quantified boolean formulae (QBFs) [Bryant, 1992].

Definition 2.1 (Quantified Boolean Formulae (QBFs)). *Given a set $\mathcal{P}$ of propositional variables, a language of QBFs, denoted by $\mathcal{L}(\mathcal{P})$, is the set of formulae such that*

- *every propositional variable $x_i \in \mathcal{P}$ is a formula in $\mathcal{L}(\mathcal{P})$,*
- *if $\phi \in \mathcal{L}(\mathcal{P})$ and $\theta \in \mathcal{L}(\mathcal{P})$, then $\neg\phi$, $\phi \wedge \theta$, $\phi \vee \theta$, and $\phi \rightarrow \theta$ are QBF formulae in $\mathcal{L}(\mathcal{P})$, and*
- *if $\phi \in \mathcal{L}(\mathcal{P})$ and $x_i \in \mathcal{P}$, then $\exists_{x_i}\phi$ and $\forall_{x_i}\phi$ are formulae in $\mathcal{L}(\mathcal{P})$.*

If the set of propositional variables $\mathcal{P}$ is obvious from the context, $\mathcal{L}(\mathcal{P})$ will be simply denoted by $\mathcal{L}$.

A *symbol renaming operation*, which we use below, can be defined on $\mathcal{L}$, denoted by $\mathcal{L}[\mathcal{P}/\mathcal{P}']$, which means that a new language is obtained by substituting the symbols of $\mathcal{P}$ with the symbols of $\mathcal{P}'$ where $\mathcal{P}'$ contains the same set of propositions as that of $\mathcal{P}$ but uses different symbol names (notice that $|\mathcal{P}'| = |\mathcal{P}|$). Similarly for a formula $\xi \in \mathcal{L}$, if $\vec{x}$ is a vector of propositional variables for $\mathcal{P}$, then a variable renaming operation can be defined by $\xi[\vec{x}/\vec{x}']$ which means that all the appearances of variables $\vec{x} = x_1 x_2 \ldots x_n$ are substituted by $\vec{x}' = x'_1 x'_2 \ldots x'_n$ which is a vector of the corresponding variables or constants in $\mathcal{P}'$.

Example 2.1 (Symbol Renaming Operations). *Let $\mathcal{P} = \{x_1, x_2, x_3, x_4\}$ and $\phi = x_1 \wedge x_2 \vee \neg x_1 \wedge x_2$.*

$$\phi[x_1/x_3] = x_3 \wedge x_2 \vee \neg x_3 \wedge x_2$$

$$\phi[\langle x_1, x_2\rangle/\langle x_3, x_4\rangle] = x_3 \wedge x_4 \vee \neg x_3 \wedge x_4$$

In QBF, propositional variables can be universally and existentially quantified: If $\phi[\vec{x}]$ is a QBF formula with propositional variable vector $\vec{x}$ and x_i is one of its variables, the *existential quantification* of x_i in ϕ is defined as

$$\exists x_i \phi[\vec{x}] = \phi[\vec{x}][x_i/\text{FALSE}] \vee \phi[\vec{x}][x_i/\text{TRUE}]$$

and the *universal quantification* of x_i in ϕ is defined as

$$\forall x_i \phi[\mathbf{x}] = \phi[\vec{x}][x_i/\text{FALSE}] \wedge \phi[\vec{x}][x_i/\text{TRUE}].$$

Here FALSE and TRUE are two propositional constants representing "true" (1) and "false" (0) in the logic. Quantification over a set $X = \{x_1, x_2, \ldots, x_n\}$ of variables is defined as sequantial quantification over each variable x_i in the set:

$$\exists_X \xi = \left(\exists_{x_n} \left(\exists_{x_{n-1}} \left(\ldots \left(\exists_{x_1} \xi\right) \ldots\right)\right)\right)$$
$$\forall_X \xi = \left(\forall_{x_n} \left(\forall_{x_{n-1}} \left(\ldots \left(\forall_{x_1} \xi\right) \ldots\right)\right)\right)$$

Example 2.2 (QBF quantification). *Let* $\mathcal{P} = \{x_1, x_2\}$ *and* $\phi = x_1 \wedge x_2 \vee \neg x_1 \wedge x_2$.

$$\begin{aligned}
\exists_{x_1} \phi &= (\mathit{TRUE} \wedge x_2 \vee \neg \mathit{TRUE} \wedge x_2) \vee (\mathit{FALSE} \wedge x_2 \vee \neg \mathit{FALSE} \wedge x_2) \\
&= (x_2 \vee \mathit{FALSE}) \vee (\mathit{FALSE} \vee x_2) \\
&= x_2 \\
\forall_{x_1} \phi &= (\mathit{TRUE} \wedge x_2 \vee \neg \mathit{TRUE} \wedge x_2) \wedge (\mathit{FALSE} \wedge x_2 \vee \neg \mathit{FALSE} \wedge x_2) \\
&= (x_2 \vee \mathit{FALSE}) \wedge (\mathit{FALSE} \vee x_2) \\
&= x_2 \\
\exists_{x_2} \phi &= (x_1 \wedge \mathit{TRUE} \vee \neg x_1 \wedge \mathit{TRUE}) \vee (x_1 \wedge \mathit{FALSE} \vee \neg x_1 \wedge \mathit{FALSE}) \\
&= (x_1 \vee \neg x_1) \vee \mathit{FALSE}
\end{aligned}$$

$$
\begin{aligned}
&= TRUE \\
\forall_{x_2}\phi &= (x_1 \wedge TRUE \vee \neg x_1 \wedge TRUE) \wedge (x_1 \wedge FALSE \vee \neg x_1 \wedge FALSE) \\
&= (x_1 \vee \neg x_1) \wedge FALSE \\
&= FALSE
\end{aligned}
$$

The introduction of quantification doesn't increase the expressive power of propositional logic but allows us to write concise expressions whose quantification-free versions are exponentially large [Coudert and Madre, 1995].

Semantics of QBFs

Each QBF variable can be assigned a binary value: TRUE or FALSE. A set $\mathcal{P}$ of QBF variables can be assigned a vector of binary values which is called a truth assignment to $\mathcal{P}$.

Definition 2.2 (QBF semantics). *A QBF on $\mathcal{P}$ can be interpreted as a set of truth assignments (also called interpretations or models in the literature) $\mathcal{I} \subseteq 2^{\mathcal{P}}$ to the propositional variables. Each truth assignment $I \in \mathcal{I}$ is a function:*

$$I : \mathcal{P} \to \{TRUE, FALSE\}.$$

If ξ and ϕ are QBF formulae, and $I \in 2^{\mathcal{P}}$ is a truth assignment, then the model relation $\models$ can be defined recursively with respect to the QBF syntax:

- $I \models \xi$ *iff* $\xi \in \mathcal{P}$ *is a QBF propositional variable and* $I(\xi) = TRUE$,
- $I \models \neg\xi$ *iff* $I \not\models \xi$,
- $I \models \xi \wedge \phi$ *iff* $I \models \xi$ *and* $I \models \phi$,
- $I \models \xi \vee \phi$ *iff* $I \models \xi$ *or* $I \models \phi$,
- $I \models \xi \to \phi$ *iff* $I \models \neg\xi \vee \phi$,

- $I \models \xi \leftarrow \phi$ *iff* $I \models \phi \rightarrow \xi$,
- $I \models \xi \leftrightarrow \phi$ *iff* $I \models (\xi \rightarrow \phi) \wedge (\xi \leftarrow \phi)$,
- $I \models \exists_{x_i} \xi$ *iff* $I \models \xi[x_i = TRUE] \vee \xi[x_i = FALSE]$, *and*
- $I \models \forall_{x_i} \xi$ *iff* $I \models \xi[x_i = TRUE] \wedge \xi[x_i = FALSE]$.

Given a QBF $\xi \in \mathcal{L}(\mathcal{P})$ *of propositions, the set of truth assignments that can satisfy* ξ *is defined as*

$$\mathcal{I}(\xi) \stackrel{def}{=} \{I \in 2^{\mathcal{P}} | I \models \xi\}.$$

2.2 Binary decision diagrams

The set of QBFs and the logical operations over them can be represented and efficiently computed using a data structure called a Binary Decision Diagram (BDD) [Bryant, 1992].

A BDD for a formula ξ is a rooted directed acyclic graph with two terminal nodes. The terminal nodes are either TRUE or FALSE. The BDD node for a formula ξ starting with a variable x_i is a triple:

$$\langle x_i, low, high \rangle$$

where *low* and *high* are identifiers of the BDD nodes for $\xi[x_i/\text{FALSE}]$ and $\xi[x_i/\text{TRUE}]$ respectively. The identifier for a BDD node is typically an index of the array which holds all the BDD nodes or a pointer to the memory holding the BDD node in the implementation. The truth value of ξ can then be determined by traversing the graph from the root to the leaves following the boolean assignment given to the variables of ξ.

A special form of BDD, called a Reduced Ordered Binary Decision Diagram (ROBDD) [Bryant, 1992], is widely implemented. A ROBDD is a compact BDD which

- uses a fixed ordering over the variables from the root to the leaves,
- merges duplicate subgraphs into one, and

- directs all the incoming edges of the duplicate subgraph into the merged subgraph.

Following the notation traditionally used in symbolic model checking and AI planning, we will refer to an ROBDD simply as a BDD. A BDD example can found in Figure 2.1. It is the ROBDD representation of the QBF:

$$\varphi = (\neg x_0 \wedge \neg x_1 \wedge \neg x_2) \vee (x_0 \wedge x_1) \vee (x_1 \wedge x_2)$$

with variable ordering of x_0, x_1, x_2. In Figure 2.1, a path in the ROBDD from x_0 to 1 to a truth assignment to x_0, x_1, x_2 which can satisfy φ. For example, in Figure 2.1 a path from x_0 to 1 that is composed of the edges (x_0, x_1) labeled by 0, (x_1, x_2) labeled by 0, and $(x_2, 1)$ labeled by 0 encodes the truth assignment $I(x_0) = 0$, $I(x_1) = 0$ and $I(x_2) = 0$. This truth assignment satisfies φ.

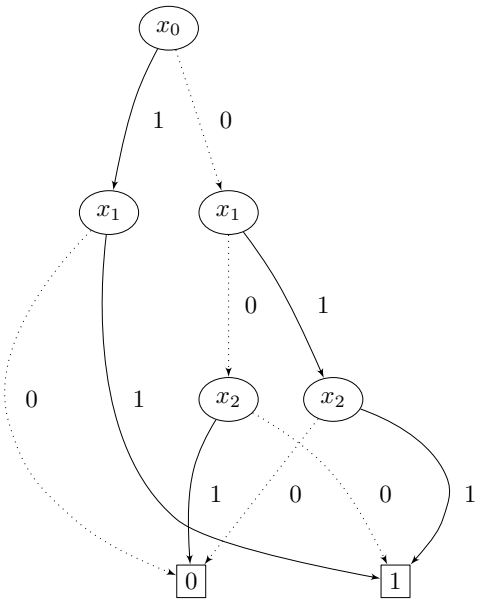

Figure 2.1: A BDD representation for $(\neg x_0 \wedge \neg x_1 \wedge \neg x_2) \vee (x_0 \wedge x_1) \vee (x_1 \wedge x_2)$

ROBDDs are canonical representations of QBFs. For any two QBFs, if they have the same set of models, then they will be represented by the same ROBDD. In other words, ROBDDs are canonical representations of the models (truth assignments) to QBFs.

QBF operator	BDD operator	Complexity
$\neg\xi$	$\neg G(\xi)$	$O(\|\|\xi\|\|)$
$\exists x_i(\xi)$	$G(\xi_{x_i=0}) \vee G(\xi_{x_i=1})$	$O(\|\|\xi\|\|^2)$
$\forall x_i(\xi)$	$G(\xi_{x_i=0}) \wedge G(\xi_{x_i=1})$	$O(\|\|\xi\|\|^2)$
$\xi_1 \wedge \xi_2$	$G(\xi_1) \wedge G(\xi_2)$	$O(\|\|\xi_1\|\| \cdot \|\|\xi_2\|\|)$
$\xi_1 \vee \xi_2$	$G(\xi_1) \vee G(\xi_2)$	$O(\|\|\xi_1\|\| \cdot \|\|\xi_2\|\|)$
$\xi_1 \rightarrow \xi_2$	$G(\xi_1) \rightarrow G(\xi_2)$	$O(\|\|\xi_1\|\| \cdot \|\|\xi_2\|\|)$
$\|X\|$	$Sat\text{-}count(G(\xi(X)))$	$O(\|\|\xi(X)\|\|)$

Table 2.1: The mapping between QBF operators and BDD operators. ξ, ξ_1, ξ_2 are formulae in QBF; $G(\xi), G(\xi_1), G(\xi_2)$ are BDD representations for these formulae.

2.2.1 Complexity issues of BDDs

The advantage of using BDDs to represent QBF formulae is that most basic operations on QBFs can be performed in linear or quadratic time in terms of the number of nodes used in a BDD representation of the formulae. Let ξ, ξ_1, ξ_2 be QBF formulae, let the number of nodes used in the BDD representation of a formula be denoted by $||\cdot||$. With this BDD representation, the complexity of a QBF binary operator $\langle op \rangle$ (e.g. $\wedge, \vee, \rightarrow$) on two formulae ξ_1 and ξ_2, namely $\xi_1 \langle op \rangle \xi_2$, is $O(||\xi_1|| \times ||\xi_2||)$, that of negation $\neg\xi$ is $O(||\xi||)$ (or $O(1)$ if complement edges are introduced to the BDDs), and the complexity of quantification $Q_{x_i}(f[\vec{x}])$, where Q is either $\exists$ or $\forall$, is $O(||f||^2)$ [Bryant, 1992; Coudert and Madre, 1995], as summarized in Table 2.1.

BDDs can be viewed as a form of deterministic finite automata [Burch et al., 1994] that encode a certain amount of information underlying the corresponding QBFs. A BDD with a vector $\vec{x}$ containing n variables is identified with a language composed of a set of strings in $\{0,1\}^n$ that corresponds to the set of truth assignments to $\vec{x}$ that make the BDD "true". Since this language is finite and all finite languages are regular, there is a minimal finite automaton that accepts this set of strings. A BDD with an optimal variable order can be viewed as a form of this minimal automata. Assuming the variable ordering is $x_1, \ldots, x_n$, the number of nodes labeled by the variable $x_i \in \vec{x}$ corresponds to the number of states that are required in the automaton to memorize the information encoded in the variables $x_1, \ldots, x_i$ for the language before reading the additional

information encoded in the remaining variables $x_{i+1}, \ldots, x_n$. This interpretation is closely related to the concept of Kolmogrov complexity of a set — the length of the shortest program which can output members of the set on a Turing machine and then halt [Nannen, 2003] (namely it is the length of the shortest description of the information embedded in the set) — while here the machine is restricted to finite automata, so it is not a surprise that there is research on using Kolmogrov complexity concepts to infer the best variable order for BDDs and their variants, for example [Oliveira and Sangiovanni-Vincentelli, 1996]. Viewing ROBDDs as minimal finite automata that encode the set of truth assignments also explains why BDD approaches are very efficient in many applications [Yang et al., 1998].

However, nothing is free. There are some cases in which the BDD representation requires exponential complexity for any variable orderings. In general, the QBF SAT problem is PSPACE hard [Ladner, 1977] and so are BDD operations in general. For example, the complexity lower bound of any BDD implementation of integer multiplication is $2^{n/8}$. For word sizes encountered in practice this is not very large (e.g. for 64 bit multiplication, the number is 256) [Bryant, 1986], but experiments indicate that the true bound is far worse: for a word size less than or equal to $n = 8$, the $2n$ output functions of a multiplier requires no more than 5000 vertices for a variety of different input orderings; for $n > 10$, some outputs require BDDs with more than $100,000$ vertices. There are various kinds of approaches to overcome this with respect to the application. For example in [Burch, 1991], specific to circuit verification, the author proposed a conservative method with complexity $O(n^3)$: it will never classify an incorrect circuit as correct, but it will possibly classify a correct circuit as incorrect.

There has been a lot of work on reducing the size of BDDs. Many successful approaches have been developed in literature, especially those developed for symbolic model checking in software and hardware verification [Jhala and Majumdar, 2009], and in non-deterministic AI planning [Cimatti et al., 2003]. Examples of techniques for reducing the size of BDDs are early quantification [Hojati et al., 1996], quantification scheduling [Chauhan et al., 2001], transition partitioning [Burch et al., 1991], iterative squaring [Burch et al., 1990; Cabodi et al., 1997], frontier simplification [Coudert

et al., 1989], input splitting [Meinel and Theobald, 1998; Moon et al., 2000], and state set A^* branching [Jensen et al., 2002b,a, 2008] (a BDD version of the A^* search heuristic [Pearl, 1984]).

Another factor affecting the BDD size greatly is variable ordering. The problem of finding an optimal variable ordering is NP-complete [Bollig and Wegener, 1996]. Algorithms based on dynamic programming [Drechsler et al., 1998], heuristics [Jain et al., 1998], dynamic variable reordering [Panda et al., 1994] and machine learning approaches [Grumberg et al., 2003] have been proposed for finding a good variable ordering in reasonable time[1].

Proposition 2.1 (Sequential BDD complexity). *If a BDD $G(\phi)$ can be computed by a sequence of K BDD operations, then the complexity to such a sequence of K BDD operations is bounded by $O(K \cdot M)$ where M is the maximum size of the BDDs that are used to hold the intermediate results in such a sequence of K operations.*

Proof. According to Table 2.1, for every unary and binary BDD operation over one BDD and two BDD operands respectively, the complexity is bounded by the size of resulting BDD which is in turn bounded by $O(M)$ [Bryant, 1986]. Therefore the complexity of a sequence of K BDD operation is then bounded by $O(K \cdot M)$.

Therefore from now on, I will use the number of BDD operations as an indicator of the complexity of algorithms based on BDD operations, and assume that the maximum size of intermediate BDDs will be reduced to a feasible through the approaches surveyed above and appropriate modifications and relaxation of the algorithms proposed in this dissertation that can fit the specific applications of interests.

[1] [Grumberg et al., 2003] is also a good source for other references on BDD variable (re-)ordering.

2.3 Encoding sets, relations and sets of sets

2.3.1 Encoding sets and relations

With QBFs, we can encode sets and relations into QBFs by representing the elements of sets and relations as truth assignments. In the QBF encoding of a set, each element of the set corresponds to a truth assignment to the QBF.

Let y be an element of a set Y. Let $\mathcal{P}(Y)$ be a set of propositional variables of size $\lceil \log(|Y|) \rceil$. y can then be explicitly encoded by a conjunction composed of all proposition symbols in $\mathcal{P}(Y)$ in either positive or negative form

$$\xi(y) = \bigwedge_{p_i \in \mathcal{P}(Y), y \models p_i} p_i \wedge \bigwedge_{p_j \in \mathcal{P}(Y), y \not\models p_j} \neg p_j$$

where $y \models p_i$ means that the corresponding bit p_i is set to be TRUE in the encoding of y, and $y \not\models p_j$ means that the corresponding bit p_j is set to be FALSE in the encoding of y.

Example 2.3 (Encoding set with QBFs/BDDs).
Let $Y = \{apple, orange, grape, pear\}$. Then we can have a set of encoding propositional variables $\mathcal{P}(Y) = \{b_0, b_1\}$, and encode the elements of the set as:

$$\xi(apple) = \neg b_0 \wedge \neg b_1$$
$$\xi(orange) = \neg b_0 \wedge b_1$$
$$\xi(grape) = b_0 \wedge \neg b_1$$
$$\xi(pear) = b_0 \wedge b_1$$

Then a set of elements can be characterized by a formula $\gamma \in \mathcal{L}$, with the set denoted by $Y(\gamma)$, where $Y(\gamma) = \{y | y \models \gamma\}$.

Example 2.4 (Encoding set operations with QBF/BDD operations).

Set operator	QBF operator
$Y_1 \cap Y_2$	$\xi(Y_1) \wedge \xi(Y_2)$
$Y_1 \cup Y_2$	$\xi(Y_1) \vee \xi(Y_2)$
$Y_1 \setminus Y_2$	$\xi(Y_1) \wedge \neg\xi(Y_2)$
$y \in Y$	$\xi(y) \rightarrow \xi(Y)$
$Y_1 \subseteq Y_2$	$\xi(Y_1) \rightarrow \xi(Y_2)$

Table 2.2: The mapping between set operators and QBF operators

Let $Y = \{apple, orange, grape, pear\}$ *and* $\mathcal{P}(Y) = \{b_0, b_1\}$*, then*

$$\xi(\{apple, orange\}) = (\neg b_0 \wedge \neg b_1) \vee (\neg b_0 \wedge b_1)$$

$$\xi(\{grape, pear\}) = (\ b_0 \wedge \neg b_1) \vee (\ b_0 \wedge b_1)$$

Two special sets, the empty set $\emptyset$ and the universal set $\mathcal{U}$, are represented by FALSE and TRUE respectively.

With this notion we can have a mapping between the set operations on states and the boolean operations on formulae as shown in Table 2.2 when Y_1 and Y_2 are interpreted as two sets of states[2]. The key achievement of using BDDs (and the front end language of QBFs) to represent sets and relations is that the complexity of the operations will depend on the complexity of the BDD representation instead of the size of the sets and relations, and the complexity of the BDD representation of the sets and relations doesn't depend on the size of those sets and relations. Instead, the operations on BDDs are polynomial in the size of the BDD, and so operations on sets and relations will be polynomial in the size of their BDD representation rather than exponential in their size.

2.3.2 Encoding sets of sets

Similar to the encoding of sets with QBFs, we can encode sets of sets and relations on sets of sets with QBFs. In the QBF encoding, we have the QBF variables correspond to elements of a set; a

[2]The minus sign "−" is used interchangeably with "\" for set difference operation.

set of elements corresponds to a truth assignment to the QBF.

Let $\mathcal{U} = \{y_1, \ldots, y_n\}$ be a universal set, and $\mathcal{P}_L(\mathcal{U})$ be a set of propositional variables of size $|\mathcal{U}|$ where each element y_i having a label variable $l_{y_i} \in \mathcal{P}_L(\mathcal{U})$, a subset $Y \subseteq \mathcal{U}$ can be encoding by a QBF

$$\xi(y) = \bigwedge_{y_i \in Y} l_{y_i} \wedge \bigwedge_{y_i \notin Y} \neg l_{y_i}$$

Example 2.5 (Encoding powerset with QBF/BDD). *Let* $Y = \{e_0, e_1, e_2, e_3\}$ *and* $\mathcal{P}_L(Y) = \{l_0, l_1, l_2, l_3\}$*, then*

$$\begin{aligned}
\xi(\{\{e_0\}\}) &= b_0 \wedge \neg b_1 \wedge \neg b_2 \wedge \neg b_3 \\
\xi(\{\{e_1\}\}) &= \neg b_0 \wedge b_1 \wedge \neg b_2 \wedge \neg b_3 \\
\xi(\{\{e_1, e_2\}\}) &= \neg b_0 \wedge b_1 \wedge b_2 \wedge \neg b_3 \\
\xi(\{\{e_0\}, \{e_1\}\}) &= (b_0 \wedge \neg b_1 \wedge \neg b_2 \wedge \neg b_3) \\
&\quad \vee (\neg b_0 \wedge b_1 \wedge \neg b_2 \wedge \neg b_3) \\
\xi(\{\{e_3\}, \{e_1, e_2\}\}) &= (\neg b_0 \wedge \neg b_1 \wedge \neg b_2 \wedge b_3) \\
&\quad \vee (\neg b_0 \wedge b_1 \wedge b_2 \wedge \neg b_3)
\end{aligned}$$

Relations on sets of sets can also be encoded. Let $\mathcal{P}_{L'}(\mathcal{U})$ be another set of propositional variables that label the elements in $\mathcal{U}$ which is a copy of $\mathcal{P}_L(\mathcal{U})$. A subset relation *subseteq* on $\mathcal{U}$ ($Y \subseteq Y'$) can be implemented by

$$subseteq(\mathcal{U}) = \bigwedge_{y_i \in \mathcal{U}} l_{y_i} \rightarrow l_{y'_i}$$

A proper subset relation $\subset$ on $\mathcal{U}$ ($Y \subset Y'$) can be implemented by

$$subset(\mathcal{U}) = subseteq(\mathcal{U}) \wedge \neg \bigwedge_{y_i \in \mathcal{U}} (l_{y_i} \leftrightarrow l_{y'_i})$$

Example 2.6 (Encoding subset relation with QBF/BDD). *Let* $U = \{e_0, e_1, e_2\}$, $\mathcal{P}_L(Y) = \{l_0, l_1, l_2\}$ *and* $\mathcal{P}_{L'}(Y) = \{l'_0, l'_1, l'_2\}$, *then*

$$subseteq(U) = (l_0 \rightarrow l'_0) \wedge (l_1 \rightarrow l'_1) \wedge (l_2 \rightarrow l'_2).$$

Combining the encoding of sets and sets of sets, the membership relation $\in$: $Y \times 2^Y$ (for every $y \in \mathcal{U}$ and every $Y \subseteq \mathcal{U}$ such that $y \in Y$) can also be encoded by:

$$\in (Y) = \bigvee_{y_i \in Y} (\xi(y_i) \wedge l_{y_i}).$$

Each term $(\xi(y_i) \wedge l_{y_i})$ encodes all pairs of the form $\langle y_i, Y \rangle$ such that $y_i \in Y$; the union of all these pairs then encodes the membership relation between individual elements and the subsets composed of these elements. $\in (Y)$ is also denoted by $\in (\mathcal{P}(Y), \mathcal{P}_L(Y))$ when the QBFs that encode them are given.

2.4 Encoding finite domain predicate language

While QBFs and BDDs provide an efficient mechanism to compute the reasoning, predicate languages are widely used in knowledge representation as the allow structures in propositions, making it more convenient to represent properties and relations. In this section, I will introduce a limited version of predicate language which is defined on finite domains and is equivalent to propositional logic in expressiveness.

2.4.1 Predicate language syntax

Definition 2.3 (Finite domain predicate language). *A finite domain predicate language* $\mathcal{L}_{\mathsf{Pred}}$ *is defined on a triple (called the signature of the predicate language* $\mathcal{L}_{\mathsf{Pred}}$*)*

$$\langle \mathsf{Pred}, \mathsf{VAR}, \Omega \rangle$$

where Pred *is a finite set of predicate symbols,* VAR *is a finite set of variable symbols, and* Ω *is a finite domain of constants. Each symbol* $P_j \in \mathsf{Pred}$ *is associated with an arity* n_j*. The formulae of the finite domain predicate language* $\mathcal{L}_{\mathsf{Pred}}$ *is defined recursively as follows:*

- *A variable* $v \in \mathsf{VAR}$ *is a term,*
- *A constant* $c \in \Phi$ *is a term,*
- *If* $P_j \in \mathsf{Pred}$ *is a predicate symbol of arity* n_j *and* $t_1, t_2, ..., t_{n_j}$ *are terms, then* $P_j(t_1, ..., t_{n_j})$ *is a predicate and it is in the language* $\mathcal{L}_{\mathsf{Pred}}$*,*
- *If* $\phi \in \mathcal{L}_{\mathsf{Pred}}$ *and* $\theta \in \mathcal{L}_{\mathsf{Pred}}$*, then* $\neg\phi$*,* $\phi \wedge \theta$*,* $\phi \vee \theta$*, and* $\phi \rightarrow \theta$ *are predicate formulae in* $\mathcal{L}_{\mathsf{Pred}}$*, and*
- *If* $\phi \in \mathcal{L}_{\mathsf{Pred}}$ *and* $x \in \mathsf{VAR}$*, then* $\exists_x \phi$ *and* $\forall_x \phi$ *are in* $\mathcal{L}_{\mathsf{Pred}}$*.*

To further regulate the predicate languages, it is assumed that, in a predicate of the form $P_j(t_1, t_2, \ldots, t_{j_n})$ with n parameters, each parameter t_k can take a value in a domain $\Omega_{j,k} \subseteq \Phi$. The size of a domain $\Omega_{j,k}$ is denoted by $|\Omega_{j,k}|$. The union of domains of all parameters is Ω, namely $\Omega = \cup_{j,k}\Phi_{j,k}$. Note that it is possible for the domains of the two parameters' to intersect, e.g. $\Omega_{j,k} \cap \Omega_{i,g} \neq \emptyset$. For representational convenience, the domain of predicate variables can be extended with an invalid constant, denoted by NIL. Given a domain Ω, its *NIL-extended domain* Ω^- is defined as $\Omega \cup \{NIL\}$.

Definition 2.4 (Free variable). *We define a variable* x *to be a* free variable *in a formula as the following:*

- *x is a free variable in a predicate $P_j(t_1, ..., x, ...)$ (x is one of P_j's parameters);*
- *x is a free variable in a predicate $P_j(t_1, ...)$ (x does not appear in P_j's parameter list);*
- *x is a free variable in φ OP ψ if OP is either $\wedge, \vee, \rightarrow, \leftrightarrow$ and x is free in either φ or ψ,*
- *x is a free variable in $\neg\varphi$ if x is free in φ,*
- *x is a free variable in $Q_y\varphi$ (where Q is either $\exists$ or $\forall$) if $x \neq y$ and x is free in φ.*

Definition 2.5 (Predicate variable substitution). *If a variable x is a free in a formula φ, we define $\varphi[x/y]$ to be a substitute operation to replace all free appearances of x in φ into y.*

Definition 2.6 (Predicate variable domain). *The domain of a variable x in a formula ϕ, denoted by $\Omega(\phi, x)$, is defined recursively as follows:*

- *If x is in place of the parameter $t_{j,k}$ of a predicate $\phi \stackrel{def}{=} P_j(..., x, ...)$ and $t_{j,k}$'s domain is $\Omega_{j,k}$, then $\Omega(\varphi, x) = \Omega_{j,k}$.*
- *If x is a free variable that appears in $\phi \stackrel{def}{=} \neg\varphi$, then $\Omega(\phi, x) = \Omega(\varphi, x)$.*
- *If x is a free variable that appears in $\phi \stackrel{def}{=} \varphi$ OP ψ (where OP is either $\wedge$, $\vee$, $\rightarrow$, or $\leftrightarrow$, then*
  - *$\Omega(\phi, x) = \Omega(\varphi, x) \cap \Omega(\psi, x)$, if x appears free in both φ and ψ,*
  - *$\Omega(\phi, x) = \Omega(\varphi, x)$, if x appears free in φ but not in ψ, or*
  - *$\Omega(\phi, x) = \Omega(\psi, x)$, if x appears free in ψ but not in φ.*

2.4.2 Predicate language semantics

Adapting from [Brachman and Levesque, 2004], a formula in $\mathcal{L}_{Pred}$ can be interpreted into truth conditionals which are defined via the concepts of valuations and models.

Definition 2.7 (Valuation). *A* valuation *V for a set of variables* VAR *is a mapping from* VAR *to the domain Ω:*

$$V : \mathsf{VAR} \rightarrow \Omega.$$

By overloading the notion of valuation, a valuation function is also defined on Ω*. It maps a constant to itself, namely* $V(c) = c$ *for all* $c \in \Omega$*.*

Definition 2.8 (Predicate model). *A* model *is a pair* $M = \langle \Omega, I \rangle$ *where*

- Ω *is the domain of the predicate language,*
- I *is an interpretation that maps*
  - *every grounded predicate (whose parameters are constants in* Ω*)* $P_j(c_1, ..., c_{n_j})$ *to* $\{TRUE, FALSE\}$*. Namely* $I(P_j(c_1, ..., c_{n_j})) \in \{TRUE, FALSE\}$ *where* $c_1, ..., c_{n_j}$ *are constants in domains* $\Phi_{j,1}, ..., \Phi_{j,n_j}$ *respectively.*

In a finite domain predicate language $\mathcal{L}_{\mathsf{Pred}}$, if the domain Ω is fixed in the context, a model $M = \langle \Omega, I \rangle$ can be simplified into $M = \langle I \rangle$, namely a model is simply an interpretation (truth assignment) to the grounded predicates.

Definition 2.9 (Truth condition). *A* truth condition *is a pair* $\langle M, V \rangle$ *of model and valuation. Let any* ϕ *and* ψ *be formulae in* $\mathcal{L}_{\mathsf{Pred}}$*, the truth conditions of a formula can be defined recursively by:*

- $M, V \not\models FALSE$*,*
- $M, V \models P_j(t_1, t_2, ..., t_{n_j})$ *iff* $I(P_j(V(t_1), ..., V(t_{n_j}))) = TRUE$*,*
- $M, V \models t_i = t_k$ *iff* $V(t_i) = V(t_k)$*,*
- $M, V \models \neg\phi$ *iff* $M, V \not\models \phi$*,*
- $M, V \models \phi \wedge \psi$ *iff* $M, V \models \phi$ *and* $M, V \models \psi$*,*
- $M, V \models \phi \vee \psi$ *iff* $M, V \models \phi$ *or* $M, V \models \psi$*,*
- $M, V \models \phi \rightarrow \psi$ *iff* $M, V \models \neg\phi$ *and* $M, V \models \psi$*,*

- $M, V \models \phi \leftrightarrow \psi$ *iff* $M, V \models \phi$ *iff* $M, V \models \psi$,
- $M, V \models \exists_x \phi$ *iff* $M, V \models \phi[x/c]$ *for some* $c \in \Phi(x)$,
- $M, V \models \forall_x \phi$ *iff* $M, V \models \phi[x/c]$ *for all* $c \in \Phi(x)$.

Given a predicate language $\mathcal{L}_{\mathsf{Pred}}$, the set of all possible truth conditions are denoted by $\mathcal{TC}(\mathcal{L}_{\mathsf{Pred}})$. Given a formula $\phi \in \mathcal{L}_{\mathsf{Pred}}$,

- the set of truth conditions that can satisfy ϕ is denoted by

$$\mathcal{TC}(\phi) = \{\langle M, V\rangle \in \mathcal{TC}(\mathcal{L}_{pred}) \mid M, V \models \phi\},$$

- the set of models that can satisfy ϕ is denoted by

$$\mathcal{M}(\phi) = \{M \mid \text{ there exists a valuation } V \text{ such that } M, V \models \phi\},$$

- the set of valuation that can satisfy ϕ is denoted by

$$\mathcal{V}(\phi) = \{V \mid \text{ there exists a model } M \text{ such that } M, V \models \phi\},$$

With the above definitions, a predicate formulae can formally be interpreted into a set of *truth conditions*. These truth conditions are then mapped to the reality of the world.

2.4.3 Encoding predicate formulae in QBFs/BDDs

With the capability of encoding of sets and power sets, QBFs can be used to encode the semantics of a predicate language $\mathcal{P}_{\mathsf{Pred}}$. This can be done by encoding each truth condition $\langle M, V\rangle$ of a predicate formula ϕ by a truth assignment s to its corresponding QBF $QBF(\phi)$. To improve the efficiency of the encoding, I adapt the approach proposed in [Edelkamp and Helmert, 1999] (which is particularly focused on encoding planning domain predicates into QBFs).

Before introducing the encoding, we should differentiate two types of parameters in a predicate: single-valued parameters and multi-valued parameters.

Definition 2.10 (Predicate parameter types). *In a particular application, the parameters of a predicate $P_j(t_{j,1}, \ldots, t_{j,n})$ can be categorized into two types:*

- *$t_{j,k}$ is called a* single-valued parameter *if $t_{j,k}$ can only take one value in all satisfiable truth conditions in the application. Namely, in every truth condition $\langle M_i, V_i \rangle$ that can satisfy a scenario of the application, V_i assigns at most one constant in $\Omega_{j,k}$ to $t_{j,k}$ (either constants or variables) in all the appearances of P_j.*

- *$t_{j,k}$ is called a* multi-valued parameter *if $t_{j,k}$ can take multiple values in some satisfiable truth conditions in the application. Namely, in every truth condition $\langle M_i, V_i \rangle$ that can satisfy a scenario of the application, V_i may assign more than one constant in $\Omega_{j,k}$ to $t_{j,k}$ (either constants or variables) in all the appearances of P_j.*

The set of single-valued parameters of P_j is denoted by $T_S(P_j)$ and the set of multi-valued parameters of P_j is denoted by $T_M(P_j)$.

Before we can encode a formula in $\mathcal{L}_{\mathsf{Pred}}$, we need to identify whether a variable is a single-valued in the formula or not.

Definition 2.11 (Formula variable types). *Let x be a variable in $\mathcal{L}_{\mathsf{Pred}}$. We define x to be a* single-valued variable *in a formula recursively as the follows:*

- *x is a* single-valued variable *in a predicate P_j if x is in the place of a single-valued parameter following Definition 2.10,*

- *x is a single-valued variable in φ OP ψ (where OP is either $\wedge, \vee, \rightarrow, \leftrightarrow$ if x is a single-valued variable in either φ or ψ,*

- *x is a single-valued variable in $\neg\varphi$ if x is a single-valued variable in φ,*

- *x is a single-valued variable in $Q_y\varphi$ (where Q is either $\exists$ or $\forall$) if x is a single-valued variable in φ.*

A variable x is a multi-valued variable *in φ if x is not a single-valued variable in φ and x appears in φ.*

Definition 2.12 (Predicate QBF/BDD encoding). *Given a predicate language $\mathcal{L}_{\mathsf{Pred}}$ that is composed of a finite set* Pred *of finite domain predicates $\{P_1, ..., P_N\}$, the QBF/BDD encoding task is to map each truth condition $\langle M_i, V_i\rangle$ into a truth assignment of the corresponding QBFs. This can be done by mapping each predicate P_j to a set of QBF proposition variables $\mathcal{P}(P_j)$ where $\mathcal{P}(P_j)$ is constructed in two steps as follows:*

- *For every singled-valued parameter $t_{j,k} \in T_S(P_j)$,*
  - *for every $\mathbf{v} \in \Pi_{l=1,...,|T_M(P_j)|}\Omega_{j,l}$ that can be assigned to the set of multi-valued parameters $T_M(P_j)$, we create a frame (of discernment) variable $F_j(\mathbf{v}, t_{j,k})$.*
- *For every frame variable $F_j(\mathbf{v}, t_{j,k})$ ($\mathbf{v} \in \Pi_{l=1,...,|T_M(P_j)|}\Omega_{j,l}$), we create a binary encoding for the set of values that can be assigned to such a variable. This will require $\lceil log_2|\Omega(t_{j,k})|\rceil$ number of propositional variables in $\mathcal{P}(P_j)$, denoted by $\mathcal{P}_j(\mathbf{v}, t_{j,k})$.*

The QBF proposition variables for a finite set $\mathsf{Pred} = \{P_1, ..., P_N\}$ of predicates is the union of the QBF proposition variables of all predicates in the set:

$$\mathcal{P}(\mathsf{Pred}) = \cup_{P_i \in \mathsf{Pred}}\mathcal{P}(P_i).$$

Now for each predicate P_j and each valuation of its multi-valued parameters $T_M(P_j)$, we encode all possible values that can be assigned to a single-valued parameter $t_{j,k}$ (i.e. $\Omega(t_{j,k})$) with the propositional variables $\mathcal{P}_j(v_{j,l}, t_{j,k})$. This can be done following the set encoding scheme introduced in Section 2.3.1. In total, we create a set of propositional variables $\mathcal{P}(\mathcal{L}_{\mathsf{Pred}}) = \cup_{P_j \in \mathsf{Pred}}\mathcal{P}(P_j)$ such that each truth condition of $\mathcal{L}_{\mathsf{Pred}}$ can be encoded by a truth assignment to $\mathcal{P}(\mathcal{L}_{\mathsf{Pred}})$. The total

number of propositional variables for a predicate language $\mathcal{L}_{\mathsf{Pred}}$ is

$$\Sigma_{P_j \in \mathsf{Pred}} \left(\Pi_{t_{j,k} \in T_M(P_j)} |\Omega_{j,k}| \times \Sigma_{t_{j,k} \in T_S(P_j)} \left(\lceil log_2 |\Omega(t_{j,k})| \rceil \right) \right).$$

As the size of domain of some single-valued parameters might not be an exact power of 2, we define the concept of *universe* of a predicate P_j, denoted by $Universe(P_j)$, to regulate the encoding assignments to conform with the set of valid valuations. The concept of universe of a formula corresponds to the frame of discernment defined in Section 2.5.

Definition 2.13 (Predicate universe). *The universe of a predicate P_j is defined as:*

$$Universe(P_j) = \bigwedge_{\mathbf{v} \in \Omega_{T_M(P_j)}} \left[\bigwedge_{t_{j,k} \in T_S(P_j)} \left(\bigvee_{c \in \Omega_{j,k}} (\mathcal{P}(\mathbf{v}, t_{j,k}) = c) \right) \right]$$

where $\mathcal{P}(\mathbf{v}, t_{j,k}) = c$ denotes a QBF encoding of the element c in the set $\Omega_{j,k}$ following the set encoding scheme introduced in Section 2.3.1. The universe of a formula ϕ, denoted by $Universe(\phi)$, is defined recursively as follows:

- *If $\phi \stackrel{def}{=} P_j(t_1, ...)$, then $Universe(\phi) = Universe(P_j)$.*
- *If $\phi \stackrel{def}{=} \neg\varphi$, then $Universe(\phi) = Universe(\varphi)$.*
- *If $\phi \stackrel{def}{=} Q_y\varphi$ (where Q is either $\exists$ or $\forall$), then $Universe(\phi) = Universe(\varphi)$.*
- *If $\phi \stackrel{def}{=} \varphi \ OP \ \psi$ (where OP is either $\wedge, \vee, \rightarrow, \leftrightarrow$), then $Universe(\phi) = Universe(\varphi) \wedge Universe(\psi)$.*

Now we are able to encode all possible truth conditions of a finite predicate language $\mathcal{L}_{\mathsf{Pred}}$ with truth assignments to the corresponding QBFs/BDDs.

Definition 2.14 (QBF/BDD encoding of predicate formula). *The encoding of a formula $\phi \in \mathcal{L}_{\mathsf{Pred}}$ can be done recursively as the following:*

- *If* $\phi \stackrel{def}{=} P_j(c_{j,1}, ..., c_{j,n})$ *with all its parameters being constants, then* ϕ *is encoded by a QBF as the following*

$$QBF(\phi) = \bigwedge_{t_{j,k} \in T_S(P_j)} (\mathcal{P}(\langle c_{j,l_1}, ..., c_{j,l_M} \rangle, t_{j,k}) = c_{j,k})$$

  where $\langle c_{j,l_1}, ..., c_{j,l_M} \rangle$ *is a sub-vector of the constants* $\langle c_{j,1}, ..., c_{j,n} \rangle$ *that are assigned to multi-valued parameters, and* $t_{j,k}$ *is a single-valued parameter which takes the value* $c_{j,k}$ *in the predicate.*

- *If* $\phi \stackrel{def}{=} \varphi$ *OP* ψ *(where OP is either* $\wedge, \vee, \rightarrow, \leftrightarrow$*), and there are no free variables in both* φ *and* ψ*, then*

$$QBF(\phi) = QBF(\varphi) \ OP \ QBF(\psi).$$

- *If* $\phi \stackrel{def}{=} \neg\varphi$*, and there are no free variables in* φ*, then*

$$QBF(\phi) = \neg QBF(\varphi) \wedge Universe(\phi).$$

- *If* $\phi \stackrel{def}{=} Q_x\varphi$ *(where* Q *is either* $\exists$ *or* $\forall$*) and* x *is not free in* φ*, then*

$$QBF(\phi) = QBF(\varphi).$$

- *If* $\phi \stackrel{def}{=} \exists_x\varphi$ *and* x *is a free variable in* φ*, then*

$$QBF(\phi) = \bigvee_{c \in \Omega(\phi,x)} QBF(\varphi[x/c]).$$

- *If* $\phi \stackrel{def}{=} \forall_x\varphi$ *and* x *is a free variable in* φ*, then*

$$QBF(\phi) = \bigwedge_{c \in \Omega(\phi,x)} QBF(\varphi[x/c]).$$

- *If ϕ is a formula with a free single-valued variable x, then*

$$QBF(\phi) = \bigvee_{c \in \Omega(\phi,x)} QBF(\varphi[x/c]).$$

- *If ϕ is a formula with a free multi-valued variable x, then*

$$QBF(\phi) = \bigwedge_{c \in \Omega(\phi,x)} QBF(\varphi[x/c]).$$

In the above, for a predicate formula with free variables, multi-valued free variables are interpreted as universal quantifications over these variables as in the classic first-order logic, but single-valued free variables are interpreted as existential quantification over these variables. In this work, the single-valued free variables are viewed as placeholders for terms yet to be chosen or as parameters in a generalized plan as in [Pednault, 1987] which follows the semantics of free variables in first-order dynamic logic [Harel, 1979].

Example 2.7 (Predicate QBF/BDD encoding). *Two special cases of encoding a formula with free variables are:*

- *A predicate $P_j(c_{j,1}, ..., x, ..., c_{j,n})$ with a free single-valued variable x (corresponding to parameter $t_{j,k}$ of P_j) is encoded by a QBF as the following*

$$QBF\left(P_j(c_{j,1}, ..., x, ..., c_{j,n})\right) = \bigvee_{c_{j,k,l} \in \Omega_{j,k}} QBF\left(P_j(c_{j,1}, ..., c_{j,k,l}, ..., c_{j,n})\right).$$

- *A predicate $P_j(c_{j,1}, ..., x, ..., c_{j,n})$ with a free multi-valued variable x (corresponding to parameter $t_{j,k}$ of P_j) is encoded by a QBF as the following*

$$QBF\left(P_j(c_{j,1}, ..., x, ..., c_{j,n})\right) = \bigwedge_{c_{j,k,l} \in \Omega_{j,k}} QBF\left(P_j(c_{j,1}, ..., c_{j,k,l}, ..., c_{j,n})\right).$$

2.4.4 Optimizations of the encoding

Definition 2.15 (Optimizing QBF/BDD encoding of predicate language). *Based on the encoding defined in Definition 2.14, the encoding of a formula $\phi \in \mathcal{L}_{\mathsf{Pred}}$ can be optimized in various ways:*

- *If $\phi \stackrel{def}{=} P_i(c_{i,1}, ..., x, ..., c_{i,n})$ OP $P_j(c_{j,1}, ..., x, ..., c_{j,n})$ (where OP is either $\wedge, \vee$) with all the parameters other than the variable x being constants and let $t_i(x)$ and $t_j(x)$ be the parameters in predicates P_i and P_j where x appears, then ϕ can be encoded by a QBF as the following*

$$
\begin{aligned}
QBF(\phi) = & \\
& \bigwedge_{t_{i,k} \in T_S(P_i), t_{i,k} \neq t_i(x)} \Big(\mathcal{P}(\langle c_{i,l_1}, ..., c_{i,l_{M_i}} \rangle, t_{i,k}) = c_{i,k} \Big) \\
& OP \bigwedge_{t_{j,k} \in T_S(P_j), t_{j,k} \neq t_j(x)} \Big(\mathcal{P}(\langle c_{j,l_1}, ..., c_{j,l_{M_j}} \rangle, t_{j,k}) = c_{j,k} \Big) \\
& \wedge \Big(\mathcal{P}(\langle c_{i,l_1}, ..., c_{i,l_{M_i}} \rangle, t_i(x)) = \mathcal{P}(\langle c_{j,l_1}, ..., c_{j,l_{M_j}} \rangle, t_j(x)) \Big)
\end{aligned}
$$

Other optimizations of the encoding can be employed depending on the applications. For example, [Edelkamp and Helmert, 1999] proposed an encoding specialized for AI planning.

2.5 An algebra of information

An information algebra adapted from [Langel and Kohlas, 2005] can be used to characterize how QBFs and BDDs are used to represent agent systems. Probability theory, possibility theory and Dempster-Shafer theory have been proposed to reason about uncertainties. As showed in [Kohlas, 2003a,b], all these uncertainty reasoning models can be viewed as instantiations of the same information algebra introduced below.

In the information algebra, a finite collection $\mathcal{P} = \{p_1, \ldots, p_n\}$ of variables is associated with finite sets of possible values $\Omega_1, \ldots, \Omega_n$. Ω_i is called the *frame of discernment* (also referred to

simply as *frame*) for the variable p_i, also denoted by $\Omega(p_i)$. In the case of QBFs/BDDs, $\Omega(p_i) = \{\text{FALSE}, \text{TRUE}\}$ for all proposition variables $p_i \in \mathcal{P}$.

Bold font small letters, such as $\mathbf{s}, \mathbf{t}, \mathbf{u}$ and so on, are used to denote subsets of variables in $\mathcal{P}$. For a nonempty subset $\mathbf{s} \subseteq \mathcal{P}$ of variables, $\Omega(\mathbf{s})$ is the Cartesian product of the frames $\Omega(p_j)$ ($p_j \in \mathcal{P}$).

$$\Omega(\mathbf{s}) = \Pi_{p_j \in \mathbf{s}} \Omega(p_j)$$

is the set of possible value tuples of the group $\mathbf{s}$ of variables. $\Omega(\mathbf{s})$ is called the *domain* of $\mathbf{s}$. When the frame for each variable is fixed, the domain (also called *the frame of discernment*) for a group $\mathbf{s}$ of variables are fully determined by the subset $\mathbf{s}$. An element $\mathbf{x} \in \Omega(\mathbf{s})$ is called a *configuration* of variables $\mathbf{s}$. $\mathbf{x}$ corresponds to a value assignment to the variables in $\mathbf{s}$. Following the notation used in [Kohlas, 2003a], the notations of a variable set $\mathbf{s}$ and its domain configuration $\Omega(\mathbf{s})$ are used interchangeably unless an explicit distinction is needed.

Each domain $\mathbf{s}$ can be taken as a *question*: Which configuration $\mathbf{x} \in \Omega(s)$ for the variables $\mathbf{s}$ is selected? The set $2^{\mathcal{P}}$ of subsets of the variables in $\mathcal{P}$ along with the inclusion relation $\subseteq$ ($\mathcal{P}$) forms a partial order. This partial order captures the concept of *fineness of domains* (and their corresponding questions). If $\mathbf{t} \subset \mathbf{s}$, then the domain (and its corresponding question) of $\mathbf{t}$ is less fine than the domain (and its corresponding question) of $\mathbf{s}$. $\mathcal{P}$ is the finest possible question, and the empty domain $\mathbf{s} = \emptyset$ is the coarsest question.

In the partial order of domains, every pair $\mathbf{s}$ and $\mathbf{t}$ of variable sets have a *supremum* and *infimum*. They are the set union $\mathbf{s} \cup \mathbf{t}$ and the set intersection $\mathbf{s} \cap \mathbf{t}$ respectively. Therefore the partial order on the subsets of variables form a lattice. Correspondingly, two questions $\mathbf{s}$ and $\mathbf{t}$ can be *combined* into the supremum of $\mathbf{s}$ and $\mathbf{t}$ (the lattice's join). Also, the common part of two questions $\mathbf{s}$ and $\mathbf{t}$ can be obtained as the infimum of $\mathbf{s}$ and $\mathbf{t}$ (the lattice's meet).

Syntactically, a piece of information is represented by a QBF ϕ (or its finite domain predicate

language representation). Semantically, we interpret the formula ϕ into its models

$$\mathcal{I}(\phi) \subseteq \Omega(dom(\phi))$$

where the domain of each variables in $dom(\phi)$ is $\{\text{TRUE}, \text{FALSE}\}$. A configuration $\mathbf{x}$ corresponds to a truth assignment: $\mathbf{x} : dom(\phi) \rightarrow \{\text{TRUE}, \text{FALSE}\}^{|dom(\phi)|}$. Putting ϕ into different domains/questions will lead to different models. For example, a formula $p \wedge q$ can be interpreted as $\{\langle p = 1, q = 1\rangle\}$ if it is put in the domain $\{p, q\}$; it can also be interpreted as $\{\langle p = 1, q = 1, r = 0\rangle, \langle p = 1, q = 1, r = 1\rangle\}$ if it is put in the domain $\{p, q, r\}$. As the domain of a formula does matter when we interpret it, we label a QBF formula ϕ relative to its domain $\mathbf{s}$ by:

$$dom(\phi) = \mathbf{s}$$

to denote that the semantic interpretation of ϕ is in the discernment frame of $\Omega(\mathbf{s})$.

2.5.1 Projection

A piece of labeled information ϕ can not only be considered relative to its domain $\mathbf{s} = dom(\phi)$ but also be considered relative to a subset of its domain $\mathbf{t} \subseteq \mathbf{s}$. Formally, an information ϕ relative to domain $\mathbf{s}$ can be *project*ed onto a sub domain $\mathbf{t} \subseteq \mathbf{s}$:

$$\phi^{\downarrow \mathbf{t}} \stackrel{\text{def}}{=} \{\mathbf{x_t} | \text{ there is an } \mathbf{x_{s-t}} \text{ such that } (\mathbf{x_t}, \mathbf{x_{s-t}}) \in \Omega(\phi) \ \}$$

and

$$dom(\phi^{\downarrow \mathbf{t}}) = \mathbf{t}$$

The projection operation can be implemented using a QBF/BDD existential quantification

$$\phi^{\downarrow \mathbf{t}} = \exists_{dom(\phi) \backslash \mathbf{t}} \phi$$

2.5.2 Extension

A piece of labeled information ϕ with $dom(\phi) = \mathbf{s}$ can be *extend*ed to answer a finer question regarding a $\mathbf{t} \supseteq \mathbf{s}$. This can be done without introducing new information to ϕ:

$$\phi^{\uparrow \mathbf{t}} \stackrel{\text{def}}{=} \Omega(\phi) \times \Omega_{\mathbf{t}-\mathbf{s}}$$

whose valuations are the cartesian product of ϕ's valuations and the frames of $\mathbf{t} - \mathbf{s}$ and

$$dom(\phi^{\uparrow \mathbf{t}}) = \mathbf{t}$$

Extension can be implemented in QBF/BDD by

$$\phi^{\uparrow \mathbf{t}} \stackrel{\text{def}}{=} \phi$$

2.5.3 Transport

A piece of information can not only be used to answer questions regarding its domain or a subset of its domain, but also be used to answer arbitrary questions. When a piece of information ϕ is used to answer a question in an arbitrary domain $\mathbf{t}$, it is called the *transport* of ϕ with domain $\mathbf{s} = dom(\phi)$ into domain $\mathbf{t}$:

$$\phi^{\rightarrow \mathbf{t}} = (\phi^{\uparrow \mathbf{s} \cup \mathbf{t}})^{\downarrow \mathbf{t}}$$

with a resulting domain $dom(\phi^{\rightarrow \mathbf{t}}) = \mathbf{t}$. The transport can be implemented in QBF/BDD by

$$\phi^{\rightarrow \mathbf{t}} = \exists_{dom(\phi) \setminus \mathbf{t}} \phi$$

2.5.4 Relative transport

In addition to the projection, extension and transport defined in [Langel and Kohlas, 2005], in this work we also define the *relative transport* operation to transport a piece of information into an

arbitrary domain $\mathbf{t}$ while putting it under a specific piece of background information ψ:

$$\phi^{\rightarrow \mathbf{t}|\psi} = \phi^{\rightarrow \mathbf{t}} \wedge \psi^{\rightarrow \mathbf{t}}$$

If the target domain $\mathbf{t}$ is not specified, we can define

$$\phi^{\rightarrow \psi} = \phi \wedge \psi^{\rightarrow dom(\phi)}$$

2.5.5 Reinterpretation

In addition to the information operators defined in [Langel and Kohlas, 2005], I also use an information *reinterpretation* operator. A piece of information on a frame of discernment can be reinterpreted into another frame of discernment:

$$\phi^{\mathbf{s}/\mathbf{t}} = \phi[\mathbf{s}/\mathbf{t}]$$

where $\mathbf{s}$ and $\mathbf{t}$ are of the same size. A particular usage of information representation is in reasoning the changes of the world, for example in agent planing. With information reinterpretation, we can reinterpret the information on the frame of the current state into the frame of next state when the world is known to not being changed with respect to such piece of information.

2.6 Summary

In this chapter, *quantified boolean formulae* (QBFs), *binary decision diagrams* (BDDs), *information algebras* and the QBF/BDD encoding of a *finite domain predicate language* are covered. QBFs will be used as an front-end language to represent the underlying BDDs. The finite domain predicate language then acts as a higher level front-end language to represent and manipulate the information that can be expressed in QBFs/BDDs. The information algebra is employed as a meta mathematical tool to talk about how the information can be expressed and transformed using the

underlying predicate language and QBFs/BDDs. The key achievement of using BDDs to represent information (e.g. sets and relations) is that the complexity of the operations will depend on the complexity of the BDD representation instead of the size of the information (e.g. the size of sets and relations). Instead, the operations on BDDs are polynomial in the sizes of the BDDs, and so logic of operations on the information (e.g. sets and relations) will be polynomial in the size of their BDD representation rather than exponential in the sizes of the information.

Chapter 3

Agent state space and policy planning

In this chapter, I will introduce a state-space model as a basis for the formalisation for agents and a society of agents as a whole. This model is an adaptation of a model commonly used in *non-deterministic planning* [Cimatti et al., 2003]. *State*s are objects that capture some aspect of a system, and *action*s are objects that lead to transitions between states. States and actions together define a *state-space*. When action effects are non-deterministic [Cimatti et al., 2003] then what one seeks for any state-space is a *policy*: i.e. a state-action mapping that specifies which actions one should take in a given state.

3.1 State transitions

3.1.1 State space

We define a *non-deterministic domain* to be a tuple

$$\mathcal{M} = \langle \mathcal{P}_{SAS}, \mathcal{S}, \mathcal{A}, \mathcal{S}', \mathcal{R} \rangle$$

where:

- $\mathcal{P}_{SAS} = \mathcal{P}_S \cup \mathcal{P}_A \cup \mathcal{P}_{S'}$ is a finite set of proposition variables;

- $\mathcal{S} \subseteq 2^{\mathcal{P}_S}$ is the set of all possible states;
- $\mathcal{S}' \subseteq 2^{\mathcal{P}_{S'}}$ is the set of all possible next states;
- $\mathcal{A} \subseteq 2^{\mathcal{P}_A}$ is the finite set of actions; and
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}'$ is the state-transition relation.

$\mathcal{P}_S$, $\mathcal{P}_A$ and $\mathcal{P}_{S'}$ are mutually disjoint. A truth assignment to $\mathcal{P}_S$ corresponds to an encoding of a state s; a truth assignment to $\mathcal{P}_{S'}$ corresponds to an encoding of a *next state* s'; and a truth assignment to $\mathcal{P}_A$ corresponds to an encoding of an action a. In combination, a truth assignment to $\mathcal{P}_S \cup \mathcal{P}_A \cup \mathcal{P}_{S'}$ corresponds to an encoding of a state transition; a truth assignment to $\mathcal{P}_S \cup \mathcal{P}_A$ corresponds to an encoding of a state-action pair which can be used to encode a decision on a specific state. For notational convenience, we can denote $\mathcal{P}_{SAS} = \mathcal{P}_S \cup \mathcal{P}_A \cup \mathcal{P}_{S'}$ and $\mathcal{P}_{SA} = \mathcal{P}_S \cup \mathcal{P}_A$. As the set of variables $\mathcal{P}_{SAS}$ and the *state transition relation* $\mathcal{R}$ can fully characterize the whole system, for notational convenience we can also denote the state transition system as a pair

$$\mathcal{M} = \langle \mathcal{P}_{SAS}, \mathcal{R} \rangle.$$

The set of states that are mentioned by a state transition relation $\mathcal{R}$ is then

$$StatesOf(\mathcal{R}) = \{s \mid \langle s, a, s' \rangle \in \mathcal{R} \text{ or } \langle s', a, s \rangle \in \mathcal{R}\}.$$

As a QBF formula ξ corresponds to a set of truth assignments, we can encode sets of states, actions, next states, state transitions, and state-action pairs with a QBF formula:

- a QBF ξ with $dom(\xi) = \mathcal{P}_S$ encodes a set of states, denoted by $\mathcal{S}(\xi)$,
- a QBF ξ with $dom(\xi) = \mathcal{P}_{S'}$ encodes a set of next states, denoted by $\mathcal{S}'(\xi)$,
- a QBF ξ with $dom(\xi) = \mathcal{P}_A$ encodes a set of actions, denoted by $\mathcal{A}(\xi)$,

- a QBF ξ with $dom(\xi) = \mathcal{P}_S \cup \mathcal{P}_A \cup \mathcal{P}_{S'}$ encodes a state transition relationship (a set of state transitions), denoted by $\mathcal{R}(\xi)$, and
- a QBF ξ with $dom(\xi) = \mathcal{P}_S \cup \mathcal{P}_A$ encodes a table of state-action pairs, denoted by $\mathcal{SA}(\xi)$.

With QBFs encoding sets of entities, the operations on these sets can naturally be mapped into logical operations on QBFs as showed in Table 2.2 by setting the proper domains for the encoding formulae, e.g. $\mathcal{P}_S$, $\mathcal{P}'_S$, $\mathcal{P}_A$ and their supersets.

Examples for state expressions χ_1 and χ_2 are as follows:

$$\begin{aligned} \mathcal{S}(\chi_1) \cup \mathcal{S}(\chi_2) &= \mathcal{S}(\chi_1 \vee \chi_2) \\ \mathcal{S}(\chi_1) \cap \mathcal{S}(\chi_2) &= \mathcal{S}(\chi_1 \wedge \chi_2) \\ \mathcal{S}(\chi_1) \setminus \mathcal{S}(\chi_2) &= \mathcal{S}(\chi_1 \wedge \neg\chi_2) \end{aligned}$$

where $dom(\chi_1)$ and $dom(\chi_2)$ are $\mathcal{P}_S$ or $\mathcal{P}'_S$.

Examples for action expressions α_1 and α_2 are as follows:

$$\begin{aligned} \mathcal{A}(\alpha_1) \cup \mathcal{A}(\alpha_2) &= \mathcal{A}(\alpha_1 \vee \alpha_2) \\ \mathcal{A}(\alpha_1) \cap \mathcal{A}(\alpha_2) &= \mathcal{A}(\alpha_1 \wedge \alpha_2) \\ \mathcal{A}(\alpha_1) \setminus \mathcal{A}(\alpha_2) &= \mathcal{A}(\alpha_1 \wedge \neg\alpha_2) \end{aligned}$$

where $dom(\alpha_1)$ and $dom(\alpha_2)$ are $\mathcal{P}_A$.

Examples for state transition expressions δ_1 and δ_2 are as follows:

$$\begin{aligned} \mathcal{R}(\delta_1) \cup \mathcal{R}(\delta_2) &= \mathcal{R}(\delta_1 \vee \delta_2) \\ \mathcal{R}(\delta_1) \cap \mathcal{R}(\delta_2) &= \mathcal{R}(\delta_1 \wedge \delta_2) \\ \mathcal{R}(\delta_1) \setminus \mathcal{R}(\delta_2) &= \mathcal{R}(\delta_1 \wedge \neg\delta_2) \end{aligned}$$

where $dom(\gamma_1)$ and $dom(\gamma_2)$ are $\mathcal{P}_{SAS}$.

3.1.2 Encoding planning domains

On top of the QBFs, we can employ various representations of planning: proposition representations, finite domain predicate representations, and state-variable representation [Nau et al., 2004, Chapter 2]. All of them are equivalent in expressivity and can be translated into one another with at most a linear increase in size if we restrict ourselves to finite domain predicates and variables and no function symbols in predicates [Nau et al., 2004, Chapter 2]. Efficient translation from predicate representation to QBF representation has been studied in [Edelkamp and Helmert, 1999] by doing a syntactic analysis on the input of a planning domain.

In the literature, the *planning domain* of an agent is typically given by a library of operators

$$\mathcal{OP} = \{op_k\}.$$

where

- each op_k, with $dom(op_k) = \mathcal{P}_{SAS}$, is a QBF expression of an operator that can be interpreted into a set of state transitions
- $action(op_k) = op_k^{\downarrow \mathcal{P}_A}$ is op_k's action which can be interpreted into a set of actions,
- $precond(op_k) = op_k^{\downarrow \mathcal{P}_S}$ is op_k's preconditions which can be interpreted into a set of current states, and
- $effect(op_k) = op_k^{\downarrow \mathcal{P}_{S'}}$ is op_k's effects which can be interpreted into a set of next states.

$action(op_k)$ is the set of actions that the operator op_k defines; $precond(op_k)$ is the set of states on which $action(op_k)$ can be carried out; $effect(op_k)$ is the set of states which can be resulted into when an action in $action(op_k)$ is executed in a state in $precond(op_k)$.

In the planning literature, it is typical that any two operators $op_i, op_j \in \mathcal{OP}$ are disjoint:

$$op_i \wedge op_j = \emptyset$$

Under this assumption, an operator library $\mathcal{OP}$ can be interpreted into a state transition system by:

$$\mathcal{R}(\mathcal{OP}) \quad = \quad \bigvee_{op_i \in \mathcal{OP}} op_i.$$

A scenario (see Example 3.1), which is adapted from the Blogohar scenario in [Burnett et al., 2008], exemplifies how the above state space model can be represented with a finite domain predicate language (see Section 2.4). The description of the NGO scenario is in Example 3.1. To represent and reason about the NGO scenario, we need to define the predicate signature of the scenario (see Example 3.2), and then specify the action operators (see Example 3.3). Then the underlying reasoning system can automatically convert the predicate specifications into a QBF/BDD encoding (see Example 3.4) resulting in a state transition relation in Example 3.5.

Example 3.1 (NGO scenario). *In a flooded area, an NGO medical team is deployed to relieve the affected towns (locations), the base hospital (Bs), Surina (Sr), Haram (Hr) and Tersa (Te). These towns are connected by routes, denoted by HW, SR1, SR2, NR1, and NR2, as shown in Figure 3.1.*

Example 3.2 (NGO scenario: Predicate signature of the medical team). *(Cont. Example 3.1) The scenario in Example 3.1 can be modeled by state predicates and action predicates. The locations in the scenario are captured by a domain of locations:* $\Omega_{loc} = \{\mathsf{Bs}, \mathsf{Sr}, \mathsf{Hr}, \mathsf{Te}\}$. *The routes in the scenario are captured by a domain of routes:* $\Omega_{route} = \{\mathsf{HW}, \mathsf{SR1}, \mathsf{SR2}, \mathsf{NR1}, \mathsf{NR2}\}$. *The actions in the scenario are captured by a domain of actions of the medical team:* $\Omega_{M,action} = \{\mathsf{move}, \mathsf{cure}\}$.

The state of the agent assisting the medical team is modeled by two predicates:

$$\mathsf{Pred}_{M,S} = \{\mathsf{at}(\mathsf{loc}), \mathsf{health}(\mathsf{loc})\}.$$

loc *is a variable over the domain of locations* Ω_{loc}. $\mathsf{at}(\mathsf{loc})$ *represents that the medical team is in a town* $\mathsf{loc} \in \Omega_{loc}$. $\mathsf{health}(\mathsf{loc})$ *represents that the residents of a town* $\mathsf{loc} \in \Omega_{loc}$ *are in a healthy*

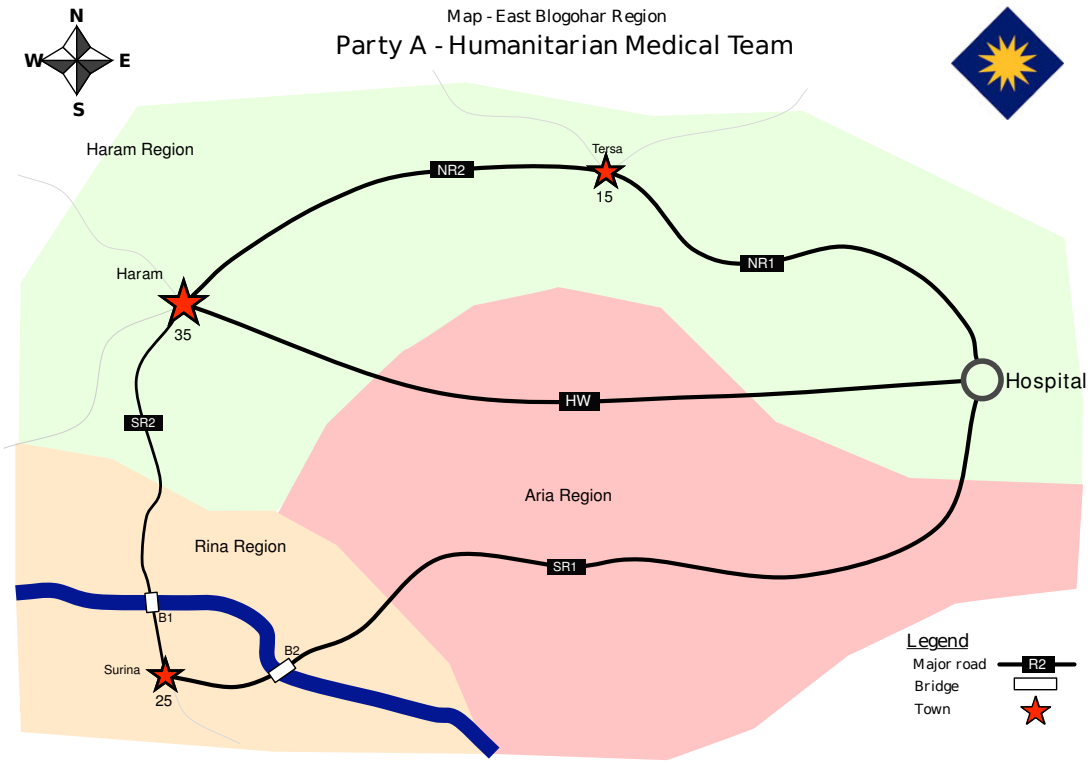

Figure 3.1: An NGO medical team's view of a flooded area

condition. We use the subscripts M, S and M, S' to differentiate between the predicates of current state and next state respectively. For example, $\mathsf{at}_S(\mathsf{M}, \mathsf{loc})$ *and* $\mathsf{health}_\mathsf{S}(\mathsf{M}, \mathsf{loc})$ *are predicates of the current state;* $\mathsf{at}_{\mathsf{M},\mathsf{S}'}(\mathsf{loc})$ *and* $\mathsf{health}_{\mathsf{M},\mathsf{S}'}(\mathsf{loc})$ *are predicates of the next state. Correspondingly, the set of next state predicates is denoted by* $\mathsf{Pred}_{M,S'}$.

The action of an agent is encoded by three predicates:

$$\mathsf{Pred}_{M,A} = \{\mathsf{action}(\mathsf{name}), \mathsf{at}(\mathsf{loc}), \mathsf{to}(\mathsf{loc}), \mathsf{route}(\mathsf{route\_name})\}.$$

name *is a variable over the domain of actions* Ω_{action}. loc *is a variable over* Ω_{loc}. route_name *is a variable over the domain of routes* Ω_{route}. action(name) *represents the identifier of the action;* at(loc), to(loc) *and* route(route_name) *represent the parameters of the action identified by* action(name). at(loc) *is the location where the action is performed; if an action doesn't have a* at *parameter,* at(NIL) *is set.* to(loc) *is the location where action is leading to; if an action doesn't have a* to *parameter,* to(NIL) *is set.* route(route_name) *identifies the route an action is taking; if an action doesn't have a* route *parameter,* route(NIL) *is set. To differentiate from the state predicates, subscript* $_A$ *is used. For example,* $\mathsf{action}_\mathsf{A}(\mathsf{name})$, $\mathsf{at}_\mathsf{A}(\mathsf{loc})$, $\mathsf{to}_\mathsf{A}(\mathsf{loc})$, $\mathsf{route}_\mathsf{A}(\mathsf{route\_name})$.

An action move *can be expressed by a macro* move(loc)*:*

$$\mathsf{move}(\mathsf{route\_name}, \mathsf{orig}, \mathsf{dest}) = \mathsf{action}_\mathsf{A}(\mathsf{move}) \wedge \mathsf{route}_\mathsf{A}(\mathsf{route\_name}) \wedge \mathsf{at}_\mathsf{A}(\mathsf{orig}) \wedge \mathsf{to}_\mathsf{A}(\mathsf{dest})$$

to represent that the medical team carries out the action of moving from the town orig *to the destination* dest *using route* route.

An action cure *can be expressed by a macro* cure(loc)*:*

$$\mathsf{cure}(\mathsf{loc}) = \mathsf{action}(\mathsf{cure}) \wedge \mathsf{at}(\mathsf{loc}) \wedge \mathsf{route}(\mathsf{NIL}) \wedge \mathsf{to}(\mathsf{NIL})$$

to represent that the medical team carries out an action of curing the residents at town loc.

Example 3.3 (NGO scenario: Operators of the medical team). *(Cont. Example 3.2) Using the above constructs, two operators can be defined:*

$$\begin{aligned}
\mathsf{background} &= \mathsf{connect}(\mathsf{nr1}, \mathsf{base}, \mathsf{tersa}) \\
&\quad \vee \mathsf{connect}(\mathsf{hw}, \mathsf{base}, \mathsf{haram}) \\
&\quad \vee \mathsf{connect}(\mathsf{sr1}, \mathsf{base}, \mathsf{surina}) \\
&\quad \vee \mathsf{connect}(\mathsf{nr2}, \mathsf{tersa}, \mathsf{haram}) \\
&\quad \vee \mathsf{connect}(\mathsf{sr2}, \mathsf{haram}, \mathsf{surina}) \\
\mathsf{op}_{\mathsf{M},\mathsf{move}} &= \mathsf{move}(\mathsf{route}, \mathsf{orig}, \mathsf{dest}) \wedge \mathsf{at}_{\mathsf{S}}(\mathsf{orig}) \wedge \mathsf{at}_{\mathsf{S}'}(\mathsf{dest}) \\
&\quad \wedge \mathsf{connect}(\mathsf{route}, \mathsf{orig}, \mathsf{dest}) \wedge \mathsf{background} \\
&\quad \wedge (\mathsf{health}_{\mathsf{S}}(\mathsf{x}) \leftrightarrow \mathsf{health}_{\mathsf{S}'}(\mathsf{x})) \\
\mathsf{op}_{\mathsf{M},\mathsf{cure}} &= \mathsf{cure}(\mathsf{loc}) \wedge \mathsf{at}_{\mathsf{S}}(\mathsf{loc}) \wedge \neg\mathsf{health}_{\mathsf{S}}(\mathsf{loc}) \wedge \mathsf{health}_{\mathsf{S}'}(\mathsf{loc}) \\
&\quad \wedge (\mathsf{at}_{\mathsf{S}}(\mathsf{x}) \leftrightarrow \mathsf{at}_{\mathsf{S}'}(\mathsf{x}))
\end{aligned}$$

An additional predicate $\mathsf{connect}(\mathsf{route\_name}, \mathsf{orig}, \mathsf{dest})$ *is used to represent that fact that a* route *connects two towns* $\langle \mathsf{orig}, \mathsf{dest} \rangle$*. The domain for* route_name *is* Ω_{route}*; the domain for both* orig *and* dest *is* Ω_{loc}*. The* route_name *parameter, the* orig *and* dest *parameter of* connect *are are multiple-valued parameters in the semantics. However, as connections between towns will not be changed by the medical team's actions, the connections are not in the team's state variables, and any of team's actions only gets involved in one route, the location parameters of* connect *can be optimized to be encoded as single-valued parameter*[1]*.*

The formula background *specifies routes and bridges connecting the towns in the area with predicate* $\mathsf{connect}(\mathsf{route}, \mathsf{orig}, \mathsf{dest})$*.* $\mathsf{op}_{\mathsf{M},\mathsf{move}}$ *specifies that the medical can move from a location* orig *to another location* dest *if* $\langle \mathsf{orig}, \mathsf{dest} \rangle$ *are connected by* route*.* $\mathsf{op}_{\mathsf{M},\mathsf{move}}$ *also specifies when the*

[1]This encoding optimization consideration results in a disjunctive ($\vee$) representation of the route connections instead of a conjunctive ($\wedge$) representation in the classic first order logic representation. How to do this optimization automatically for STRIPS style operators has been studied in [Edelkamp and Helmert, 1999].

medical team is moving the health status of all the towns is not changed. $\mathsf{op}_{\mathsf{M,cure}}$ *specifies that, when the residents at location* loc *are not in a healthy state, the medical team can cure the town resulting in the recovery of the residents' health if the medical team is at* loc. $\mathsf{op}_{\mathsf{M,cure}}$ *also specifies that when the medical team is curing a town its location is not changed.*

Example 3.4 (NGO scenario: QBF encoding). *(Cont. Example 3.3) Following the encoding scheme in Definition 2.12, the sets of current, next state and action predicates are encoded by a set of current state QBF variables* $\mathcal{P}_{M,S} = \mathcal{P}(\mathsf{Pred}_{M,S})$*, a set of next state QBF variables* $\mathcal{P}_{M,S'} = \mathcal{P}(\mathsf{Pred}_{M,S'})$*, and set of action QBF variables* $\mathcal{P}_{M,A} = \mathcal{P}(\mathsf{Pred}_{M,A})$ *respectively. In the scenario, the state predicates are*

$$\mathsf{Pred}_{M,S} = \{\mathsf{at}(\mathsf{loc}), \mathsf{health}(\mathsf{loc})\}$$

and loc *is a variable of domain* Ω_{loc} *(*$|\Omega_{loc}| = 4$*).* loc *is a single-valued parameter of predicate* at *(at any time the agent can only be in one location), therefore* at *can be encoded by* $\lceil log_2(|\Omega_{loc}|)\rceil$ *QBF variables contained in* $\mathcal{P}_S$ *so that in any truth assignment to* $\mathcal{P}_S$ *a single location of predicate* at *can be identified. For* health(loc)*,* loc *is a multi-valued parameter (at any time the residents of different locations may have different health status), therefore* health *is encoded by* $|\Omega_{loc}|$ *QBF variables so that in any truth assignment to* $\mathcal{P}_S$ *multiple locations of predicate* health(loc) *can be differentiated. In total, to encode the current and next state predicates, we need*

$$2\left(\lceil log_2(|\Omega_{loc}|)\rceil + |\Omega_{loc}|\right)$$

QBF variables. Similarly, we also have the action predicates

$$\mathsf{Pred}_{M,A} = \{\mathsf{action}(\mathsf{name}), \mathsf{at}(\mathsf{loc}), \mathsf{to}(\mathsf{loc}), \mathsf{route}(\mathsf{route\_name})\}$$

where name *is a single-valued parameter of domain* Ω_{action}*,* loc *is a single-valued parameter of domain* Ω^{-}_{loc}*, and* route_name *is a single-valued parameter of domain* Ω_{route}*. The total number of*

QBF variables needed to encode the set of action predicates is

$$\lceil log_2(|\Omega_{M,action}|)\rceil + 2\lceil log_2(|\Omega^{-}_{loc}|)\rceil + \lceil log_2(|\Omega^{-}_{route}|)\rceil$$

QBF variables.

Example 3.5 (NGO scenario: State transitions of the medical team). *(Cont. Example 3.4) A state transition relation $\mathcal{R}$ can be obtained by incorporating the background knowledge of the routes and the constraints that the medical team can not move and do medical cure at the same time into the operators, and quantifying out the variables not in $\mathcal{P}_{SAS}$ with the projection operator $\downarrow \mathcal{P}_{SAS}$:*

$$\mathcal{R}_M = (op_{M,move} \vee op_{M,cure})^{\downarrow \mathcal{P}_{SAS}}$$

The resulting state transition tables can be found at Table 3.1, and the corresponding state transition graph can be found in Figure 3.1.2.

Current state					Action				Next state				
at	health								at	Health			
	Bs	Hr	Sr	Te	action	route	at	to		Bs	Hr	Sr	Te
Bs	1	1	1	1	move	SR1	Bs	Sr	Sr	1	1	1	1
Bs	1	1	1	0	move	SR1	Bs	Sr	Sr	1	1	1	0
Bs	1	1	0	1	move	SR1	Bs	Sr	Sr	1	1	0	1
Bs	1	1	0	0	move	SR1	Bs	Sr	Sr	1	1	0	0
Bs	1	0	1	1	move	SR1	Bs	Sr	Sr	1	0	1	1
Bs	1	0	1	0	move	SR1	Bs	Sr	Sr	1	0	1	0
Bs	1	0	0	1	move	SR1	Bs	Sr	Sr	1	0	0	1
Bs	1	0	0	0	move	SR1	Bs	Sr	Sr	1	0	0	0
Sr	1	1	0	1	cure	NIL	Sr	NIL	Sr	1	1	1	1
Sr	1	1	0	0	cure	NIL	Sr	NIL	Sr	1	1	1	0
Sr	1	0	0	1	cure	NIL	Sr	NIL	Sr	1	0	1	1
Sr	1	0	0	0	cure	NIL	Sr	NIL	Sr	1	0	1	0
... transitions that are not included in this table...													
Te	0	1	0	0	cure	NIL	Te	NIL	Te	0	1	0	1
Te	0	0	1	0	cure	NIL	Te	NIL	Te	0	0	1	1
Te	0	0	0	0	cure	NIL	Te	NIL	Te	0	0	0	1

Table 3.1: Part of the state transition table for the NGO medical team

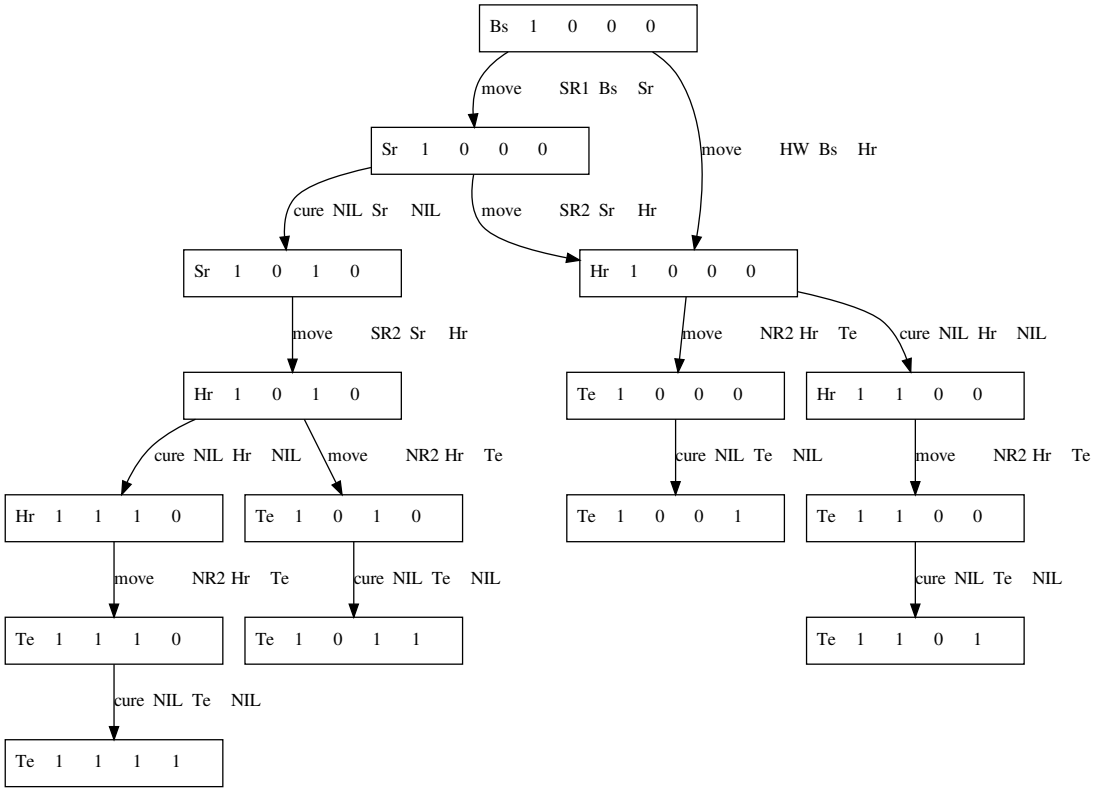

Figure 3.2: Part of a state transition graph for the NGO medical team. The initial states are set to be at the base (Bs) and the health status of all the towns except that of the base are set to FALSE (0).

3.2 Policies

The state-space model described above gives us a way of describing the world in which an agent finds itself, and the actions it can undertake. We can then turn to considering how the agent can plan to act towards its goals. Before we turn to the planning process, let's look at what the output of the planning process will be. We call this output a *policy*, and we consider it to simply be a set of state-action pairs,

$$\pi = \{\langle s_k, a_k \rangle\}.$$

A policy π can also be expressed with a QBF expression with $dom(\pi) = \mathcal{P}_{SA}$.

Example 3.6 (NGO scenario: A policy). *Continuing the scenario shown in Example 3.5, if we set the initial states to be (the following predicates are in the current state predicates* Pred_S*)*

$$at(Bs) \wedge health(Bs) \wedge \neg health(Sr) \wedge \neg health(Hr) \wedge \neg health(Te)$$

we can have the policy shown in Table 3.2 and the corresponding to the state transition graph in Figure 3.2 to achieve the goal states:

$$health(Te).$$

Table 3.2 prescribes which actions to take given all possible encountering of the states an agent can encounter, and Figure 3.2 gives a specific execution path of this policy.

Current state					Action			
at	health							
	Bs	Hr	Sr	Te	action	route	at	to
Bs	1	0	0	0	move	HW	Bs	Hr
Hr	1	0	0	0	move	NR2	Hr	Te
Te	1	0	0	0	cure	NIL	Te	NIL

Table 3.2: A state-action table of the NGO medical team to achieve its goals

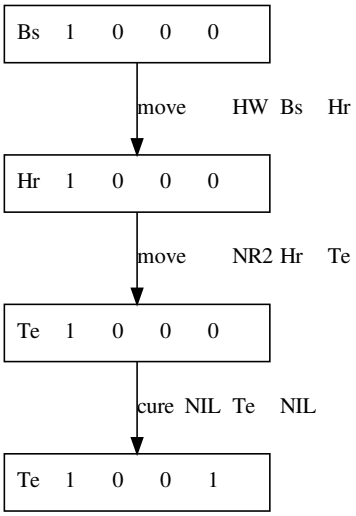

Figure 3.3: A state transition graph of the NGO medical team's execution of a policy towards its goals.

A valid policy candidate with respect to a given state transition system $\mathcal{R}$ will need to satisfy the following conditions

$$\pi \subseteq \mathcal{R}^{\downarrow \mathcal{P}_{SA}}.$$

Given an state s, the set of valid actions can be defined as

$$Applicable(s) = \{a | \exists \langle s, a, s' \rangle \in \mathcal{R}\} = \mathcal{R}^{\downarrow \mathcal{P}_S}$$

that is the set of actions that are applicable in s. The set of states in a policy π is

$$StatesOf(\pi) = \{s \mid \langle s, a \rangle \in \pi\} = \pi^{\downarrow \mathcal{P}_S}.$$

The space of all policies is denoted by

$$\Pi = \{\pi \subseteq \mathcal{R}^{\downarrow \mathcal{P}_{SA}}\}.$$

A policy π is a *deterministic policy*, if for a given state s, there is no more than one action is specified by π, otherwise it is a *non-deterministic policy*. What we are calling a policy is the state-action table used in [Cimatti et al., 2003]. It is also related to what the literature on MDPs calls a policy [Boutilier et al., 1999], but we allow a policy to only specify actions for a subset of all possible states.

A policy can be specified manually by a list of QBFs

$$\pi = \{\pi_k\}$$

where $dom(\pi) = \mathcal{P}_{SA}$. If we assume that for any two segments of policies $\pi_i, \pi_j \in \pi$, we have $\pi_i \wedge \pi_j = \emptyset$. Then we can build the whole policy as just:

$$\pi = \bigvee_k \pi_k.$$

In general, with QBFs/BDDs, we can manipulate segments of a policies with set operations as follows:

$$\begin{aligned} \pi(\alpha_1) \cup \pi(\alpha_2) &= \pi(\alpha_1 \vee \alpha_2) \\ \pi(\alpha_1) \cap \pi(\alpha_2) &= \pi(\alpha_1 \wedge \alpha_2) \\ \pi(\alpha_1) \setminus \pi(\alpha_2) &= \pi(\alpha_1 \wedge \neg\alpha_2) \end{aligned}$$

where $dom(\alpha_i) = \mathcal{P}_{SA}$.

3.3 Execution structure and solution concepts

To underpin the planning algorithms that will be introduced, we need to formally characterize the execution of a policy in an environment modeled by the state transition system. The following definition is adapted from [Cimatti et al., 2003].

Definition 3.1 (Execution Structure). *An* execution structure *induced by the policy π from a set of initial states I is a directed graph $\Sigma_\pi(I) = (V_\pi, E_\pi)$ which can be recursively defined as*

- *if $s \in I$, then $s \in V_\pi$, and*
- *if $s \in V_\pi$ and there exists a state-action pair $\langle s, a\rangle \in \pi$ such that $\langle s, a, s'\rangle \in \mathcal{R}$, then $s' \in V_\pi$ and $a : \langle s, s'\rangle \in E_\pi$ where the action a is the label of the edge.*

Two special cases of an execution structure are $\Sigma_\pi(S)$ where the set of initial states is the whole state space, and $\Sigma_\pi(S(\pi))$ where the set of initial states is the whole set of the policy states. If the policy π chooses nondeterministically between all the available actions in every state, then $\Sigma_\pi(S)$ is the original state transition graph of the whole state transition model.

Definition 3.2 (Execution path). *An* execution path *of a policy π from a set of states I is a possibly infinite sequence $s_0, a_0, s_1, a_1, s_2, a_2, \ldots$ of state-actions ended with a state in the execution structure $\Sigma_\pi(I) = \langle V_\pi, E_\pi\rangle$ such that*

- *$\langle s, a\rangle \in \pi$, and*
- *for all states s_i in the sequence:*
  - *either s_i is the last state of the sequence, in which case s_i is a* terminal state *of $\Sigma_\pi(I)$, or*
  - *$\langle s_i, s_{i+1}\rangle \in E_\pi$.*

A state s' is said to be *reachable* from s in the execution structure Σ_π if there is a path from s to s' in Σ_π. Σ_π is an *acyclic execution* iff all its execution paths are finite.

These ideas then give us a way to classify policies:

Definition 3.3 (Non-deterministic planning solution concepts). *Given a set of initial states I and a set of goal states G for a nondeterministic domain $\mathcal{M} = \langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$, let π be a policy for $\mathcal{M}$ with execution structure $\Sigma_\pi(I)$, then*

- *π is a* weak solution *to achieve G iff for any state $s_0 \in I$ there is some terminal state s' of $\Sigma_\pi(I)$ such that $s' \in G$ and it is reachable from s_0;*
- *π is a* strong solution *to achieve G iff $\Sigma_\pi(I)$ is acyclic and all terminal states of $\Sigma_\pi(I)$ are also in G;*
- *π is a* strong cyclic solution *to achieve G iff from any state s_0 in $\Sigma_\pi(I)$ some terminal state s is reachable and all the terminate states of $\Sigma_\pi(I)$ are in G.*

With a weak solution policy, we have a path from every initial state to the goals in a finite number of steps, but no guarantee that in a non-deterministic world the goal will be achieved; with a strong solution policy, we have a guarantee that the goals can be achieved in a finite number of steps from every initial state despite actions being non-deterministic if the state space is acyclic; and with a strong cyclic solution, we are guaranteed that the goals will be achieved from every initial state even in the face of non-determinism and cycles in the state-space so long as the cycle can be broken non-deterministically (i.e. eventually the cycle will be broken by some actions, but this breaking succeeds stochastically).

3.4 Kripke structure of executions

The execution structure of a policy can formally characterized using *computation tree logic* (*CTL*) which can be interpreted under the possible world semantics of Kripke (the notations in this section are adapted from [Emerson, 1990] and [Jensen, 2003]). In addition to the standard connectives of a propositional logic, CTL formulae have temporal connectives and path quantifiers over the possible

worlds that are characterized by the propositional language. The temporal connectives are X ("next-time") and U ("until"). The path quantifiers are E ("exists") and A ("for all").

Definition 3.4 (Computation Tree Logic (CTL)). *On top of a finite set $\mathcal{P}$ of propositions, CTL formulae are defined recursively as follows:*

- *Every element of $\mathcal{P}$ is a formula.*
- *If ϕ and ψ are formulae, $\neg\psi$, $\psi \vee \phi$, $\psi \wedge \phi$, $\psi \rightarrow \phi$, $EX\psi$, $AX\psi$, $E(\phi U \psi)$ and $A(\phi U \psi)$ are formulae*

The CTL semantics can be captured by *Kripke structures.*

Definition 3.5 (Kripke Structure). *A Kripke structure is a pair $K = \langle W, T \rangle$ where*

- *$W = 2^{\mathcal{P}}$ is a set of all possible worlds (each possible world corresponds to a truth assignment to the propositions in $\mathcal{P}$), and*
- *$T \subseteq W \times W$ is a transition relation.*

A path π in K is a sequence $w_0, w_1, \ldots$ of worlds in W such that $T(w_i, w_{i+1})$ for every $i \geq 0$. CTL semantics is defined recursively on whether a formula is holds in a world w of W. If ϕ and ψ are two formulae in CTL, then

- *$K, w_0 \models p$ iff $w_0 \models p$ in the propositional logic semantics; specially, for any K and w, $K, w \models TRUE$ always holds;*
- *$K, w_0 \models \neg\psi$ iff $K, w_0 \not\models \psi$; specially, for any K and w, $K, w \not\models FALSE$ always doesn't hold;*
- *$K, w_0 \models \psi \vee \phi$ iff $K, w_0 \models \psi$ or $K, w_0 \models \phi$;*
- *$K, w_0 \models EX\phi$ iff there exists a path $w_0 w_1 \ldots$ in T such that $K, w_1 \models \phi$;*
- *$K, w_0 \models AX\phi$ iff, for every path $w_0 w_1 \ldots$ starting with w_0 in T, we have $K, w_1 \models \phi$;*

- $K, w_0 \models E(\phi U \psi)$ *iff there exists a path* $w_0 w_1 \ldots$ *in* T *and an index* $i \geq 0$ *such that* $K, w_i \models \psi$ *and for all* $0 \leq j < i$, $K, w_j \models \phi$;

- $K, w_0 \models A(\phi U \psi)$ *iff, for every path* $w_0 w_1 \ldots$ *starting with* w_0 *in* T, *there exists an index* $i \geq 0$ *such that* $K, w_i \models \psi$ *and for all* $0 \leq j < i$, $K, w_j \models \phi$.

For notational convenience, we can define

- $\text{AF}\psi \stackrel{\text{def}}{=} A(\text{TRUE U } \psi)$ in which F stands for "future" or "eventual" and AF means for all paths eventually ψ will be true.

- $\text{EF}\psi \stackrel{\text{def}}{=} E(\text{TRUE U } \psi)$ in which EF means for some path eventually ψ will be true.

- $\text{AG}\psi \stackrel{\text{def}}{=} \neg\text{EF}\neg\psi$ in which G stands for "globally" or "always", and AG means for all paths globally ψ will be true.

- $\text{EG}\psi \stackrel{\text{def}}{=} \neg\text{AF}\neg\psi$ means there exists some paths ψ will globally be true.

Definition 3.6 (State transition induced Kripke structure). *Let be* $\mathcal{R} \subseteq 2^{\mathcal{P}_S} \times 2^{\mathcal{P}_S}$ *be a state transition relation, the Kripke structure* $K_{\mathcal{R}}$ *induced by* $\mathcal{R}$ *is defined as:*

- $W_{\mathcal{R}} = 2^{\mathcal{P}_S}$ *is the set of states in* $\mathcal{R}$, *and*

- $T_{\mathcal{R}}(s, s')$ *iff* $\langle s, a, s' \rangle \in \mathcal{R}$ *for some action* a.

Definition 3.7 (Policy induced Kripke structure). *Let be* $\pi \subseteq 2^{\mathcal{P}_S} \times 2^{\mathcal{P}_A}$ *be a policy for an state relation* $\mathcal{R}$, *a Kripke structure* $K_{\mathcal{R},\pi} = \langle W_{\mathcal{R}}, T_\pi \rangle$ *induced by* π *is defined as:*

- $W_{\mathcal{R}} = 2^{\mathcal{P}_S}$, *and*

- $T_\pi(s, s')$ *iff* $\langle s, a \rangle \in SA$ *and* $s' \in R(s, a)$, *or* $s = s'$ *(a self-loop is introduced in* T_π *to model that once a state is reached by a policy it will stay in the state).*

The *non-deterministic planning solution concepts* can then be formally defined using CTL:

Definition 3.8 (Non-deterministic planning solution concepts using CTL). *Let* $\mathcal{R} \subseteq 2^{\mathcal{P}_S} \times 2^{\mathcal{P}_A} \times 2^{\mathcal{P}'_S}$ *be a state transition,* $\pi \subseteq 2^{\mathcal{P}_S} \times 2^{\mathcal{P}_A}$ *be a policy on* $\mathcal{R}$*, and* $I \subseteq 2^{\mathcal{P}_S}$ *and* $G \subseteq 2^{\mathcal{P}_S}$ *be the set of initial states and goal states characterized by formulae* $\xi(I)$ *and* $\xi(G)$ *respectively, we can characterize the planning solution concepts that* π *can achieve as follows:*

- π *is a weak solution for an intial state* $s \in I$ *iff* $K_{\mathcal{R},\pi}, s \models EF\xi(G)$.
- π *is a strong solution for an intial state* $s \in I$ *iff* $K_{\mathcal{R},\pi}, s \models AF\xi(G)$.
- π *is a strong cyclid solution for an intial state* $s \in I$ *iff* $K_{\mathcal{R},\pi}, s \models AGEF\xi(G)$.

3.5 Weak and strong planning

The weak and strong planning algorithms [Cimatti et al., 2003] are the basic building blocks for major contributions of this dissertation: the algorithms in Chapter 4 for multiagent interaction, the algorithms in Chapter 5 for handling inconsistent information, and the algorithms in Chapter 6 for decentralized planning for inconsistent goals. Therefore, this section repeats the basic planning algorithm taken from [Cimatti et al., 2003] with small modifications as Algorithm 1.

Algorithm 1 can output a weak or a strong policy that can achieve the goals given a set of initial states. It relies on the function $ComputePreSA$ to iteratively compute pre-images backward from goal states. The function $ComputePreSA$ in Algorithm 1 can be set to be one of the following:

$$ComputeWeakPreSA(\mathcal{R}, FT) = \exists_{\mathcal{P}_{S'}} FT[\mathcal{P}_S/\mathcal{P}_{S'}] \wedge \mathcal{R} \tag{3.1}$$

$$ComputeStrongPreSA(\mathcal{R}, FT) = \forall_{\mathcal{P}_{S'}} (\mathcal{R} \rightarrow FT[\mathcal{P}_S/\mathcal{P}_{S'}]) \wedge \exists_{\mathcal{P}_{S'}} \mathcal{R} \tag{3.2}$$

With $ComputeWeakPreSA$, Algorithm 1 computes a weak policy; with $ComputeStrongPreSA$, Algorithm 1 computes a strong policy.

$ComputeWeakPreSA$ (Equation 3.1) computes the set of state-action pairs SA that can lead

to the frontier states FT:

$$ComputeWeakPreSA(\mathcal{R}, FT) = \{\langle s, a\rangle | \text{ there exists } s' \in FT \text{ such that } \langle s, a, s'\rangle \in \mathcal{R}\}$$

Let $FT' = FT[\mathcal{P}_S/\mathcal{P}_{S'}]$. FT' encodes the set of states in the next state variables of the frontier states FT. $\mathcal{R}$ encodes the set of state transitions. $FT' \wedge \mathcal{R}$ then encodes the set of the state transitions that can lead to the state frontiers FT'. $\exists_{\mathcal{P}_{S'}} FT' \wedge \mathcal{R}$ encodes the set of state-action pairs executing which will result into the frontier states. Recall that the semantics of QBF existential quantification over $\mathcal{P}_{S'}$ is

$$\bigvee_{\langle c_1,...,c_{|\mathcal{P}_{S'}|}\rangle \in 2^{\mathcal{P}_{S'}}} (FT' \wedge \mathcal{R})[\mathcal{P}_{S'}/\langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle].$$

This gives us a set of truth assignments to variables $\mathcal{P}_{SA}$ that encodes the set of state-action pairs that can be executed resulting into the frontier.

$ComputeStrongPreSA$ (Equation 3.2) computes the set of state-action pairs SA that can lead to the frontier states FT and only lead to FT:

$$\begin{aligned} ComputeStrongPreSA(\mathcal{R}, FT) = \{\langle s, a\rangle | \text{ there exists } s' \in FT \text{ such that } \langle s, a, s'\rangle \in \mathcal{R} \\ \text{and there exists no } s'' \notin FT \text{ such that } \langle s, a, s''\rangle \in \mathcal{R}\}. \end{aligned}$$

Let $FT' = FT[\mathcal{P}_S/\mathcal{P}_{S'}]$. The universal quantification semantics over all next state QBF variables on $\mathcal{R} \rightarrow FT'$ is

$$\begin{aligned} &\bigwedge_{\langle c_1,...,c_{|\mathcal{P}_{S'}|}\rangle \in 2^{\mathcal{P}_{S'}}} (\mathcal{R} \rightarrow FT')[\mathcal{P}_{S'}/\langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle] \\ &\quad = \bigwedge_{\langle c_1,...,c_{|\mathcal{P}_{S'}|}\rangle \in 2^{\mathcal{P}_{S'}}} (\mathcal{R}[\mathcal{P}_{S'}/\langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle] \rightarrow FT'[\mathcal{P}_{S'}/\langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle]). \end{aligned}$$

For each truth assignment $s' = \langle c_1, ..., c_{|\mathcal{P}_{S'}|} \rangle \in 2^{\mathcal{P}_{S'}}$, there are two cases:

- $s' \notin FT'$. To make $\mathcal{R}[\mathcal{P}_{S'}/\langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle] \rightarrow FT'[\mathcal{P}_{S'}/\langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle]$ TRUE, the state-action pair $\langle s, a\rangle$ must make $\mathcal{R}$ FALSE, namely $\langle s, a, s'\rangle \notin \mathcal{R}$. Therefore, if $s' \notin FT'$, the expression $(\mathcal{R} \rightarrow FT')[\mathcal{P}_{S'}/\langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle]$ encodes the set of state-action pairs that are in $\neg\mathcal{R}^{\downarrow \mathcal{P}_{SA}}$.

- $s' \in FT'$. To make $\mathcal{R}[\mathcal{P}_{S'}/\langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle] \rightarrow FT'[\mathcal{P}_{S'}/\langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle]$ TRUE, the state-action pair $\langle s, a\rangle$ can either make $\mathcal{R}$ FALSE or make $\mathcal{R}$ TRUE:

  - To make $\mathcal{R}$ FALSE, the state-action pair $\langle s, a\rangle$ must make $\langle s, a, s'\rangle \notin \mathcal{R}$. Therefore, if $s' \in FT'$, one possibility is that the expression $(\mathcal{R} \rightarrow FT')[\mathcal{P}_{S'}/\langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle]$ encodes the set of state-action pairs that are in $\neg\mathcal{R}^{\downarrow \mathcal{P}_{SA}}$.

  - To make $\mathcal{R}$ TRUE, the state-action pair $\langle s, a\rangle$ must make $\langle s, a, s'\rangle \in \mathcal{R}$. Therefore, if $s' \in FT'$, another possibility is that the expression $(\mathcal{R} \rightarrow FT')[\mathcal{P}_{S'}/\langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle]$ encodes the set of state-action pairs that are in $\mathcal{R}^{\mathcal{P}_{SA}}$ such that $\langle s, a, s'\rangle \in \mathcal{R}$ and $s' \in FT'$.

In summary, for each truth assignment $s' = \langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle$ the expression $(\mathcal{R} \rightarrow FT')[\mathcal{P}_{S'}/\langle c_1, ..., c_{|\mathcal{P}_{S'}|}\rangle]$ encodes the disjunction of the set $\neg\mathcal{R}^{\downarrow \mathcal{P}_{SA}}$ of state-action pairs, and the set of state-action pairs $\{\langle s, a\rangle\}$ such that $\langle s, a, s'\rangle \in \mathcal{R}$. When the universal quantifier is applied, the state action pairs in $\neg\mathcal{R}^{\downarrow \mathcal{P}_{SA}}$ are kept since they are in every conjunction term, and the intersection of all the state-action pairs $\{\langle s, a\rangle\}$ such that if $\langle s, a, s'\rangle \in \mathcal{R}$ then $s' \in FT'$. By conjoining with $\mathcal{R}^{\downarrow \mathcal{P}_{SA}}$, the expression $\left[\forall_{\mathcal{P}_{S'}} (\mathcal{R} \rightarrow FT[\mathcal{P}_S/\mathcal{P}_{S'}])\right] \wedge \mathcal{R}^{\downarrow \mathcal{P}_{SA}}$ excludes the state-action pairs that are not in the state transitions $\mathcal{R}$. Therefore $computeStrongPreSA$ computes the set of state-action pairs in $\mathcal{R}^{\downarrow \mathcal{P}_{SA}}$ that can lead to FT' and only lead to FT'.

In Algorithm 1, the state set $Solved$ is initialized with the set of goals and the policy SA is initialized to be the $\emptyset$. Therefore, initially when executing SA, from states $Solved$ the goals can be achieved. The algorithm then loops backward from the solved states with the function $ComputeStrongPreSA$ or $ComputeWeakPresA$ to include more states that can lead to the already

Algorithm 1: $plan(R, I, G, computePreSA)$: Planning

Input : (1) R: The transition relation; (2) I: The set of initial states; (3) G: The set of goals states; (4) $computePreSA$: Either $ComputeWeakPreSA$ or $ComputeStrongPreSA$

1 $Solved \leftarrow G$;
2 $FT \leftarrow G$;
3 $newSA \leftarrow \emptyset$;
4 $SA \leftarrow \emptyset$;
5 **while** $newSA \neq \emptyset$ *OR* $I \not\subseteq Solved$ **do**
6 $\quad newSA \leftarrow computePreSA(R, FT)$;
$\quad$ `/* Remove the state-action pairs whose states have already been solved */`
7 $\quad newSA \leftarrow newSA - Solved$;
8 $\quad SA \leftarrow SA \vee newSA$;
9 $\quad FT \leftarrow \exists_{\mathcal{P}_A} newSA$;
10 $\quad Solved \leftarrow Solved \vee FT$;
11 **end**
12 **if** $I \subseteq Solved$ **then**
13 $\quad$ **return** SA;
14 **else**
15 $\quad$ **return** $\emptyset$;
16 **end**

solved states and add the state-action pairs that can lead to the solved states to the policy SA. Therefore the $While$ loop guarantees that $Solved$ contains the set of states from which when the policy SA is executed, the goal goal states can be achieved with the chosen solution concepts. Proofs of the original algorithms can be found in Cimatti's work [Cimatti et al., 2003] and more formal proofs based on the notion of Kripke structure for these algorithms can be found in Jessen's work [Jensen, 2003].

As the complexity of planning algorithms will be the basics for the analysis of the algorithms developed in this dissertation. I will also repeat the complexity analysis of the non-deterministic planning algorithms from [Cimatti et al., 2003; Jensen, 2003] in this section. It is straightforward to see that both $ComputeWeakPreSA$ and $ComputeStrongPreSA$ can be computed by $O(|\mathcal{P}_S|)$ number of BDD operations:

- $ComputeWeakPreSA(\mathcal{R}, FT)$ can be computed with $2 + |\mathcal{P}_S|$ BDD operations: 1 variable renaming operation over FT, 1 AND ($\wedge$) operation, and $|\mathcal{P}_S|$ existential quantifications; and
- $ComputeStrongPreSA(\mathcal{R}, FT)$ can be computed with $3 + 2|\mathcal{P}_S|$ BDD operations: 1 variable renaming operation over FT, 1 IMPLICATION ($\rightarrow$) operation, 1 AND ($\wedge$) operation, $|\mathcal{P}_S|$ existential quantifications, and $|\mathcal{P}_S|$ universal quantifications.

Proposition 3.1 (Non-deterministic planning complexity). *Let D be the maximum length of paths in a state transition relation R. $plan(R, I, G, computePreSA)$ (Algorithm 1) can be computed with n BDD operations, where $n = O(D \times (|\mathcal{P}_S| + |\mathcal{P}_A|)) < O(2^{|\mathcal{P}_S|} \times (|\mathcal{P}_S| + |\mathcal{P}_A|))$.*

Proof. The loop starting at line 5 will explore the state space. $Solved$ begins with an empty set of states, and number of states encoded by $Solved$ will grow at least one state if the loop control BDD $newSA$ is not empty at each step. Therefore, the loop will be bounded by the maximum length of paths in the state space. In each round of the loop, there are 1 set-minus operation, 2 OR (lor) operation, $|\mathcal{P}_A|$ number of existential operations and one $ComputePreSA$ operation whose complexity is bounded by $O(|\mathcal{P}_S|)$. Therefore, we have the planning algorithms computed

by $O(D \times (|\mathcal{P}_S| + |\mathcal{P}_A|))$ BDD operations. The length is also bounded by number of states $2^{|\mathcal{P}_S|}$. Therefore we have the computation also bounded by $O(2^{|\mathcal{P}_S|} \times (|\mathcal{P}_S| + |\mathcal{P}_A|))$ BDD operations.

Iterative squaring

Proposition 3.1 shows that the complexity of the planning algorithm 1 can grow exponentially in the number of state variables. In the literature, a procedure called iterative squaring [Burch et al., 1990] can reduce the BDD operation complexity exponentially. An adapted version of iterative squaring is repeated as Algorithm 2.

Proposition 3.2 (Iterative squaring for weak planning). *$itsqWeakPlan(R, I, G)$ (Algorithm 2) can be computed in $O(|\mathcal{P}_S| \times (|\mathcal{P}_S| + |\mathcal{P}_A|))$ BDD operations.*

Proof. Let $K = |\mathcal{P}_S|$. The loop starting at line 7 will explore the state space. Each loop extends the maximum length of explored paths by squaring — if the maximum length of paths in the previous round is L, then after the execution of line 12 the length will become $L + L$. Therefore, the total number of loops needed is bounded by $log_2|2^{|\mathcal{P}_S|}| = |\mathcal{P}_S|$. The total number of BDD operations is bounded by $O(|\mathcal{P}_S| \times (K + |\mathcal{P}_A|)) = O(|\mathcal{P}_S| \times (|\mathcal{P}_S| + |\mathcal{P}_A|))$.

Therefore, in theory we have a planning algorithm for weak solutions that can be computed within quadratic number of BDD operations. However, in practice the iterative squaring approach usually requires very large BDDs to hold intermediate results of the transitive closures of the state transition relations [Burch et al., 1990]. Iterative squaring is an advantage only when the solutions have very long paths in the execution structure.

Complexity issues

Now, we can look at the detailed complexity of each BDD operation in the planning algorithms. For the variable renaming operator over the set FT of frontier states, let us assume that the variables in $\mathcal{P}_S$ and $\mathcal{P}'_S$ have the same related ordering (these can be achieved easily by interleaving the

Algorithm 2: $itsqWeakPlan(R, I, G)$: Iterative Squaring for Weak Planning

Input : (1) R: The transition relation; (2) I: The set of initial states; (3) G: The set of goals states;

Copy symbols $\mathcal{P}_S$ into a temporary set $\mathcal{P}_Z$;
/* Initialize the execution structure to the transition relation */
/* $SolvedR$ is a decision-result table every entry of which provides the first action for a given state and a final state it can reach given the state-action decision of the whole table */
$I \leftarrow I - G$;
$SolvedR \leftarrow R$;
$oldSolvedR \leftarrow \emptyset$;
$SolvedS \leftarrow \emptyset$;
$SA \leftarrow \emptyset$;
while $oldSolvedR \neq SolvedR \ AND \ I \not\subseteq SolvedS$ **do**
  $oldSolvedR \leftarrow SolvedR$;
  /* Extract the effective state-action control table that can guarantee to the goals w.r.t the solution concept */
  $SA \leftarrow computeWeakPreSA(SolvedR, G)$;
  $SolveS \leftarrow \exists_{\mathcal{P}_A} SA$;
  /* Refine the execution structure controlled by the current state-action table */
  $SolvedR \leftarrow (SolvedR - SolvedS) \vee (SolvedR \wedge SA)$;
  /* Enlarge the execution structure by squaring the length of paths to the solved states */
  $SolvedR \leftarrow SolvedR \vee \exists_{\mathcal{P}_Z} \left(SolvedR[\mathcal{P}_{S'}/\mathcal{P}_Z] \wedge \left(\exists_{\mathcal{P}_A} SolvedR\right)[\mathcal{P}_S/\mathcal{P}_Z]\right)$;
end
if $I \subseteq SolvedS$ **then**
  return SA;
else
  return $\emptyset$;
end

ordering of corresponding variables in $\mathcal{P}_S$ and $\mathcal{P}'_S$). Having the same related variable ordering means that, for any two variables x_i and x_j $\mathcal{P}_S$, and their corresponding variables x'_i and x'_j in $\mathcal{P}'_S$, it holds that x_i succeeds x_j iff x'_i succeeds x'_j in the variable ordering. If $\mathcal{P}_S$ and $\mathcal{P}'_S$ have the same related variable ordering, then $FT[\mathcal{P}_S/\mathcal{P}_{S'}]$ can be done in $O(|FT|)$ ($|FT|$ is the number of nodes in the BDD representing FT) by traversing the BDD and changing the variable names for each node. The resulting BDD is same size as the original BDD. Recall that, for any binary operation such as AND ($\wedge$) and IMPLICATION ($\rightarrow$) between two formulae ψ and ϕ, the BDD operation is bounded by $O(|\psi| \times |\phi|)$ and the size of the resulting BDD is also bounded by $O(|\psi| \times |\phi|)$ (see Table 2.1). Therefore, the variable renaming and the AND in the computation of $FT[\mathcal{P}_S/\mathcal{P}_{S'}] \wedge \mathcal{R}$ in $ComputeWeakPreSA$ can result in a BDD in which the number of nodes is at most $O(|\mathcal{R}| \times |FT|)$. The variable renaming and the IMPLICATION operations in the computation of $\mathcal{R} \rightarrow FT[\mathcal{P}_S/\mathcal{P}_{S'}$ in $ComputeStrongPreSA$ can result in in a BDD with the number of nodes is at most $O(|\mathcal{R}| \times |FT|)$. Recall that, for any single variable quantification (either universal or existential) over a variable x on a formula ϕ, the computation is bounded by $O(|\phi|^2)$ and the resulting BDD is of size $O(|\phi|^2)$ (see Table 2.1). For sequential quantifications over a sequence of variables $x_1, ..., x_n$ in a formula ϕ, the accumulated complexity is $O(|\phi|^n)$. Therefore, $\exists_{\mathcal{P}_{S'}} FT[\mathcal{P}_S/\mathcal{P}_{S'}] \wedge \mathcal{R}$ in $ComputeWeakPreSA$ will be $O((|\mathcal{R}| \times |FT|)^{|\mathcal{P}_S|})$ and will result in a BDD of size $O((|\mathcal{R}| \times |FT|)^{|\mathcal{P}_S|})$ in the worst case. $\forall_{\mathcal{P}_{S'}} (\mathcal{R} \rightarrow FT[\mathcal{P}_S/\mathcal{P}_{S'}]) \wedge \exists_{\mathcal{P}_{S'}} \mathcal{R}$ will be $O((|\mathcal{R}| \times |FT|)^{|\mathcal{P}_S|} \times |\mathcal{R}|^{|\mathcal{P}_S|})$ and will result in a BDD of size $O((|\mathcal{R}| \times |FT|)^{|\mathcal{P}_S|} \times |\mathcal{R}|^{|\mathcal{P}_S|})$ in the worst case.

That is, although both $ComputeWeakPreSA$ and $ComputeStrongPreSA$ can be computed in a linear number of BDD operations, in the worst case the complexity can grow exponentially in the sizes of the BDDs that represent the state transitions and the frontier states. Despite this, in practice [Bryant, 1992; Coudert and Madre, 1995; Hojati et al., 1996], the quantification operations are usually best performed as soon as possible in BDDs. Since the QBF existential and universal quantification replace the variables to be quantified with the constants TRUE and FALSE, early quantification essentially reduces the number of columns in the truth table for the formula. In the BDDs, this amounts to removing all the layers of nodes representing the variables that have been

quantified out. This usually reduces the size of the BDDs significantly.

We have proved that weak planning may, in the worst case, be computed in a quadratic number of BDD operations using iterative squaring. In the worst case, such a sequence of BDD operations might result in exponential size of BDDs. This should not be a surprise as the QBF SAT problem, which is equivalent to a sequence of BDD operations, is PSPACE hard [Ladner, 1977]. The bottom line is that the complexity of planning is in general (both state space planning and plan space planning) PSPACE hard [Bylander, 1992], and so after it is reduced to a sequence of BDD operations, the problem is still PSPACE hard. Using QBFs/BDDs techniques won't change the fundamental complexity of properties of the planning problems.

However, as shown in Proposition 2.1 of Chapter 2, the number of BDD operations is a primary indicator of the complexity of the algorithms based on BDD operations. With BDD-based algorithms for planning, if we know that the amount information encountered in the problem description and during the problem solving processes can be encoded into BDDs of reasonable size, we are sure it can be solved with algorithms that are bounded by tractable number of BDD operations. As the sizes of intermediate BDDs are concerned, many approaches and heuristics have been introduced and used in practice to reduce the size of BDDs holding intermediate results in BDD-based algorithms (please see Section 2.2 of Chapter 2 for more references).

3.6 Strong cyclic planning

Adapting from [Nau et al., 2004] and [Jensen, 2003], we have a strong cyclic planning Algorithm (Algorithm 3). This is a variant of the general non-deterministic planning Algorithm 1. In Algorithm 1, we only consider the frontier states in the computation of the pre-image for the weak and strong solutions, but in Algorithm 3 we need to consider all previously solved states to compute the pre-image for the strong-cyclic solution (see $ComputeStrongCyclicPreSA$ defined in Algorithm 4).

$$PruneOneStepOutgoing(R, SA, preSolvedS)$$
$$= SA - ComputeWeakPreSA\,(R, \neg\,(StatesOf(SA) \vee preSolvedS))$$

Proofs of the correction of the original algorithms can be found in Cimatti's work [Cimatti et al., 2003] and more formal proofs based on the notion of Kripke structure for these algorithms can be found in Jessen's work [Jensen, 2003]. The strong cyclic planning algorithm (Algorithm 3) searches backwards from the goals, generates a state-action table π incrementally to guarantee that all execution paths will eventually lead to states that have been solved w.r.t the strong cyclic solution concept. The strong cyclic solution states are either goals or the states from which the goals can be reached along every execution path (in other word, along every execution path there is a state s such that $K_{\mathcal{R},\pi}, s \models \text{AGEF}\xi(G)$). The algorithm computes the strong-cyclic pre-image by first computing state-action pairs leading to the solved states in a weak solution manner, and then recursively prunes from the policy the states that can go out of the control of the policy using $PruneOneStepOutgoing(R, SA, preSolvedS)$, and prunes the states that are not be connected to the goals at all (a state s such that $K_{\mathcal{R},\pi}, s \models \neg\text{EF}\xi(G)$) using $PruneUnconnected(R, SA, preSolvedS)$.

Proposition 3.3 (Complexity of strong cyclic planning). *Let D be the maximum length of paths in a state transition relation R. $strongCyclicPplan(R, I, G)$ (Algorithm 3) can be computed in $O(D^3 \times (|\mathcal{P}_S| + |\mathcal{P}_A|)) < O(2^{3|\mathcal{P}_S|} \times (|\mathcal{P}_S| + |\mathcal{P}_A|))$ BDD operations.*

Proof. $PruneOneStepOutgoing$ can be computed in $P = O(|\mathcal{P}_S|+|\mathcal{P}_A|)$ operations. $PruneUnconnected$ can be computed in $U = O(D \times (|\mathcal{P}_S| + |\mathcal{P}_A|))$ number of operations since the number of loops within $PruneUnconnected$ is bounded by the maximum length D and each step can be computed in $O(|\mathcal{P}_S| + |\mathcal{P}_A|)$ BDD operations. $Aux = SCPlanAux$ can be computed within $O(D \times (P + U)) = O(D \times (|\mathcal{P}_S| + |\mathcal{P}_A| + D \times (|\mathcal{P}_S| + |\mathcal{P}_A|)))$ since the number of loops within $SCPlanAux$ is bounded by the maximum length D and at each step it calls both $PruneOneStepOutgoing$ and $PruneUnconnected$ once. As the loop of $strongCyclicPplan$ is also bounded by the maxi-

mum execution path length D, *strongCyclicPlan* can be computed in $O(D \times (Aux + |\mathcal{P}_S|)) = O(D^3 \times (|\mathcal{P}_S| + |\mathcal{P}_A|))$ operations.

Algorithm 3: $strongCyclicPlan(R, I, G)$: Strong Cyclic Planning

Input : (1) R: The transition relation; (2) I: The set of initial states; (3) G: The set of goals states;

1 $Solved \leftarrow G$;
2 $FT \leftarrow G$;
3 $newSA \leftarrow \emptyset$;
4 $SA \leftarrow \emptyset$;
5 **while** $newSA \neq \emptyset$ *OR* $I \not\subseteq Solved$ **do**
6 $\quad newSA \leftarrow computeStrongCyclicPreSA(R, Solved)$;
$\quad$ `/* Remove the state-action pairs whose states have already been solved */`
7 $\quad newSA \leftarrow newSA - Solved$;
8 $\quad SA \leftarrow SA \vee newSA$;
9 $\quad FT \leftarrow \exists_{\mathcal{P}_A} newSA$;
10 $\quad Solved \leftarrow Solved \vee FT$;
11 **end**
12 **if** $I \subseteq Solved$ **then**
13 $\quad$ **return** SA;
14 **else**
15 $\quad$ **return** $\emptyset$;
16 **end**

3.7 Summary

This chapter re-iterated a formal model of state transitions for agent behaviors taken from [Cimatti et al., 2003]. The model is expressed using the language of QBFs. The concepts of states, actions, and next states are modeled using three disjoint sets of QBF variables (correspondingly three sets of finite domain predicates that are encoded using these QBF variables). In this way, the state transition model can be implicitly represented, manipulated and searched through the logical operations of QBFs (and the logical operations in the corresponding predicate formulae). A policy on which actions to perform in different states can be prescribed to control the behaviors of the agents so as to the outcome of executing these behaviors. On top of the state transition model

Algorithm 4: ComputeStrongCyclicPreSA(R, $preSolvedS$)

Input : (1) R: State transitions; (2) $preSolvedS$: previously solved states;

$wSA \leftarrow \emptyset$;
repeat
  $oldwSA \leftarrow wSA$;
  $wSA \leftarrow computeWeakPreSA(R, StatesOf(wSA) \vee preSolvedS)$;
  $wSA \leftarrow wSA - preSolvedS$;
  $scSA \leftarrow SCPlanAux(R, startSA, preSolvedS)$;
until $scSA \neq \emptyset$ OR $wSA = oldwSA$;
return $scSA$;

Algorithm 5: $SCPlanAux(R, StartSA, preSolvedS)$

Input : (1) R: State transitions; $StartSA$: Initial state-action table; (3) $preSolvedS$: the frontier of states;

$SA \leftarrow StartSA$;
repeat
  $oldSA \leftarrow SA$;
  $SA \leftarrow PruneOneStepOutgoing(R, SA, preSolvedS)$;
  $SA \leftarrow PruneUnconnected(R, SA, preSolvedS)$;
until $SA = oldSA$;
return SA;

and the controlled policy, the concept of execution structure and its formal Kripke structure can be defined to formally characterize the behaviors of an agent. This leads to three kinds of solution concepts of planning the agent behaviors: weak, strong and strong cyclic solutions towards achieving the designated goals of the agents.

Algorithm 6: $PruneUnconnected(R,\ SA,\ preSolvedS)$

```
1 newSA ← ∅;
2 repeat
3 |   oldSA ← newSA;
4 |   newSA ← SA ∩ ComputeWeakPreSA(R, preSolvedS ∪ StatesOf(newSA));
5 until OldSA = newSA;
6 return newSA;
```

Chapter 4

Multiagent state space and interaction planning

This chapter will present the first new contributions of this dissertation: a model of multiagent state space, and interaction planning based on symbolic model checking techniques. In this chapter, the state transition model of an agent introduced in Chapter 3 will be extended to become a state transition model of a system of agents. The concepts of joint states and actions are modeled as truth assignments to the QBF variables which are composed of those of individual agents representing individual agents' states and actions respectively. The inter-related information of different agents' state space and interactions of actions among the agents can then be captured by the logical connectives over the QBFs encoding the multiagent state space. Correspondingly, the concepts of single agent policy and planning solutions are naturally extended into multiagent setting. In addition, the analysis of agent coordination in multiagent state space can also be performed. This multiagent state space will also provide the basics for a defeasible multiagent action theory and the corresponding planing and coordination algorithms in Chapter 5 and in Chapter 6 respectively.

4.1 A multiagent model

A *multiagent system* AGS is composed of N agents

$$\mathsf{AGS} = \{\mathsf{Ag}_1, \ldots, \mathsf{Ag}_N\}.$$

Different agents typically have different views about the external world and different capability in maneuvering their actions in the external world. The discrepancies among the agents are reflected as the following:

- Different agents have different sets of state and action variables.
- Different agents have different specifications of the state transitions.

On top of dealing with these discrepancies, the multiagent system as whole should be able to demonstrate coherent behaviors towards their designated goals.

In this work, individual agents and the agent system AGS as a whole will both be modeled as the state transition systems introduced in Section 3.1. Each agent Ag_i has its own sets of state, action and next state variables, denoted by

$$\mathcal{P}_{i,SAS} = \langle \mathcal{P}_{i,S}, \mathcal{P}_{i,A}, \mathcal{P}_{i,S'} \rangle.$$

The whole multiagent system is modeled as a *joint state transition* system

$$\mathcal{M} = \langle \mathcal{P}_{SAS}, \mathcal{S}, \mathcal{A}, \mathcal{S}', \mathcal{R} \rangle$$

where

- $\mathcal{P}_{SAS} = \cup_{\mathsf{Ag}_i \in \mathsf{AGS}} \mathcal{P}_{i,SAS}$ where $\mathcal{P}_{i,SAS} = \mathcal{P}_{i,S} \cup \mathcal{P}_{i,A} \cup \mathcal{P}_{i,S'}$ is the set of propositions used by agent Ag_i to represent the information about the world (see Section 3.1.1),
  - $\mathcal{P}_S = \cup_{\mathsf{Ag}_i \in \mathsf{AGS}} \mathcal{P}_{i,S}$,

  - $\mathcal{P}_A = \cup_{\mathsf{Ag}_i \in \mathsf{AGS}} \mathcal{P}_{i,A}$, and
  - $\mathcal{P}_{S'} = \cup_{\mathsf{Ag}_i \in \mathsf{AGS}} \mathcal{P}_{i,S'}$,

- $\mathcal{S} = 2^{\mathcal{P}_S}$ is the set of all possible joint states of the multiagent system,
- $\mathcal{S}' = 2^{\mathcal{P}_{S'}}$ is the set of all possible joint states of the multiagent system,
- $\mathcal{A} = 2^{\mathcal{P}_A}$ is the set of all possible joint action of the multiagent system, and
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}'$ is the set of state transitions of the multiagent system.

For notational convenience, Ag_i and AGS can be used in the places of $\mathcal{P}_{i,SAS}$ and $\mathcal{P}_{SAS}$ respectively in the QBF expressions. For example, the expression $\phi^{\downarrow \mathsf{Ag}_i}$ has the same meaning as the expression $\phi^{\downarrow \mathcal{P}_{i,SAS}}$; the expression $\phi^{\rightarrow \mathsf{AGS}}$ has the same meaning as the expression $\phi^{\rightarrow \mathcal{P}_{SAS}}$ (see Section 2.5 for the definitions of the information transport operator $\rightarrow$ and the information projection operator $\downarrow$).

After laying out the state space for individual agents and the multiagent system as a whole, we are ready to investigate interactions among the agents, and the interactions between individual agents and the multiagent system as a whole.

4.2 Agents' partial views

In a multiagent system $\mathsf{AGS} = \{\mathsf{Ag}_1, \ldots, \mathsf{Ag}_N\}$, the *joint model*

$$\mathcal{M} = \langle \mathcal{P}_{SAS}, \mathcal{R} \rangle$$

is constructed from individual agents' partial models of the system. Each agent Ag_i has its own model of the joint system

$$\mathcal{M}_i = \langle \mathcal{P}_{SAS}, \mathcal{R}_i \rangle$$

with $dom(\mathcal{R}_i) = \mathcal{P}_{SAS}$. The domain of $\mathcal{R}_i$ is $\mathcal{P}_{SAS}$ instead the agent's own variables $\mathcal{P}_{i,SAS}$ means that, when modeling the external world, each agent has its own limited access of the other agents' variables. The joint model $\mathcal{M}_i$ is called Ag_i's partial view of the joint system $\mathcal{M}$.

In this chapter, we will assume that the agents' partial models are consistent with each other, so that we can combine the individual agents' models of the world to form a consistent joint model $\mathcal{M} = \langle \mathcal{P}_{SAS}, \mathcal{R} \rangle$ by

$$\mathcal{R} = \bigwedge_{\mathsf{Ag}_i \in \mathsf{AGS}} \mathcal{R}_i.$$

It is straightforward to see that under the consistent assumption we have

$$\mathcal{R}_i \subseteq \mathcal{R}.$$

This captures the fact that an agent's local view might contain more uncertainty about the external world than the joint system. It also captures the fact that an agent's local view might contain more uncertainty about the actions of the other agents than the joint system.

Example 4.1 (NGO scenario: Facility team's view). *(Cont. of Example 3.5) In addition to the medical team, a facility team is also deployed to the area to repair the damaged roads and bridges. The facility team are aware of which roads and bridges are damaged as shown in Figure 4.1.*

In the facility team's view, the scenario is modeled by two state predicates:

$$\mathsf{Pred}_{F,S} = \{\mathsf{at}(\mathsf{loc}), \mathsf{route\_status}(\mathsf{route\_name}, \mathsf{status})\}.$$

$\mathsf{at}(\mathsf{loc})$ *represents that the facility team is in town* loc *with domain* Ω_{loc}. loc *is a single-valued parameter of* at. $\mathsf{route\_status}(\mathsf{route\_name}, \mathsf{status})$ *represents the condition of the route* $\mathsf{route\_name}$. $\mathsf{route\_name}$ *is a multi-valued parameter. The domain of* $\mathsf{route\_name}$ *is* Ω_{rotue}. status *is a single-valued parameter. The domain of* status *is* $\Omega_{status} = \{\mathsf{flooded}, \mathsf{good}\}$. *Correspondingly, the next state predicate set of the facility team agent is* $\mathsf{Pred}_{F,S'}$.

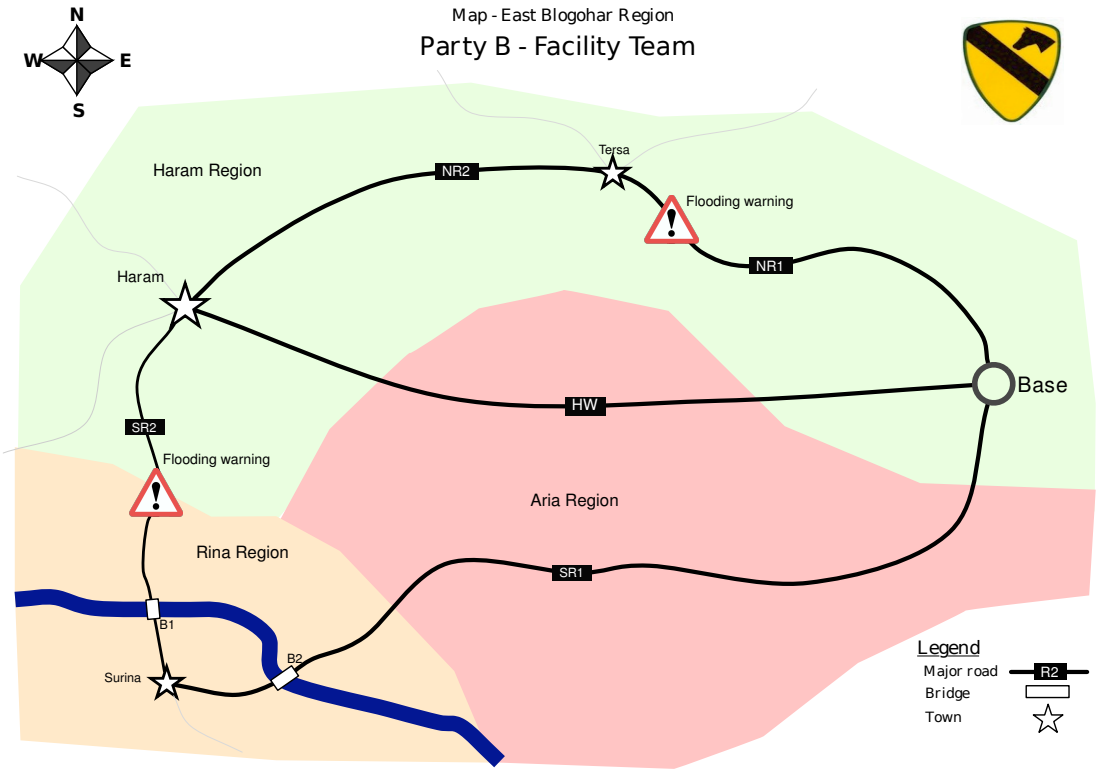

Figure 4.1: An NGO facility team's view of a flooded area

The action predicate set $\mathsf{Pred}_{F,A}$ *is composed of three predicates:*

$$\mathsf{Pred}_{F,A} = \{\mathsf{action(name)}, \mathsf{at(loc)}, \mathsf{to(loc)}\}$$

route(route_name). action(name) *represents the identifier of the action;* at(loc), to(loc) *and* route(route_name) *represent the parameters of the action identified by* action(name). *The domain of* name *is*

$$\Omega_{F,action} = \{\mathsf{move}, \mathsf{repair}\}.$$

at(loc) *is the location where the action is performed; if an action doesn't have a* at *parameter,* at(NIL) *is set.* loc *of* at *is a single-valued parameter of domain* Ω^{-}_{loc}. to(loc) *is the location to which the action leads; if an action doesn't have a* to *parameter,* to(NIL) *is set.* loc *of* to *is a single-valued parameter of domain* Ω^{-}_{loc}. route(route_name) *identifies the route an action is taking; if an action doesn't have a* route *parameter,* route(NIL) *is set.* route_name *is a single-valued parameter of domain* Ω^{-}_{route}.

Following the method used for the medical team in Example 3.2, a move *formula macro is defined as follows*

$$\mathsf{move(route\_name, orig, dest)} = \mathsf{action(move)} \wedge \mathsf{route(route\_name)} \wedge \mathsf{at(orig)} \wedge \mathsf{to(dest)}$$

to represent that the facility team carries an action of moving from the town orig *to the destination* dest *using route* route. *A* repair *formula macro is defined follows*

$$\mathsf{repair(route\_name, orig, dest)} = \mathsf{action(repair)} \wedge \mathsf{at(orig)} \wedge \mathsf{route(route\_name)} \wedge \mathsf{to(dest)}$$

to represent the fact that the facility team carries out the action of repairing the damaged roads in route route_name.

The set of state predicates is being encoded with current state QBF variables $\mathcal{P}_{F,S}$, *denoted by*

$\mathsf{at}_{F,S}$ *and* $\mathsf{condition}_{F,S}$*; the same set of state predicates is also being encoded with next state QBF variables* $\mathcal{P}_{F,S'}$*, denoted by* $\mathsf{at}_{F,S'}$ *and* $\mathsf{condition}_{F,S'}$*. The set of action predicates are encoded into the action QBF variables* $\mathcal{P}_{F,A}$*, denoted by* $\mathsf{action}_{F,A}$*,* $\mathsf{route}_{F,A}$*,* $\mathsf{at}_{F,A}$ *and* $\mathsf{to}_{F,A}$*.*

Using the above constructs, two operators can be defined:

$$\begin{aligned}
\mathsf{op}_{F,move} = {} & \mathsf{move}_{\mathsf{F,A}}(\mathsf{route},\mathsf{orig},\mathsf{dest}) \wedge \mathsf{at}_{\mathsf{F,S}}(\mathsf{orig}) \wedge \mathsf{at}_{\mathsf{F,S'}}(\mathsf{dest}) \\
& \wedge \mathsf{connect}(\mathsf{route},\mathsf{orig},\mathsf{dest}) \wedge \mathsf{background} \\
& \wedge \big(\mathsf{condition}_{\mathsf{F,S}}(\mathsf{x},\mathsf{y}) \leftrightarrow \mathsf{condition}_{\mathsf{F,S'}}(\mathsf{x},\mathsf{y})\big) \\
\mathsf{op}_{F,repair} = {} & \mathsf{repair}_{F,A}(\mathsf{route_name},\mathsf{orig},\mathsf{dest}) \wedge \mathsf{at}_{F,S}(\mathsf{orig}) \wedge \mathsf{at}_{F,S'}(\mathsf{dest}) \\
& \wedge \mathsf{connect}(\mathsf{route_name},\mathsf{orig},\mathsf{dest}) \wedge \mathsf{background} \\
& \wedge \mathsf{condition}_{F,S}(\mathsf{route_name},\mathsf{damaged}) \wedge \mathsf{condition}_{F,S'}(\mathsf{route_name},\mathsf{good})
\end{aligned}$$

The formula background *(see the definition in Example 3.3) specifies routes and bridges connecting the towns in the area with predicate* $\mathsf{connect}(\mathsf{route},\mathsf{orig},\mathsf{dest})$*.* $\mathsf{op}_{F,move}$ *specifies that the facility team can move from a location* orig *to another location* dest *if* $\langle \mathsf{orig},\mathsf{dest}\rangle$ *are connected by route* route_name*.* $\mathsf{op}_{F,move}$ *also specifies when the facility team is moving the health status of all the routes is not changed.* $\mathsf{op}_{F,repair}$ *is similar to* $\mathsf{op}_{F,move}$ *but instead of just moving from* org *to* dest*, the facility also repairs the damaged road.* $\mathcal{R}_F$ *obtains the state transition relation of the facility team by incorporating the background knowledge of the routes and the constraints that the medical team can not move and do repairing at the same time into the operators, and quantifying out the variables not in* $\mathcal{P}_{F,SAS}$ *with the projection operator* $\downarrow \mathcal{P}_{F,SAS}$*:*

$$\mathcal{R}_F = \big(\mathsf{op}_{F,move} \vee \mathsf{op}_{F,repair}\big)^{\downarrow \mathcal{P}_{F,SAS}}$$

The resulting state transition tables of the facility team can be found in Table 4.1, and the corresponding state transition graph can be found in Figure 4.2.

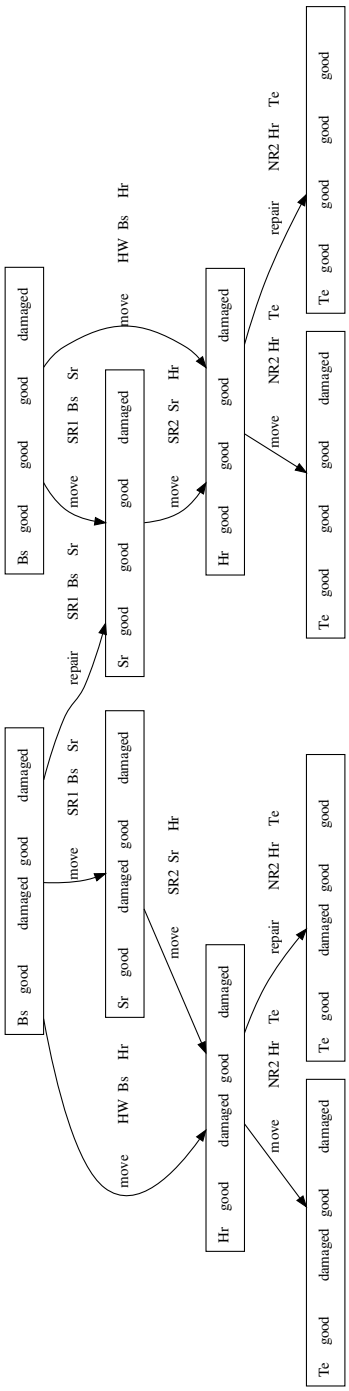

Figure 4.2: Part of the state transition graph for the NGO facility team. The initial states are set to be at the base (Bs) and the condition of routes HW and $SR2$ are set to "good" and $NR2$ are set to "damaged".

Current state				Action				Next state			
		condition							condtion		
at	HW	SR1	SR2	NR2	action	route	at	to	at	HW	SR1	SR2	NR2
Bs	damaged	damaged	damaged	damaged	move	SR1	Bs	Sr	Sr	damaged	damaged	damaged	damaged
Bs	damaged	damaged	damaged	damaged	repair	SR1	Bs	Sr	Sr	damaged	good	damaged	damaged
Bs	damaged	damaged	damaged	good	move	SR1	Bs	Sr	Sr	damaged	damaged	damaged	good
Bs	damaged	damaged	damaged	good	repair	SR1	Bs	Sr	Sr	damaged	good	damaged	good
Bs	damaged	damaged	good	damaged	move	SR1	Bs	Sr	Sr	damaged	damaged	good	damaged
Bs	damaged	damaged	good	damaged	repair	SR1	Bs	Sr	Sr	damaged	good	good	damaged
Bs	damaged	damaged	good	good	move	SR1	Bs	Sr	Sr	damaged	damaged	good	good
Bs	damaged	damaged	good	good	repair	SR1	Bs	Sr	Sr	damaged	good	good	good
Bs	damaged	good	damaged	damaged	move	SR1	Bs	Sr	Sr	damaged	good	damaged	damaged
Bs	damaged	good	damaged	good	move	SR1	Bs	Sr	Sr	damaged	good	damaged	good
...	...	...	...	...	...	...	...	...	...	...	...	...	...
Hr	good	damaged	good	damaged	repair	NR2	Hr	Te	Te	good	damaged	good	good
Hr	good	damaged	good	good	move	NR2	Hr	Te	Te	good	damaged	good	good
Hr	good	good	damaged	damaged	move	NR2	Hr	Te	Te	good	good	damaged	damaged
Hr	good	good	damaged	damaged	repair	NR2	Hr	Te	Te	good	good	damaged	good
Hr	good	good	damaged	good	move	NR2	Hr	Te	Te	good	good	damaged	good
Hr	good	good	good	damaged	move	NR2	Hr	Te	Te	good	good	good	damaged
Hr	good	good	good	good	repair	NR2	Hr	Te	Te	good	good	good	good
Hr	good	good	good	good	move	NR2	Hr	Te	Te	good	good	good	good

Table 4.1: Part of the state transition table for the NGO facility team.

Example 4.2 (NGO scenario: Joint model). *In combination, we can have a joint model of the medical and facility teams. To differentiate between the agent for the medical team and the agent for the facility team, the medical team representation in Example 3.3 is denoted with a subscript M, such as $\mathcal{P}_{M,SAS}$, $\mathsf{at}_{\mathsf{M,S}}$, $\mathsf{health}_{\mathsf{M,S}}$, $\mathsf{health}_{\mathsf{M,S'}}$, $\mathsf{route}_{\mathsf{M,A}}$, $\mathsf{op}_{\mathsf{M,move}}$, $\mathsf{op}_{\mathsf{M,repair}}$, $\mathcal{R}_M$ and so on. The facility team representation in Example 4.1 is denoted with a subscript F, such as $\mathcal{P}_{F,SAS}$, $\mathsf{at}_{\mathsf{F,S}}$, $\mathsf{health}_{\mathsf{F,S}}$, $\mathsf{health}_{\mathsf{F,S'}}$, $\mathsf{route}_{\mathsf{F,A}}$, $\mathsf{op}_{\mathsf{F,move}}$, $\mathsf{op}_{\mathsf{F,repair}}$, $\mathcal{R}_{\mathsf{F}}$ and so on.*

Combining the two models, we have the following joint model:

$$\mathcal{R} = \mathcal{R}_M \wedge \mathcal{R}_F.$$

A segment of the resulting joint state transition graph can be found in Figure 4.3.

Inconsistent partial views

For each joint state action pair $\langle s, a\rangle$, we say any two agents Ag_i and Ag_j's specifications are *consistent* on $\langle s, a\rangle$ iff

$$\mathcal{R}_i(s, a) \wedge \mathcal{R}_j(s, a) \neq FALSE.$$

Otherwise, we say that agent Ag_i and Ag_j are *inconsistent* on $\langle s, a\rangle$. In Section 5.2, I will discuss how to handle the inconsistent individual agents' partial views, and accommodate incrementally receiving information regarding individual agents' partial views.

4.3 Decentralized planning with partial views

We can plan the agents' joint activities the joint model in a centralized manner. This can be done by plugging the joint state transition system, the initial joint states and the joint goal states into the planning algorithms introduced in Section 3.5. Another approach is to do decentralized planning without the need to construct the joint state transition systems. Algorithm 7 is a way to do this

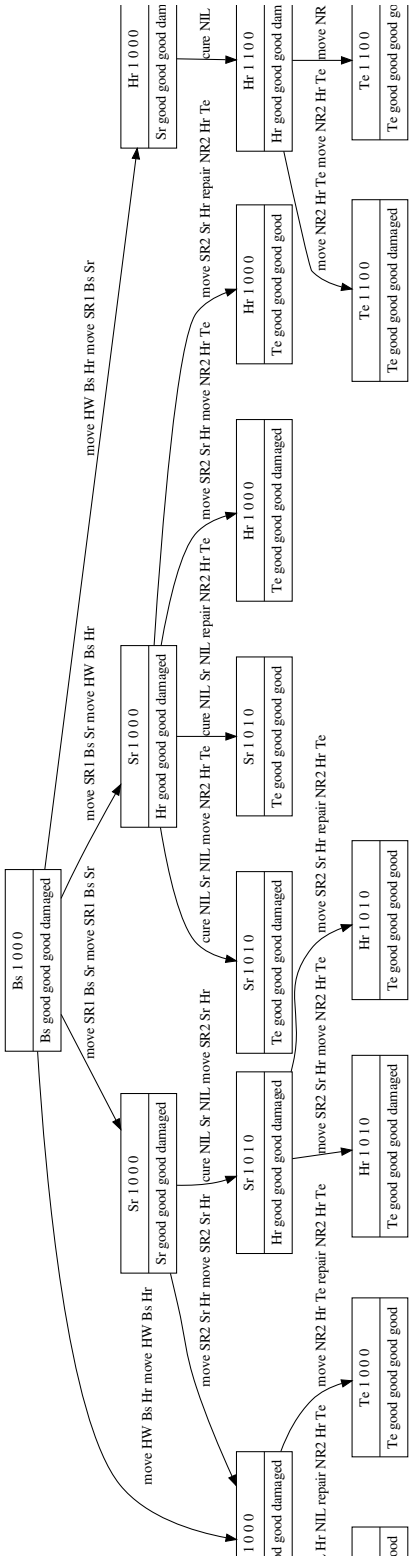

Figure 4.3: Part of the state transition graph for the joint NGO teams.

with the following function defined

$$combineR(relR_1, ..., relR_N) = \bigwedge_{\mathsf{Ag}_j \in \mathsf{AGS}} relR_j. \tag{4.1}$$

Algorithm 7: $DecPlan(\mathsf{Ag}_i, \mathsf{AGS}, I, G, computePreSA)$: Decentralized Planning

Input : (1) Ag_i: The agent who calls the procedure; (2) AGS: The known agent system; (3) I: The set of joint initial states; (4) G: The set of joint goals states; (5) $computePreSA$: Either $ComputeWeakPreSA$ or $ComputeStrongPreSA$

```
1  Solved ← G;
2  FT ← G;
3  newSA ← ∅;
4  while newSA ≠ ∅ OR I ⊆ Solved do
5  |  relR_i ← computeRelevantSAS(Ag_i, FT);
6  |  Send relR_i to every other agents Ag_j ∈ AGS (j ≠ i);
7  |  Wait for relR_j from the other agents Ag_j ∈ AGS (j ≠ i);
8  |  R ← combineR(relR_1, ..., relR_N) ;
9  |  newSA ← computePreSA(R, FT) ;
   |  /* Remove the state-action pairs whose states have already been solved  */
10 |  newSA ← newSA − Solved;
11 |  SA ← SA ∨ newSA ;
12 |  FT ← ∃_{P_A} newSA ;
13 |  Solved ← Solved ∨ FT;
14 end
15 if I ⊆ Solved then
16 |  return SA;
17 else
18 |  return ∅;
19 end
```

Before the execution of the decentralized Algorithm 7, the agents are assumed to have established a set of goal states that they want to achieve together. Thereafter the frontier states FT are initialized to a set of shared goal states. In the main loop of the algorithm, each agent Ag_i computes the state transitions in their local models relevant to the frontier FT by calling Algorithm 8 and then sending the relevant state transitions to the other agents. After obtaining these relevant transitions, each agent can then compute the joint state transitions that are relevant to the frontier.

Algorithm 8: *computeRelevantSAS*(Ag_i, FT): Compute agent Ag_i's joint state transitions relevant to FT

Input : (1) Ag_i: The agent who provide the information; (2) FT: The frontier states

1 $SA \leftarrow \exists_{\mathcal{P}_{S'}}(\mathcal{R}_i \wedge FT[\mathcal{P}_S/\mathcal{P}_{S'}])$;
2 $relR_i \leftarrow (SA \wedge \mathcal{R}_i)$;
3 **return** $relR_i$;

With these relevant joint state transitions, each agent then follows a planning procedure similar to that of the single agent planning (Algorithm 1).

Lemma 4.1 (Compute relevant joint state transitions). *Given the same frontier FT, in Algorithm 7 every agent will obtain the same set of relevant state transitions $\{relR_1, ..., relR_N\}$ with computeRelevantSAS (Algorithm 8).*

Proof. Every agent Ag_i computes its own relevant $relR_i$ locally with the same given frontier states FT:

$$relR_1 = computeRelevantSAS(\mathsf{Ag}_i, FT)$$

$$...$$

$$relR_N = computeRelevantSAS(\mathsf{Ag}_N, FT)$$

Then every agent Ag_i shares its locally computed relevant state transitions $relR_i$ with all other agents. As a result, all agents will obtain the same set of relevant state transitions $\{relR_1, ..., relR_N\}$.

Lemma 4.2 (Unique joint state transitions). *Given the same frontier FT, every agent will compute the same joint state transitions R with combineR (Equation 4.1) in in Algorithm 7.*

Proof. Lemma 4.1 guarantees that every agent will obtain a same set of locally relevant state transitions $\{relR_1, ..., relR_N\}$. Therefore every agent will have the same input the function

$$R = combineR(relR_1, ..., relR_N)$$

and compute the same joint state transitions R.

Proposition 4.1 (Soundness, completeness and consistency of decentralized planning). *Given a multi-agent system* $\mathsf{AGS} = \{\mathsf{Ag}_1, ..., \mathsf{Ag}_N\}$ *with a joint model* $\mathcal{M} = \langle \mathcal{P}_{SAS}, \mathcal{R} \rangle$ *and a list of individual partial views* $\mathcal{M}_i = \langle \mathcal{P}_{i,SAS}, \mathcal{R}_i \rangle$ *with*

$$\mathcal{R} = \bigwedge_i \mathcal{R}_i.$$

Let each agent $\mathsf{Ag}_i \in \mathsf{AGS}$ *be given the same set* I *of initial states and the same set* G *of goal states.*

- *Soundness: If* π *is a joint policy output by Algorithm 7, then* π *can achieve the goal states* G *from the initial states* I *in the joint state transition system* $\mathcal{M}$*.*
- *Completeness: If there is a policy* π *that can achieve the goal states* G *from the initial states* I *in the joint state transition system* $\mathcal{M}$*, then Algorithm 7 will output* π*.*
- *Consistency: If every agent follows Algorithm 7, then every agent will obtain the same joint policy* π*.*

Proof. By Lemma 4.2, Line 8 of Algorithm 7 guarantees that at each planning step every agent will compute the same relevant joint state transition R. With the same joint state transitions R, every agent follows the same planning procedure as in Algorithm 1. If a policy π can be reached, then every agent will obtain the same policy. This proves the consistency property.

As every agent's planning step is based on the same state transitions R, the soundness and completeness can then be directly derived from the soundness and completeness of Algorithm 1.

Complexity

Lemma 4.3 (Complexity for combining individual state transitions). *Let* $|\mathsf{AGS}|$ *be the number agents in an agent system* AGS*.* $combineR(relR_1, ..., relR_N)$ *(Equation 4.1) can be computed in* $O(|\mathsf{AGS}|)$ *BDD operations.*

Proof. It is obvious by counting the number of AND ($\wedge$) operations.

Lemma 4.4 (Complexity for computing relevant joint state transitions). *$computeRelevantSAS(\mathsf{Ag}_i, FT)$ (Algorithm 8) can be computed in $O(|\mathcal{P}_S|)$ BDD operations.*

Proof. Immediate by counting the number of BDD operations.

Proposition 4.2 (Complexity of decentralized planning). *Let D be the maximum length of paths in the joint state transition relation R. $DecPlan(\mathsf{Ag}_i, \mathsf{AGS}, I, G, computePreSA)$ (Algorithm 7) can be computed by agent Ag_i in $O(D \times (|\mathcal{P}_S| + |\mathcal{P}_A| + |\mathsf{AGS}|)) < O(2^{|\mathcal{P}_S|} \times (|\mathcal{P}_S| + |\mathcal{P}_A| + |\mathsf{AGS}|))$ BDD operations.*

Proof. The major loop of Algorithm 7 is almost the same as that of Algorithm 1 with two additional computations: $combineR(relR_1, ..., relR_N)$ and $computeRelevantSAS(\mathsf{Ag}_i, FT)$. According to Lemma 4.3 and Lemma 4.4, they can be computed with $O(|mcP_S|)$ and $O(|\mathsf{AGS}|)$ BDD operations respectively. Followed from the proof of Proposition 3.1, we have the BDD complexity for $DecPlan$ as $O(D \times (|\mathcal{P}_S| + |\mathcal{P}_A| + |\mathsf{AGS}|))$ which is also bounded by $2^{|\mathcal{P}_S|} \times (|\mathcal{P}_S| + |\mathcal{P}_A| + |\mathsf{AGS}|))$.

4.4 Agents' projected views and interaction analysis

In parallel to the individuals' partial views about the joint system, there is a projected view of an agent Ag_i.

$$\underline{\mathcal{M}}_i = \langle \mathcal{P}_{i,SAS}, \underline{\mathcal{R}}_i \rangle$$

where

- $\underline{\mathcal{R}}_i = \mathcal{R}^{\downarrow \mathsf{Ag}_i}$, and
- $dom(\underline{\mathcal{R}}_i) = \mathcal{P}_{i,SAS}$.

In the projected view, each agent Ag_i is only concerned about its own state action variables $\mathcal{P}_{i,SAS}$. We assume that without communication, each agent can not obtain the values of the other agents'

variables. Therefore, a projected view of an agent corresponds to a view in which the agent can effectively observe and act in the external world without any privileged knowledge of other agents. In the same manner, a joint policy π can be projected into individual policy $\underline{\pi}_i = \pi^{\downarrow \mathsf{Ag}_i}$ of agent Ag_i. Note that without communication an agent Ag_i can only observe the variables in $\mathcal{P}_{i,S}$ and execute the actions identified by $\mathcal{P}_{i,A}$.

The multiagent system transition system $\mathcal{R}$ contains information about the how multiple agents can interact with each other. Let the combination of individual agents' projected views be

$$\underline{\mathcal{R}} = \bigwedge_{\mathsf{Ag}_i \in ags} \underline{\mathcal{R}}_i.$$

Proposition 4.3 (Combined projected views). *The projections of individual agents combined together contains more uncertainty than that of the multiagent system as a whole:*

$$\mathcal{R} \subseteq \underline{\mathcal{R}}.$$

The measurement of uncertainty is in terms of the number of state transitions resulting from every state-action pair.

Proof. Since $\underline{\mathcal{R}}_i = \exists_{\mathcal{P}_{j,SAS}, j \neq i} \mathcal{R}$, it is straightforward to see that, for any agent $\mathsf{Ag}_i \in \mathsf{AGS}$, if $\langle s, a, s' \rangle \in \mathcal{R}$ then $\langle s, a, s' \rangle \in \underline{\mathcal{R}}_i$. Therefore if $\langle s, a, s' \rangle \in \mathcal{R}$ then $\langle s, a, s' \rangle \in \bigwedge_{\mathsf{Ag}_i \in \mathsf{AGS}} \underline{\mathcal{R}}_i$. This proves $\mathcal{R} \subseteq \underline{\mathcal{R}}$.

Definition 4.1 (Confusion state-action region). *A state-action region $CF(\mathcal{R}) \subseteq \mathcal{R}^{\downarrow \mathcal{P}_{SA}}$ of the joint state transition system $\mathcal{R}$ is said to be a* confusion state-action region *iff we have*

$$\bigwedge_{\mathsf{Ag}_i \in \mathsf{AGS}} \left[\underline{\mathcal{R}}_i(s^{\downarrow \mathsf{Ag}_i}, a^{\downarrow \mathsf{Ag}_i}) \right] \neq \mathcal{R}(s, a)$$

for each state transition $\langle s, a \rangle \in CF(\mathcal{R})$.

This captures the confusing joint state-action region of the state transitions in which an agent can not determine what to do given its local perceptions. Formally, if a state-action pair $\langle s, a\rangle \in CF(\mathcal{R})$ then $\underline{\mathcal{R}}(s, a) \neq \mathcal{R}(s, a)$. From Proposition 4.3, it is straightforward to see that for any $\langle s, a\rangle \in \mathcal{R}^{\mathcal{P}_{SA}}$, we have

$$\bigwedge_i \underline{\mathcal{R}}_i(s^{\downarrow \mathsf{Ag}_i}, a^{\downarrow \mathsf{Ag}_i}) \supseteq \mathcal{R}(s, a).$$

This means that if there is no communication among the agents about each other's local states and actions of each other, from individual agent's view, there are more possible next states when executing a state-action in the confusion region $CF(\mathcal{R})$.

Proposition 4.4 (Computing confusion state-action region). *The confusion region can be computed by*

$$CF = (\underline{\mathcal{R}} - \mathcal{R})^{\downarrow \mathcal{P}_{SA}}.$$

Proof. $\underline{\mathcal{R}} - \mathcal{R}$ contains all the confusing state transitions that are produced by the lack of the other agents' states and activities. $(\underline{\mathcal{R}} - \mathcal{R})^{\downarrow \mathcal{P}_{SA}}$ obtains the confusion state-action region of such state transitions.

In contrast to confusion state-action regions, we also have independent state-action regions.

Definition 4.2 (Independent state-action region). *A state-action region $IP(\mathcal{R}) \subseteq \mathcal{R}^{\downarrow \mathcal{P}_{SA}}$ is said to be an* independent state-action region *iff we have*

$$\bigwedge_i \underline{\mathcal{R}}_i(s^{\downarrow \mathsf{Ag}_i}, a^{\downarrow \mathsf{Ag}_i}) = \mathcal{R}(s, a)$$

for each state-action pair $\langle s, a\rangle \in IP$.

This captures the independent joint state-action region of the state transitions in which an agent can determine what to do given its local perceptions. From the above definition, we know that if a state-action pair $\langle s, a\rangle \in IP(\mathcal{R})$ then $\underline{\mathcal{R}}(s, a) = \mathcal{R}(s, a)$. This means that if there is no

communication among the agents about the local states and actions of each other, the outcome of a joint state-action $\langle s, a\rangle \in IP(\mathcal{R})$ can be precisely predicted through individual agents' local views.

Proposition 4.5 (Computing independent state-action region). *The independent state-action region can be computed by*

$$IP(\mathcal{R}) = \mathcal{R}^{\downarrow \mathcal{P}_{SA}} - CF(\mathcal{R}).$$

Proof. This can be drived directly from the proof of Proposition 4.4.

The concepts of confusion state-action region and independent state-action help us identify the communication need of a joint policy π:

- If a joint policy $\pi \subseteq IP(\mathcal{R})$, then individual agents can execute π without communicating with each other.
- If a joint policy $\pi \cap CF(\mathcal{R}) \neq \emptyset$, then to execute π correctly the agents must prepare to communicate with each others on their states and actions so as to narrow down the possible outcomes towards the goals of the systems.

4.5 Summary

This chapter contributes a model of multiagent state space. It is extended from the state space model of a single agent. Based on the multiagent state space, the agent's partial views, projected views and interactions among the agents are analysis. Centralized and decentralized policy planning algorithms are introduced to plan the behaviors towards the goals pre-assigned to the agents.

Chapter 5

A defeasible factored action theory

In this chapter, a defeasible factored action theory will be introduced to specify the state space of a single agent as well as that of a system of agents. The defeasible factored action theory allows the state space to be specified flexibly module by module. Further more, it allows defeasible (non-monotonic) reasoning about the state space — newly available information can disqualify existing information. In this way, individual agents can come up with competing views about the joint multiagent state space, exchange the specifications that describe these views with other agents, and perform defeasible reasoning to form a coherent joint view of the multiagent space as a whole. After the development of the defeasible action theory in this Chapter, Chapter 6 will develop planning and coordination algorithms for this defeasible action theory, and Chapter 7 will ground this defeasible action theory in an argumentation theory to provide well-defined semantics for the ad-hoc reasoning introduced in this chapter.

5.1 Motivations

In the literature of agent research [Wooldridge and Jennings, 1995; Wooldridge, 2002], agent theories are concerned about agents' mental models (e.g. beliefs, desires, and intentions) [Levesque, 1984; Bratman, 1990; Cohen and Levesque, 1990; Rao and Georgeff, 1991]; and agent architectures are

concerned about how agents, can be divided into components and how these components can work together to perceive the environment, reason, and determine how to act, e.g. deliberative architectures [Genesereth and Nilsson, 1987] (typically they contain a planning system, such as STRIPS, as a key component), reactive architecture [Rosenschein and Kaelbling, 1986; Brooks, 1986, 1991a,b; Kaelbling and Rosenschein, 1990], and hybrid architecture [Georgeff and Lansky, 1987; Ferguson, 1992; Mller and Pischel, 1994]. On the other hand, in the AI literature there are action theories [Thielscher, 2000; McCarthy, 1963; McCarthy and Hayes, 1969; Kowalski and Sergot, 1986; Reiter, 2001; Vo and Foo, 2005] which are concerned with specifying actions and drawing conclusions from the specifications; there are also practical planning techniques [Fikes and Nilsson, 1971; Chapman, 1987; McAllester and Rosenblitt, 1991; Soderland and Weld, 1991; Penberthy and Weld, 1992; Erol et al., 1994; Cimatti and Roveri, 2000; Cimatti et al., 2003] which are concerned with constructing plans for achieving goals.

The development of action theory focuses on solving three fundamental problems: the frame problem [McCarthy and Hayes, 1969], the qualification problem [McCarthy, 1977], and the ramification problem [Lin and Reiter, 1994]. The frame problem refers to the need to specify what does not change from situation to situation as actions are carried out. The qualification problem refers to need to specify precisely the conditions under which the effects of the action can take place. The ramification problem refers to the need to specify precisely the indirect effects of an action derived from the dependency among various world aspects. The difficulty of these three fundamental problems lies in the fact that the number of frame specifications, qualification conditions and ramification effects, as well as their possible combinations, can be huge or even infinite. It is also unavoidable to have inconsistent combinations of these frame specifications, qualification conditions and ramification effects. The kind of inconsistency can be very complicated. For example, a qualification condition might disqualify some action effects which then disqualify some ramification effects; as a result, the disqualified ramification effects might re-establish some of the frame specifications of the action. This implies that solving these three fundamental problems together inherently requires non-monotonic reasoning [Vo and Foo, 2005].

Many action theories has been proposed for single agent, including default logic [Reiter, 1980, 2001], circumscription [McCarthy, 1987], defeasible reasoning and argumentation [Simari et al., 2004; García et al., 2008; Vo and Foo, 2005], and so on (see [Vo and Foo, 2005] for a comprehensive survey on action theories). To the best of my knowledge, very little work has been done on applying non-monotonic action theory in multiagent systems and on building non-deterministic planning for non-monotonic action theories. This chapter will fill in the gap by constructing a defeasible action theory in multiagent systems. The contribution of this chapter is twofold:

- a defeasible multiagent action theory composed of competing specifications, and
- using symbolic model checking techniques to reason about the defeasible multiagent action theory.

From the agent architecture point of view [Wooldridge and Jennings, 1995], the defeasible multiagent action theory acts as a building block for a deliberative agent to reason about its actions, their applicable conditions, and their outcomes. On the other hand, the multiagent policy produced by non-deterministic planning algorithms in the next chapter (Chapter 6) can be viewed as producing reactive rules for the agents. These two components combined together compose a hybrid architecture for the agents, one where it is executing a reactive plan while deliberating about what to do in the future.

From agent theory point of view, [Wooldridge and Jennings, 1995] classified mental attitudes into *information attitude*s, such as beliefs and knowledge, and *pro-attitude*s, such as desire, intention, obligation, commitment, choice and so on. A defeasible specification implements the concept of information attitudes (i.e. beliefs of actions) in the agent theory. The underlying meaning of competing specifications resembles the competing rules that are used in reactive and hybrid agent architectures, but instead of having competing rules to specify which actions to take, in this defeasible action theory the specifications identify which state transitions are reasonable. In fact, the defeasible action theory proposed in this chapter along with the planning and coordination algorithms in Chapter 6 together act as a synthesizer which can produce reactive rules for agents

in a hybrid architecture. The next chapter (Chapter 6) will also present algorithms that can reason about how the agents' inconsistent goals, which are a special form of pro-mental attitudes in agent theory [Wooldridge and Jennings, 1995], can propagate through dynamics of the agents' actions.

5.2 Factored specification modules

A key approach in problem solving is modularity. Following the approach used in STRIPS [Fikes and Nilsson, 1971], in Section 3.1.2, an agent is specified by a list of operators $\mathcal{OP} = \{op_k\}$ with the requirements that $op_j \wedge op_k = \text{FALSE}$ for any two different operators op_j and op_k ($j \neq k$). Each operator captures how the world can be changed by the agent's operator. However, each operator op_k is required to specify all the changing aspects as well as all the unchanging aspects. In addition, $op_j \wedge op_k = \text{FALSE}$ should be enforced so that there are no interactions between any two operators. These requirements in the specifications of operators and interactions makes it difficult to specify even a single agent. When it comes to multiagent systems, these requirements become most formidable to achieve as every agent only has a partial view of the overall results. In this chapter, I propose a modularized specification system in which different aspects of the systems can be specified with different pieces of specification, and these can then be combined together to obtain a full image of the behaviors of the agents and how these agents interact with each other.

Definition 5.1 (Agent specification system). *In a system of modularized specifications, each agent* Ag_i *is specified by a list of specifications denoted by*

$$\mathcal{SPEC}_i = \{spec_{i,k}\}.$$

Each $spec_{i,k}$ *is associated with*

- *a* specification type*, denoted by* $type(spec_{i,k}) \in TYPE$*,*
- *a* specification layer*, denoted by* $layer(spec_{i,k}) \in LAYER$*, and*

- *a* level*, denoted by* $level(spec_{i,k}) \in LEVEL$.

In this work, the following types, layers, and levels are employed:

- $TYPE = \{SA, SAS\}$ specifies whether an item of information is in the state-action space $\mathcal{P}_{SA}$, denoted $type = SA$, or in the state transition space $\mathcal{P}_{SAS}$, denoted by $type = SAS$.
- $LAYER = \{IR, SPL, OP, FRM\}$ specifies whether an item of information is about multiagent interaction (IR), about specific cases (SPL), about the major aspects of the operators (OP), or about the frame axioms (FRM) defining what is not changed by default.
- $LEVEL = \Re$ is a real number [1].

The specification type ($TYPE$) provides meta information about whether a specification is concerned about the qualification conditions on the state-action space (SA) or the state transition space (SAS). When a specification is concerned with the state-action space, it will participate in reasoning about applicable actions; when a specification is concerned with the state transition space, it will participate in reasoning about the plausibility of the state transitions. The specification layer ($LAYER$) provides meta information about which aspect of the joint state transition system the specification is concerned with. When $LAYER$ is set to FRM, it means that the specification is about what is not changed in general. Usually FRM will be automatically set by the system from the ontology definition of the state space (i.e. one FRM specification for each fluent). The layer OP corresponds to the common sense specification of the major effects of an action. Multiple OP specifications are allowed for the same action, and furthermore these specifications for the same action can be inconsistent. On the other hand, an OP specification can also be about multiple actions. This is especially useful to specify the indirect effects which are derived from dependencies in the state ontology so that it can apply to any action that can cause the root effects in the indirect effect chains. SPL specifications are the agents' local understanding of exceptions

[1] In this work, as the lists of specifications are finite, we only use a finite subset of the real numbers. However real numbers are used to accept various kinds of measurements from which the meaning of levels are derived, such as probabilities, belief levels, certainty levels, costs, trust levels, and other measurements used in the AI literature.

to the specifications in the major effects specified by OP specifications. IR specifications are the agents' beliefs about the how different agents can interact. In this dissertation, it is assumed that the IR specifications can override the SPL specifications, and the SPL specifications can override the OP specifications, and the OP specifications can override the FRM specifications. The level associated with each specification is a place to hold the information, such as preference, trust and so on. The $TYPE$ and $LAYER$ of each specification is defining which component the specification belongs to; the *level* accommodates additional information that can be used to compare the state transitions that are composed of these specifications.

In a specific application, this meta information can be re-organized to reflect different kinds of action theory. Identifying what kind of action theories can specialized out of this defeasible action theory (it is actually a scheme) and the properties and applicability of these specialized action theories are directions for future research.

For notational convenience, we locate subsets of specifications by layer:

- $IR(\mathcal{SPEC}) = \{spec \mid spec \in \mathcal{SPEC} \text{ and } layer(spec) = IR\}$,
- $SPL(\mathcal{SPEC}) = \{spec \mid spec \in \mathcal{SPEC} \text{ and } layer(spec) = SPL\}$,
- $OP(\mathcal{SPEC}) = \{spec \mid spec \in \mathcal{SPEC} \text{ and } layer(spec) = OP\}$, and
- $FRM(\mathcal{SPEC}) = \{spec \mid spec \in \mathcal{SPEC} \text{ and } layer(spec) = FRM\}$.

We can also locate a subset of specifications by type:

- $SA(\mathcal{SPEC}) = \{spec \mid spec \in SPEC \text{ and } type(spec) = SA\}$, and
- $SAS(\mathcal{SPEC}) = \{spec \mid spec \in SPEC \text{ and } type(spec) = SAS\}$.

Before we extend our running example to be an example for a factored specification, we first extend the predicate signature of to be that of Example 5.1 to accommodate the complications in the coming examples that require factored information specifications.

Example 5.1 (NGO scenario: Predicate signature for defeasible action theory). *To allow us to capture more detail, the predicate language is extended as follows:*

$$\begin{aligned}
\Omega_{M,action} &= \{\mathsf{move}, \mathsf{cure}, \mathsf{stay}\} \\
\Omega_{F,action} &= \{\mathsf{move}, \mathsf{repair}, \mathsf{stay}, \mathsf{pump}, \mathsf{escort}\} \\
\Omega_{loc} &= \{\mathsf{Bs}, \mathsf{Sr}, \mathsf{Hr}, \mathsf{Te}\} \\
\Omega_{route} &= \{\mathsf{HW}, \mathsf{SR1}, \mathsf{SR2}, \mathsf{NR1}, \mathsf{NR2}\} \\
\Omega_{status} &= \{\mathsf{flooded}, \mathsf{good}\} \\
\mathsf{Pred}_{M,S} &= \mathsf{Pred}_{M,S'} \\
&= \{\mathsf{at}(\mathsf{loc}), \mathsf{health}(\mathsf{loc}), \mathsf{site_status}(\mathsf{loc}, \mathsf{status})\} \\
\mathsf{Pred}_{M,A} &= \{\mathsf{action}(\mathsf{name}), \mathsf{at}(\mathsf{loc}), \mathsf{to}(\mathsf{loc})\} \\
\mathsf{Pred}_{F,S} &= \mathsf{Pred}_{F,S'} \\
&= \{\mathsf{at}(\mathsf{loc}), \mathsf{route_status}(\mathsf{route_name}, \mathsf{status})\} \\
\mathsf{Pred}_{F,A} &= \{\mathsf{action}(\mathsf{name}), \mathsf{at}(\mathsf{loc}), \mathsf{to}(\mathsf{loc})\}
\end{aligned}$$

With this extension, the medical team is able to move *along roads,* cure *residents, and* stay *in any location. The facility team is able to* move *along roads,* repair *routes,* stay *in any location,* pump *water at a site, and* escort *the medical team through flooded areas. Among the predicates* $\mathsf{Pred}_{M,S}$ *and* $\mathsf{Pred}_{M,S'}$ *of the medical team states,* loc *of* at(loc) *is a single-valued parameter of domain* Ω_{loc}*;* loc *of* health(loc) *is a multi-valued parameter of domain* Ω_{loc}*;* loc *of* site_status(loc, status) *is a multi-valued parameter of domain* Ω_{loc} *and* status *of* site_status(loc, status) *is a single-valued parameter of domain* Ω_{status}*. Among the predicates* $\mathsf{Pred}_{F,S}$ *and* $\mathsf{Pred}_{f,S'}$ *of the facility team states,* loc *of* at(loc) *is a single-valued parameter of domain* Ω_{loc}*;* route_name *of* route_status(route_name, status) *is a multi-valued parameter of domain* Ω_{route} *and* status *of* route_status(route_name, status) *is a single-valued parameter of domain* Ω_{status}*. Regarding the action predicates* $\mathsf{Pred}_{M,A}$ *of the medical team,* name *of* action(name) *is a single-valued parameter of domain* $\Omega_{M,action}$*;* loc *of* at(loc) *is a*

single-valued parameter of the NIL-extended domain $\Omega^{-}_{M,loc}$*; and* loc *of* to(loc) *is a single-valued parameter of the NIL-extended domain* $\Omega^{-}_{M,loc}$*.*

With the predicate language set up, we can define the following action macros for the two agents. For the medical agent, using Pred_M *we have*

$$\begin{aligned}
\mathsf{move}(\mathsf{route_name}, \mathsf{orig}, \mathsf{dest}) &= \mathsf{action}(\mathsf{move}) \wedge \mathsf{route}(\mathsf{route_name}) \wedge \mathsf{at}(\mathsf{orig}) \wedge \mathsf{to}(\mathsf{dest}) \\
\mathsf{cure}(\mathsf{loc}) &= \mathsf{action}(\mathsf{cure}) \wedge \mathsf{at}(\mathsf{loc}) \wedge \mathsf{route}(\mathsf{NIL}) \wedge \mathsf{to}(\mathsf{NIL}) \\
\mathsf{stay}(\mathsf{loc}) &= \mathsf{action}(\mathsf{stay}) \wedge \mathsf{at}(\mathsf{loc}) \wedge \mathsf{route}(\mathsf{NIL}) \wedge \mathsf{to}(\mathsf{loc})
\end{aligned}$$

For the facility agent, using Pred_F *we have*

$$\begin{aligned}
\mathsf{move}(\mathsf{route_name}, \mathsf{orig}, \mathsf{dest}) &= \mathsf{action}(\mathsf{move}) \wedge \mathsf{route}(\mathsf{route_name}) \wedge \mathsf{at}(\mathsf{orig}) \wedge \mathsf{to}(\mathsf{dest}) \\
\mathsf{repair}(\mathsf{route_name}, \mathsf{orig}, \mathsf{dest}) &= \mathsf{action}(\mathsf{repair}) \wedge \mathsf{at}(\mathsf{orig}) \wedge \mathsf{route}(\mathsf{route_name}) \wedge \mathsf{to}(\mathsf{dest}) \\
\mathsf{stay}(\mathsf{loc}) &= \mathsf{action}(\mathsf{stay}) \wedge \mathsf{at}(\mathsf{loc}) \wedge \mathsf{route}(\mathsf{NIL}) \wedge \mathsf{to}(\mathsf{loc}) \\
\mathsf{escort}(\mathsf{route_name}, \mathsf{orig}, \mathsf{dest}) &= \mathsf{action}(\mathsf{escort}) \wedge \mathsf{at}(\mathsf{orig}) \wedge \mathsf{route}(\mathsf{route_name}) \wedge \mathsf{to}(\mathsf{dest}) \\
\mathsf{pump}(\mathsf{loc}) &= \mathsf{action}(\mathsf{pump}) \wedge \mathsf{at}(\mathsf{loc}) \wedge \mathsf{route}(\mathsf{NIL}) \wedge \mathsf{to}(\mathsf{NIL})
\end{aligned}$$

With the extended predicate language, we have a list of specifications as in Example 5.2 for the medical team agent.

Example 5.2 (NGO scenario: Medical team specifications). *(Cont. of Example 5.1) The effects of the medical team's actions in the general sense are specified by a list of operators* $\mathcal{SPEC}_{M,OP}$*. For each* spec $\in \mathcal{SPEC}_{M,OP}$ *we set* $\mathsf{type}(\mathsf{spec}) = SAS$*,* $\mathsf{layer}(\mathsf{spec}) = OP$*, and* $\mathsf{level}(\mathsf{spec}) = 1$ *(assuming equal preference level in this example). The predicate* connect *and the formula* background *are defined as in Example 3.3.* $\mathcal{SPEC}_{M,OP}$ *contains the following items:*

$$op_{M,move} = \mathsf{move}(\mathsf{route}, \mathsf{orig}, \mathsf{dest}) \wedge \mathsf{at}_{\mathsf{S}}(\mathsf{orig}) \wedge \mathsf{at}_{\mathsf{S}'}(\mathsf{dest})$$

$$\wedge \mathsf{connect}(\mathsf{route}, \mathsf{orig}, \mathsf{dest}) \wedge \mathsf{background}$$

$$op_{M,cure} = \mathsf{cure}(\mathsf{loc}) \wedge \mathsf{at}_{\mathsf{S}}(\mathsf{loc}) \wedge \neg \mathsf{health}_{\mathsf{S}}(\mathsf{loc}) \wedge \mathsf{health}_{\mathsf{S}'}(\mathsf{loc})$$

$$op_{M,stay} = \mathsf{stay}(\mathsf{loc}) \wedge \mathsf{at}_{\mathsf{S}}(\mathsf{loc}) \wedge \mathsf{at}_{\mathsf{S}'}(\mathsf{loc})$$

$op_{M,move}$ *says that if a* route *connects location* orig *and* dest *in background knowledge, the medical team can move from* orig *to* dest *through* route. $op_{M,cure}$ *says that if the medical team is at location* loc *and residents are not in a healthy status, then the medical team can successfully cure the residents.* $op_{M,stay}$ *says that the medical team can stay at its current location* loc.

However, there are cases where the general effects of the operators doesn't apply. Therefore, we have a list $\mathcal{SPEC}_{M,SPL}$ *of specifications which are special. For each* $\mathsf{spec} \in \mathcal{SPEC}_{M,SPL}$ *we set* $\mathsf{layer}(\mathsf{spec}) = SPL$, *and* $level(\mathsf{spec}) = 1$. $\mathcal{SPEC}_{M,SPL}$ *contains the following items:*

$$spl_{M,cure} = [\mathsf{site\_status}_{\mathsf{S}}(\mathsf{loc}, \mathsf{flooded}) \wedge \mathsf{action}(\mathsf{cure})$$
$$\wedge \mathsf{at}_{\mathsf{A}}(\mathsf{loc}) \wedge (\mathsf{health}_{\mathsf{S}}(\mathsf{loc}) \leftrightarrow \mathsf{health}_{\mathsf{S}'}(\mathsf{loc}))]$$

$$spl_{M,flooded} = \mathsf{route\_status}_{\mathsf{S}}(\mathsf{route\_name}, \mathsf{flooded}) \wedge \neg(\mathsf{action}(\mathsf{move}) \wedge \mathsf{route}_{\mathsf{A}}(\mathsf{route\_name}))$$

$spl_{M,cure}$ *says that if there is* flooded *in location* loc, *the medical team's* curing *has no effect — the residents can not be cured.* $spl_{M,flooded}$ *says that if a route is flooded, then the medical team cannot move through the route. In general, there are some specializations concerned with state transitions and some others about the state-action space. In this example,* $spl_{M,cure}$ *is concerned with the state transitions, therefore* $\mathsf{type}(spl_{M,cure}) = SAS$ *is set;* $spl_{M,flooded}$ *is about the state-action space, therefore* $\mathsf{type}(spl_{M,flooded}) = SA$ *is set. Also note that* $spl_{M,flooded}$ *mentions a predicate of the facility team's state space:* route_status.

In addition, as the specifications might not be able to cover every aspect of the world, we have a list $\mathcal{SPEC}_{M,FRM}$ *of frame specifications saying that if there are no specifications saying that there will be changes on certain aspects of the world, then such aspects will remain unchanged. For each*

$\mathsf{spec} \in \mathcal{SPEC}_{M,FRM}$ *we set* $\mathsf{type}(\mathsf{spec}) = SAS$, $\mathsf{layer}(spec) = FRM$, *and* $level(\mathsf{spec}) = 1$ *For the medical team,* $\mathcal{SPEC}_{M,FRM}$ *contains the following items:*

$$frm_{M,loc} = \mathsf{at_S}(\mathsf{loc}) \leftrightarrow \mathsf{at_{S'}}(\mathsf{loc})$$
$$frm_{M,health} = \mathsf{health_S}(\mathsf{loc}) \leftrightarrow \mathsf{health_{S'}}(\mathsf{loc})$$
$$frm_{M,status} = \mathsf{site\_status}_S(\mathsf{loc}, \mathsf{status}) \leftrightarrow \mathsf{site\_status}_{S'}(\mathsf{loc}, \mathsf{status})$$

In this example, the interaction specification list $\mathcal{SPEC}_{M,IR}$ *of the medical team is empty.*

Similarly, we have a list of specifications as in Example 5.3 for the facility team agent.

Example 5.3 (NGO scenario: Facility team specifications). *The effects of the facility team's actions in the general sense are specified by a list of operators* $\mathcal{SPEC}_{F,OP}$. *For each* $\mathsf{spec} \in \mathcal{SPEC}_{F,OP}$ *we set* $\mathsf{type}(\mathsf{spec}) = SAS$, $\mathsf{layer}(\mathsf{spec}) = OP$, *and* $\mathsf{level}(\mathsf{spec}) = 1$ *(assuming equal preference level in this example). The predicate* connect *and the formula* background *are defined as in Example 3.3.* $\mathcal{SPEC}_{F,OP}$ *contains the following items:*

$$\begin{aligned}
op_{F,move} = {} & \mathsf{move}(\mathsf{route}, \mathsf{orig}, \mathsf{dest}) \wedge \mathsf{at_S}(\mathsf{orig}) \wedge \mathsf{at_{S'}}(\mathsf{dest}) \\
& \wedge \mathsf{connect}(\mathsf{route}, \mathsf{orig}, \mathsf{dest}) \wedge \mathsf{background} \\
op_{F,repair} = {} & \mathsf{repair_{F,A}}(\mathsf{route_name}, \mathsf{orig}, \mathsf{dest}) \wedge \mathsf{at_{F,S}}(\mathsf{orig}) \wedge \mathsf{at_{F,S'}}(\mathsf{dest}) \\
& \wedge \mathsf{connect}(\mathsf{route_name}, \mathsf{orig}, \mathsf{dest}) \wedge \mathsf{background} \\
& \wedge \bigvee_{route_name} [\mathsf{route_A}(\mathsf{route_name}) \quad \wedge \mathsf{route_status_S}(\mathsf{route_name}, \mathsf{flooded}) \\
& \wedge \mathsf{route_status_{S'}}(\mathsf{route_name}, \mathsf{good})] \\
op_{F,escort} = {} & \mathsf{escort_{F,A}}(\mathsf{route_name}, \mathsf{orig}, \mathsf{dest}) \wedge \mathsf{at_{F,S}}(\mathsf{orig}) \wedge \mathsf{at_{F,S'}}(\mathsf{dest}) \\
& \wedge \mathsf{connect}(\mathsf{route_name}, \mathsf{orig}, \mathsf{dest}) \wedge \mathsf{background} \\
op_{F,pump} = {} & \mathsf{pump}(\mathsf{loc}) \wedge \mathsf{at_S}(\mathsf{loc}) \wedge \mathsf{site_status_S}(\mathsf{loc}, \mathsf{flooded}) \wedge \mathsf{site_status_{S'}}(\mathsf{loc}, \mathsf{good}) \\
op_{F,stay} = {} & \mathsf{stay}(\mathsf{loc}) \wedge \mathsf{at_S}(\mathsf{loc}) \wedge \mathsf{at_{S'}}(\mathsf{loc})
\end{aligned}$$

$op_{F,move}$ *says that if* route *connects location* orig *and* dest *in the background knowledge, the facility team can move from* orig *to* dest *through* route. $op_{F,repair}$ *says that if* route *connects location* orig *and* dest *in the background knowledge, and the route is flooded, the facility team can fix the route* route_name *and move from* orig *to* dest. $op_{F,escort}$ *says that if* route *connects location* orig *and* dest *in the background knowledge, the facility team can* escort *the medical team through route* route_name *and move from* orig *to* dest. $op_{F,pump}$ *says that the facility team can pump the flood at the location* loc*, and clear up the site into a good location.* $op_{F,stay}$ *says that the facility team can stay at the location* loc.

In this example, the specialization specification list $\mathcal{SPEC}_{F,IR}$ *of the facility team is empty.*

As the agents interact with each other, some of the interactions might override the operator and specialization specifications. Therefore, we have a list $\mathcal{SPEC}_{F,IR}$ *of specifications which are special. For each* spec $\in \mathcal{SPEC}_{F,IR}$ *we set* layer(spec) $= IR$*, and* level(spec) $= 1$*.* $\mathcal{SPEC}_{F,IR}$ *contains the following items:*

$$ir_{F,escort} = \mathsf{route\_status}(\mathsf{route\_name}, \mathsf{flooded}) \wedge (\mathsf{move}_{\mathsf{M,A}}(\mathsf{route\_name}, \mathsf{orig}, \mathsf{dest}) \wedge \mathsf{escort}_{\mathsf{F,A}}(\mathsf{route\_name}, \mathsf{orig}, \mathsf{dest}))$$

type($ir_{F,escort}$) *is set to* SA*.* $ir_{F,escort}$ *says that if it is flooded in a route, then the facility team must escort the medical team.*

In addition, as the specifications might not be able to cover every aspect of the world, we have a list $\mathcal{SPEC}_{F,FRM}$ *of frame specifications saying that if there are no specifications saying that there will be changes on certain aspects of the world, then such aspects will remain unchanged. For each* spec $\in \mathcal{SPEC}_{F,FRM}$ *we set* type(spec) $= SAS$*,* layer(spec) $= FRM$*, and* level(spec) $= 1$ *For the facility team,* $\mathcal{SPEC}_{F,FRM}$ *contains the following items:*

$$frm_{F,loc} = \mathsf{at}_{\mathsf{S}}(\mathsf{loc}) \leftrightarrow \mathsf{at}_{\mathsf{S}'}(\mathsf{loc})$$

$$frm_{F,route\_status} = \mathsf{route\_status}_{\mathsf{S}}(\mathsf{loc}, \mathsf{status}) \leftrightarrow \mathsf{route\_status}_{\mathsf{S}'}(\mathsf{loc}, \mathsf{status})$$

5.3 Specification combinations

Definition 5.2 (Mutual consistent specifications). *Two specifications $spec_j$ and $spec_k$ are* consistent *iff $spec_j \wedge spec_k \neq FALSE$. Two specifications $spec_j$ and $spec_k$ are* inconsistent *iff $spec_j \wedge spec_k = FALSE$.*

Definition 5.3 (Consistent set of specifications). *A set $SPEC$ of specifications is* consistent *iff $\bigwedge_{spec_k \in SPEC} spec_k \neq FALSE$. A set $SPEC$ of specifications is* inconsistent *iff $\bigwedge_{spec_k \in SPEC} spec_k = FALSE$.*

Given a list $SPEC$ of specifications, if the specifications are consistent with each other, we can combine them into a set of state-transitions by

$$Combine(SPEC) = \bigwedge_{spec_k \in SPEC} spec_k.$$

However, it not always the case that a list of specifications are all consistent with each other. An ad-hoc approach to combining specifications is to consider the combinations with respect to each state:

$$Combine(s, SPEC) = \bigwedge_{spec_k \in SPEC \text{ and } spec_k \wedge s \neq \text{FALSE}} s \wedge spec_k.$$

In this way, we can try to combine as much state transition information which mentions the specific state as possible. Similarly, we can combine the information with respect to a state-action pair:

$$Combine(s, a, SPEC) = \bigwedge_{spec_k \in SPEC \text{ and } spec_k \wedge s \wedge a \neq \text{FALSE}} s \wedge a \wedge spec_k.$$

However, neither of these approaches can guarantee that the specifications will be consistent with each other. We need a more systematic way to do the combination and underpin the combination with the appropriate semantics.

5.4 All possible consistent combinations

To reason about possible combinations of the specifications, we associate a label QBF variable with each specification $spec_{i,k}$:

$$l_{spec_{i,k}}.$$

The set of all these labels is denoted by $\mathcal{P}_L$. The agent Ag_i's labels are denoted by $\mathcal{P}_{i,L}$. Each label variable will interpreted as saying whether the corresponding specification is taken into the combination. For example, $spec_{i,k} \wedge l_{spec_{i,k}}$ will label the set of all consistent specifications that include $spec_{i,k}$.

Given a set $\mathcal{SPEC}$ of specifications, we can build up the set of all sets of consistent specifications by

$$LCONS(\mathcal{SPEC}) = \bigwedge_{spec_k \in \mathcal{SPEC}} (l_{spec_k} \rightarrow spec_k) - \bigwedge_{spec_k \in \mathcal{SPEC}} \neg l_{spec_k}. \tag{5.1}$$

Let

$$LSET(\mathcal{SPEC}) = \bigwedge_{spec_k \in \mathcal{SPEC}} (l_{spec_k}) \wedge \bigwedge_{spec_k \notin \mathcal{SPEC}} (\neg l_{spec_k}) \tag{5.2}$$

$LSET(\mathcal{SPEC})$ is the explicit QBF set encoding of the set $\mathcal{SPEC}$ with the label variables in $\mathcal{P}_L$ (see the encoding of sets of sets in Section 2.3.2).

Proposition 5.1 (Non-empty consistent combinations of specifications). *$LCONS(\mathcal{SPEC})$ encodes the set of all non-empty consistent labeled combinations.*

Proof. $\bigwedge_{spec_k \in \mathcal{SPEC}} (l_{spec_k} \rightarrow spec_k)$ can be expanded into a disjunction:

$$\bigvee_{\sigma \subseteq \mathcal{SPEC}} \bigwedge_{spec_k \in \sigma} (l_{spec_k} \wedge spec_k) \wedge \bigwedge_{spec_k \notin \sigma} \neg l_{spec_k}$$

$$= \bigvee_{\sigma \subseteq \mathcal{SPEC}} \left[\bigwedge_{spec_k \in \sigma} (l_{spec_k}) \wedge \bigwedge_{spec_k \notin \sigma} (\neg l_{spec_k}) \wedge \bigwedge_{spec_k \in \sigma} (spec_k) \right]$$

$$= \bigvee_{\sigma \subseteq \mathcal{SPEC} \text{ and } Combine(\sigma) \neq \text{FALSE}} \left[\bigwedge_{spec_k \in \sigma} (l_{spec_k}) \wedge \bigwedge_{spec_k \notin \sigma} (\neg l_{spec_k}) \wedge Combine(\sigma) \right]$$

$$= \bigvee_{\sigma \subseteq \mathcal{SPEC} \text{ and } Combine(\sigma) \neq \text{FALSE}} [LSET(\sigma) \wedge Combine(\sigma)]$$

In the disjunction, each term corresponds to a subset σ of $\mathcal{SPEC}$. Each term explicitly encodes a subset σ as $LSET(\sigma)$ and its combination $Combine(\sigma)$. Because only consistent subsets of specifications can make $Combine(\sigma)$ true, $\bigwedge_{spec_k \in \mathcal{SPEC}} (l_{spec_k} \rightarrow spec_k)$ encodes the disjunction of all consistent subsets of $\mathcal{SPEC}$. By removing the empty subset of $\mathcal{SPEC}$, we can obtain the sets of all possible non-empty consistent subsets of $\mathcal{SPEC}$. Removing the empty set can be done by the set-minus of the empty set

$$\bigwedge_{spec_k \in \mathcal{SPEC}} \neg l_{spec_k}.$$

Proposition 5.2 (Complexity of non-empty consistent combinations of specifications). *Let $|\mathcal{SPEC}|$ be the number of specifications in $\mathcal{SPEC}$. $LCONS(\mathcal{SPEC})$ (Equation 5.1) can be computed with $O(|\mathcal{SPEC}|)$ BDD operations.*

Proof. Immediate by counting the number of BDD operations in Equation 5.1.

5.5 Maximal consistent subsets of specifications

$LCONS(\mathcal{SPEC})$ gives us the set of all consistent subsets of combinations of $\mathcal{SPEC}$. However, some consistent combinations contain more information than others. For each state or state-action pair, we would like to have as much information as possible. To do this, we need a way to identify the maximal subsets of combinations.

Following the QBF encoding of sets of sets introduced in Section 2.3.2, we can build up a subset (superset) relation and proper subset (superset) relation over a set $\mathcal{SPEC}$ of specifications with two

copies of its label variables $\mathcal{P}_{L,SPEC}$ and $\mathcal{P}'_{L,SPEC}$ by

$$supseteq(\mathcal{SPEC}) = \bigwedge_{spec_k \in \mathcal{SPEC}} (l'_{spec_k} \rightarrow l_{spec_k}) \tag{5.3}$$

$$supseteq'(\mathcal{SPEC}) = \bigwedge_{spec_k \in \mathcal{SPEC}} (l_{spec_k} \rightarrow l'_{spec_k}) \tag{5.4}$$

$$subseteq(\mathcal{SPEC}) = \bigwedge_{spec_k \in \mathcal{SPEC}} (l_{spec_k} \rightarrow l'_{spec_k}) \tag{5.5}$$

$$subseteq'(\mathcal{SPEC}) = \bigwedge_{spec_k \in \mathcal{SPEC}} (l'_{spec_k} \rightarrow l_{spec_k}) \tag{5.6}$$

$$supset(\mathcal{SPEC}) = supseteq(\mathcal{SPEC}) \wedge \neg subseteq(\mathcal{SPEC}) \tag{5.7}$$

$$supset'(\mathcal{SPEC}) = supseteq'(\mathcal{SPEC}) \wedge \neg subseteq'(\mathcal{SPEC}) \tag{5.8}$$

$$subset(\mathcal{SPEC}) = subseteq(\mathcal{SPEC}) \wedge \neg supseteq(\mathcal{SPEC}) \tag{5.9}$$

$$subset'(\mathcal{SPEC}) = subseteq'(\mathcal{SPEC}) \wedge \neg supseteq'(\mathcal{SPEC}) \tag{5.10}$$

The above QBF formulae on $\mathcal{P}_{L,SPEC}$ and $\mathcal{P}'_{L,SPEC}$ effectively encode the following relations:

$$supseteq(\mathcal{SPEC}) = \{\langle \sigma, \sigma' \rangle | \sigma \supseteq \sigma',\ \sigma, \sigma' \subseteq \mathcal{SPEC}\}$$

$$supseteq'(\mathcal{SPEC}) = \{\langle \sigma', \sigma \rangle | \sigma \supseteq \sigma',\ \sigma, \sigma' \subseteq \mathcal{SPEC}\}$$

$$subseteq(\mathcal{SPEC}) = \{\langle \sigma, \sigma' \rangle | \sigma \subseteq \sigma',\ \sigma, \sigma' \subseteq \mathcal{SPEC}\}$$

$$subseteq'(\mathcal{SPEC}) = \{\langle \sigma', \sigma \rangle | \sigma \subseteq \sigma',\ \sigma, \sigma' \subseteq \mathcal{SPEC}\}$$

$$supset(\mathcal{SPEC}) = \{\langle \sigma, \sigma' \rangle | \sigma \supset \sigma',\ \sigma, \sigma' \subseteq \mathcal{SPEC}\}$$

$$supset'(\mathcal{SPEC}) = \{\langle \sigma', \sigma \rangle | \sigma \supset \sigma',\ \sigma, \sigma' \subseteq \mathcal{SPEC}\}$$

$$subset(\mathcal{SPEC}) = \{\langle \sigma, \sigma' \rangle | \sigma \subset \sigma',\ \sigma, \sigma' \subseteq \mathcal{SPEC}\}$$

$$subset'(\mathcal{SPEC}) = \{\langle \sigma', \sigma \rangle | \sigma \subset \sigma',\ \sigma, \sigma' \subseteq \mathcal{SPEC}\}$$

The QBF of $subseteq(\mathcal{SPEC})$ says that if an element $spec_k$ (encoded by l_{spec_K}) is in a set σ (encoded

with variables in $\mathcal{P}_{L,SPEC}$) then such an element must also be in a set σ' (encoded with variables in $\mathcal{P}'_{L,SPEC}$). The QBF of $supseteq(\mathcal{SPEC})$ says that if an element $spec_k$ (encoded by l'_{spec_K}) is in a set σ' (encoded with variables in $\mathcal{P}'_{L,SPEC}$) then such an element must also be in a set σ (encoded with variables in $\mathcal{P}_{L,SPEC}$). The $subset(\mathcal{SPEC})$ and $supset(\mathcal{SPEC})$ encodes the proper subset and super set relation respectively by excluding the relations in which the two sets σ and σ' are equal. The other equations can be interpreted similarly.

Proposition 5.3 (Complexity of set relations over consistent combinations). *The subset and superset relations (Equation 5.3—5.10) between subsets of $\mathcal{SPEC}$ of specifications can be computed in $O(|\mathcal{SPEC}|)$ BDD operations.*

Proof. Immediate by counting the number of BDD operations in Equation 5.3—5.10.

In reasoning about actions and state transitions, we are interested in identifying all the consistent subsets of combinations with respect to the frames of discernment in states and state-action pairs. We call this kind of frames of discernment the *WRT frame* and denote it by $\mathcal{P}_{WRT}$. The consistent combinations with respect to a *WRT* frame are called *WRT-consistent combination*s. *WRT*-consistent combinations can better be understood as a conditional probability distribution $\Pr(\vec{X}|\vec{Y})$ which ranges over discrete boolean values $\{0, 1\}$. The *WRT* frame defines the space of conditions over which $\vec{Y}$ can vary. Assuming the Markov property on the state-action space — the applicability of an action only depends the current state — setting the *WRT*-consistent combinations' *WRT* frame to be the state frame $\mathcal{P}_S$ enables us to reason about the set of applicable actions over these states producing a *valid state-action space*. Assuming the Markov property on the state transition space — the next state only depends the current state and the current action — setting the *WRT*-consistent combinations' *WRT* frame to be the state-action frame $\mathcal{P}_{SA}$ enables us to reason about the set of plausible state transitions over these state-actions producing a *valid state transition space*. This idea can be implemented by projecting the set of labeled consistent combinations onto the frame of $\mathcal{P}_{WRT} \cup \mathcal{P}_L$ to encode which subsets of combinations are consistent

on each interpretation of the with-respect-frame $\mathcal{P}_{WRT}$:

$$\begin{aligned} LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC}) &= (LCONS(\mathcal{SPEC}))^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L} \\ &= \exists_{dom(\mathcal{SPEC}) \cup \mathcal{P}_L \setminus (\mathcal{P}_{WRT} \cup \mathcal{P}_L)} LCONS(\mathcal{SPEC}) \end{aligned} \tag{5.11}$$

where the domain $dom(\mathcal{SPEC})$ of the specifications is $\mathcal{P}_{SAS}$.

Proposition 5.4 (Complexity of *WRT*-consistent combinations). *$LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC})$ (Equation 5.11) can be computed in $O(|\mathcal{SPEC}| + |\mathcal{P}_{SAS}|)$ BDD operations.*

Proof. $LCONS(\mathcal{SPEC})$ can be computed with $O(|\mathcal{SPEC}|)$ BDD operations (Proposition 5.2). Then it is followed by $|dom(\mathcal{SPEC}) \cup \mathcal{P}_L \setminus (\mathcal{P}_{WRT} \cup \mathcal{P}_L)|$ existential quantifications which is bounded by $|\mathcal{P}_{SAS}|$ specifications. In total, the computation is bounded by $O(|\mathcal{SPEC}|+|\mathcal{P}_{SAS}|)$ BDD operations.

Now we can build up the concepts of *WRT-subset relation* and *WRT-superset relation*. A *WRT*-subset relation encodes a collection of subset relations. Each subset relation is defined over all consistent combinations that are consistent with a truth assignment to the *WRT* frame. For example, if a *WRT* frame is set to the state frame $\mathcal{P}_S$, then the *WRT*-subset relation is a collection of subset relations each of which corresponds to a state and the subset relation is over all consistent combinations that can be constructed for such a state. Similarly, a *WRT*-superset relation encodes a collection of superset relations. Each superset relation is defined over all consistent combinations that are consistent with a truth assignment to the *WRT* frame. For example, if a *WRT* frame is set to the state frame $\mathcal{P}_S$, then the *WRT*-superset relation is a collection of superset relations each of which corresponds to a state and the superset relation is over all consistent combinations that can be constructed for such a state.

$$\begin{aligned} lsupseteq(\mathcal{P}_{WRT}, \mathcal{SPEC}) &= supseteq(\mathcal{SPEC}) \\ &\quad \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}'_L}(\mathcal{SPEC}) \\ lsupseteq'(\mathcal{P}_{WRT}, \mathcal{SPEC}) &supseteq'(\mathcal{SPEC}) \end{aligned} \tag{5.12}$$

$$\wedge\, LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}'_L}(\mathcal{SPEC}) \tag{5.13}$$

$$lsubseteq(\mathcal{P}_{WRT}, \mathcal{SPEC}) = subseteq(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}'_L}(\mathcal{SPEC}) \tag{5.14}$$

$$lsubseteq'(\mathcal{P}_{WRT}, \mathcal{SPEC}) = subseteq'(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}'_L}(\mathcal{SPEC}) \tag{5.15}$$

$$lsupset(\mathcal{P}_{WRT}, \mathcal{SPEC}) = supset(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}'_L}(\mathcal{SPEC}) \tag{5.16}$$

$$lsupset'(\mathcal{P}_{WRT}, \mathcal{SPEC}) = supset'(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}'_L}(\mathcal{SPEC}) \tag{5.17}$$

$$lsubset(\mathcal{P}_{WRT}, \mathcal{SPEC}) = subset(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}'_L}(\mathcal{SPEC}) \tag{5.18}$$

$$lsubset'(\mathcal{P}_{WRT}, \mathcal{SPEC}) = subset'(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC}) \wedge LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}'_L}(\mathcal{SPEC}) \tag{5.19}$$

Proposition 5.5 (Complexity for set relations over WRT-consistent combinations). *The labeled subset and superset relation (Equation 5.12—5.19) WRT-frame $\mathcal{P}_{WRT}$ between subsets of $\mathcal{SPEC}$ of specifications can be computed in $O(|\mathcal{SPEC}| + |\mathcal{P}_{SAS}|)$ BDD operations.*

Proof. Every form of the labeled subset and superset relations is composed of plain subset and superset relations and two labeled-with-respect-to-frame sets of consistent combinations. From Proposition 5.3, the every form of subset and superset relation can be computed by $O(|\mathcal{SPEC}|)$ number of BDD operations. The labeled-with-respect-to sets of consistent combinations can be computed with $O(|\mathcal{SPEC}| + |\mathcal{P}_{SAS}|)$ BDD operations (see Proposition 5.4). In total, the computation is bounded by $O(|\mathcal{SPEC}| + |\mathcal{P}_{SAS}|)$ BDD operations.

Definition 5.4 (Maximal combinations). *Given a set of combinations $Q \subseteq \mathcal{SPEC}$ and a partial*

relation $B \subseteq \mathcal{SPEC} \times \mathcal{SPEC}'$ *(e.g. the superset* $\supseteq$ *relation on the combinations) on* $\mathcal{SPEC}$*, the set of maximal combinations in* Q *with respect to* B *is*

$$Maximal(Q, B) = \{A \in Q | \textit{for all } C \in Q, (C, A) \in B \textit{ implies } (A, C) \in B\}.$$

Proposition 5.6 (*WRT*-maximal consistent combinations). *With the above constructs, we compute the maximal subsets of consistent specification combinations with respect to set inclusion* ($\subseteq$) *for each interpretation of the WRT-frame* $\mathcal{P}_{WRT}$ *by:*

$$\begin{aligned} maximal(\mathcal{P}_{WRT}, \mathcal{SPEC}) = \\ & LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC}) \\ & \wedge \left[\forall_{\mathcal{P}'_L} \left(lsupseteq'(\mathcal{P}_{WRT}, \mathcal{SPEC}) \rightarrow lsupseteq(\mathcal{P}_{WRT}, \mathcal{SPEC})\right)\right] \end{aligned} \quad (5.20)$$

Proof. This is a direct translation of Definition 5.4. $LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC})$ guarantees that the set of consistent combinations for each interpretation of $\mathcal{P}_{WRT}$ is included. $\forall_{\mathcal{P}'_L}(lsupseteq'(\mathcal{P}_{WRT}, \mathcal{SPEC}) \rightarrow lsupseteq(\mathcal{P}_{WRT}, \mathcal{SPEC}))$ captures the definition of maximal set of combinations for each interpretation of $\mathcal{P}_{WRT}$ according to Definition 5.4.

Formally, let $\mathbf{sl} \in 2^{\mathcal{P}_L}$ be a truth-assignment vector of $\mathcal{P}_L$ and **wrt** be a truth-assignment vector of $\mathcal{P}_{WRT}$ such that $\mathbf{sl} \wedge \mathbf{wrt}$ satisfies $maximal(\mathcal{P}_{WRT}, \mathcal{SPEC})$. We have in every truth-assignment $\langle \mathbf{wrt}, \mathbf{sl} \rangle$ to $\mathcal{P}_{WRT} \cup \mathcal{P}_L$ that

- **sl** must satisfy $LCONS(\mathcal{SPEC})$; and
- **sl** must also satisfy $\forall_{\mathcal{P}'_L}(lsupseteq'(\mathcal{SPEC}) \rightarrow lsupseteq(\mathcal{SPEC}))$ because
  - if $\langle \mathbf{sl}', \mathbf{sl} \rangle \in lsupseteq'(\mathcal{SPEC})$, then $\langle \mathbf{sl}, \mathbf{sl}' \rangle \in lsupseteq(\mathcal{SPEC})$. Namely if there is a $\mathbf{sl}' \supseteq \mathbf{sl}$ then it must be the case that $\mathbf{sl} \supseteq \mathbf{sl}'$,
  - if it is not the case that $\langle \mathbf{sl}', \mathbf{sl} \rangle \in lsupseteq'(\mathcal{SPEC})$, then **sl** can satisfy every term in the conjunction expansion of the QBF universal quantification over the variables in $\mathcal{P}'_L$

on $(lsupseteq'(\mathcal{SPEC}) \rightarrow lsupseteq(\mathcal{SPEC}))$.

Proposition 5.7 (Complexity of *WRT*-maximal consistent combinations). *The maximally consistent with-respect-to-frame subsets* $maximal(\mathcal{P}_{WRT}, \mathcal{SPEC})$ *(Equation 5.20) can be computed in* $O(|\mathcal{SPEC}| + |\mathcal{P}_{SAS}|)$ *BDD operations.*

Proof. $LCONS^{\downarrow \mathcal{P}_{WRT} \cup \mathcal{P}_L}(\mathcal{SPEC})$ can be computed with $O(|\mathcal{SPEC}| + |\mathcal{P}_{SAS}|)$ BDD operations (Proposition 5.4). $lsupseteq'(\mathcal{P}_{WRT}, \mathcal{SPEC})$ and $lsupseteq(\mathcal{P}_{WRT}, \mathcal{SPEC})$ can also be computed with $O(|\mathcal{SPEC}|+|\mathcal{P}_{SAS}|)$ BDD operations (Proposition 5.5). Together with $|\mathcal{P}_L| = |\mathcal{SPEC}|$ universal quantifications, we can compute $maximal(\mathcal{P}_{WRT}, \mathcal{SPEC})$ in $O(|\mathcal{SPEC}| + |\mathcal{P}_{SAS}|)$ BDD operations.

Two special cases are maximal combinations of state-action space and state transitions:

$$maximal_{SA}(\mathcal{SPEC}) = maximal(\mathcal{P}_S, SA(\mathcal{SPEC}))$$
$$maximal_{SAS}(\mathcal{SPEC}) = maximal(\mathcal{P}_{SA}, SAS(\mathcal{SPEC}))$$

$maximal_{SA}(\mathcal{SPEC})$ encodes the set of maximally consistent combinations for each state, therefore it defines the valid state-action space for each state as the maximally consistent combinations of the list of state-action specifications, namely $SA(\mathcal{SPEC})$. $maximal_{SAS}(\mathcal{SPEC})$ encodes the set of maximally consistent combinations for each state-action, therefore it defines the valid state transitions for each state-action as the maximally consistent combinations of the list of state-action specifications, namely $SA(\mathcal{SPEC})$.

5.6 Combination preference levels

In Definition 5.1, each specification $spec_k$ is associated with a level $level(spec_k)$. In the applications, these levels are used to capture various kinds of preferences and derived preferences such as probability, possibility, plausibility and so on. These preferences can help us narrow down the possibilities

among the consistent combinations of specifications. We should assume that these measurement systems will eventually output a (partial) ordering $PREF$ on specification combinations.

Definition 5.5 (Minimum level). *Given a set $\mathcal{SPEC}$ of specifications, we define the minimum level of $\mathcal{SPEC}$ as*

$$minLevel(\mathcal{SPEC}) = \begin{cases} min\{level(spec) \mid spec \in \mathcal{SPEC}\} & \text{if } \mathcal{SPEC} \neq \emptyset \\ -\infty & \text{if } \mathcal{SPEC} = \emptyset \end{cases}$$

Let $\mathcal{P}_{PREF}$ be the set of QBF variables to encode the levels assigned to the specifications in $\mathcal{SPEC}$. Assume that there are M different levels assigned to $\mathcal{SPEC}$, we can then encode all possible levels with

$$\lceil log_2 M \rceil$$

variables, namely $|\mathcal{P}_{PREF}| = \lceil log_2 M \rceil$.

Given a list $\mathcal{SPEC}$ of specifications, the following QBF can assign to the encoding of every consistent combination the levels of its members:

$$LCONSV(\mathcal{SPEC}) = LCONS(\mathcal{SPEC}) \wedge \bigvee_{spec_k \in \mathcal{SPEC}} [l_{spec_k} \wedge \xi\left(level(spec_k)\right)] \tag{5.21}$$

Proposition 5.8 (Complexity of $LCONSV(\mathcal{SPEC})$ (Equation 5.21)). *$LCONSV(\mathcal{SPEC})$ can be computed in $O(|\mathcal{SPEC}|)$ BDD operations.*

Proof. $LCONS(\mathcal{SPEC})$ can be computed in $O(|\mathcal{SPEC}|)$ BDD operations (see Proposition 5.2). $\bigvee_{spec_k \in \mathcal{SPEC}} [l_{spec_k} \wedge \xi\left(level(spec_k)\right)]$ is also in $O(|\mathcal{SPEC}|)$ BDD operations. In total, we can compute $LCONSV(\mathcal{SPEC})$ in $O(|\mathcal{SPEC}|)$ BDD operations.

Example 5.4 (Consistent combinations labeled by member levels).
Let $\mathcal{SPEC} = \{\langle spec_1, 1\rangle, \langle spec_2, 2\rangle, \langle spec_3, 3\rangle\}$ (where 1 is encoded by $(01)_2$, 2 is encoded by $(10)_2$ and 3 is encoded by $(11)_2$), and $spec_1$ is consistent with $spec_2$, and $spec_2$ is consistent with $spcec_3$,

and $spec_3$ is consistent with $spec_1$ otherwise all other combinations are inconsistent. Then we will have

$$\begin{aligned} LCONSV(\mathcal{SPEC}) = \{ & \\ & \langle \{spec_1, spec_2\}, (01)_2 \rangle \\ & \langle \{spec_1, spec_2\}, (10)_2 \rangle \\ & \langle \{spec_2, spec_3\}, (10)_2 \rangle \\ & \langle \{spec_2, spec_3\}, (11)_2 \rangle \\ & \langle \{spec_1, spec_3\}, (10)_2 \rangle \\ & \langle \{spec_1, spec_3\}, (11)_2 \rangle \\ \} & \end{aligned}$$

With the input of $LCONSV(\mathcal{SPEC})$, we can compute the maximum and minimum levels of all these subsets, and associate the maximum and minimum level to these subsets with Algorithm 9 and Algorithm 10 respectively. The effect of $\mathcal{P}_{WRT}$ is to have the minimization and maximization compare the levels of all the consistent combinations for each truth assignment given to WRT-frame (see Example 5.5 and Example 5.6).

Proposition 5.9 (Complexity of minimum level labeling).
Let $LCombineV = LCONSV(\mathcal{SPEC})$ (Equation 5.21). $minLevel(\mathcal{P}_{WRT}, LCombineV)$ (Algorithm 9) computes all consistent subsets of $\mathcal{SPEC}$ and associates to each subset the minimum preference level among the members of such a subset.

Proof. In Algorithm 9, the bits of the level encoding are examined from the most significant bit to the least significant bit. For each bit bit_k, the algorithm selects the smaller levels by conjoining the previously computed combinations $minLCombineV$ with $\neg bit_k$. On the other hand $(minLCombineV - smaller^{\downarrow \mathcal{P}_{WRT}})$ preserves the combinations (in $\mathcal{P}_L$) and the with-respect-to

Algorithm 9: $minLevel(\mathcal{P}_{WRT}, LCombineV)$: Compute the set of consistent combinations labeled by the minimum level of its members

Input : 1) $\mathcal{P}_{WRT}$: The frame variables over whose interpretations the combinations are built;
2) $LCombineV$: A labeled set of consistent combinations with levels of its members attached

1 **for** *each* $bit_k \in \mathcal{P}_{PREF}$ *in an descending order of significance in the level encoding* **do**
2 $\quad minLCombineV \leftarrow LCombineV$;
$\quad$ `/* Locate the combinations with smaller levels */`
3 $\quad smaller \leftarrow minLCombineV \wedge \neg bit_k$;
$\quad$ `/* Remove all other levels associated with the located combinations */`
4 $\quad minLCombineV \leftarrow smaller \vee (minLCombineV - smaller^{\downarrow \mathcal{P}_{WRT}})$;
5 **end**
6 **return** $minLCombineV$;

frame configurations (in $\mathcal{P}_{WRT}$)) of the subsets in whose level encoding bit_k is set. After the smaller levels are selected, the algorithm removes all other levels associated with the located combinations on each truth-assignment to the $\mathcal{P}_{WRT}$ frame by excluding (set minus) $smaller^{\downarrow \mathcal{P}_{WRT}}$.

Example 5.5 (Minimal level consistent combinations). *In addition to the assumptions made in Example 5.4, let's further assume* $\{x_1, x_2, x_3\} \subseteq \mathcal{P}_{SAS}$, $\mathcal{P}_{WRT} = \{x_1, x_2\}$ *and* $spec_1$ *is a specification that satisfies* $x_1 \wedge x_2$, $spec_2$ *is a specification that satisfies* x_2, *and* $spec_3$ *is a specification that satisfies* $x_2 \wedge x_3$.

$$
\begin{aligned}
LCONSV(\mathcal{SPEC}) = \{ & \\
& \langle x_1 \wedge x_2 \wedge \{spec_1, spec_2\}, (01)_2 \rangle \\
& \langle x_1 \wedge x_2 \wedge \{spec_1, spec_2\}, (10)_2 \rangle \\
& \langle \neg x_1 \wedge x_2 \wedge x_3 \wedge \{spec_2, spec_3\}, (10)_2 \rangle \\
& \langle \neg x_1 \wedge x_2 \wedge x_3 \wedge \{spec_2, spec_3\}, (11)_2 \rangle \\
& \langle x_1 \wedge x_2 \wedge x_3 \wedge \{spec_1, spec_3\}, (10)_2 \rangle \\
& \langle x_1 \wedge x_2 \wedge x_3 \wedge \{spec_1, spec_3\}, (11)_2 \rangle
\end{aligned}
$$

Algorithm 10: $maxLevel(\mathcal{P}_{WRT}, LCombineV)$: Compute the set of consistent combinations labeled by the minimum level of its members

Input : 1) $\mathcal{P}_{WRT}$: The frame variables over whose interpretations the combinations are built;
2) $LCombineV$: A labeled set of consistent combinations with levels of its members attached

1 **for** *each* $bit_k \in \mathcal{P}_{PREF}$ *in an descending order of significant in the number encoding* **do**
2 $\quad maxLCombineV \leftarrow LCombineV$;
$\quad$ `/* Locate the combinations with smaller levels */`
3 $\quad bigger \leftarrow maxLCombineV \wedge bit_k$;
$\quad$ `/* Remove all other levels associated with the located combinations */`
4 $\quad maxLCombineV \leftarrow bigger \vee (maxLCombineV - bigger^{\downarrow \mathcal{P}_{WRT}})$;
5 **end**
6 **return** $maxLCombineV$;

$$\}$$

Let $LCombineV = LCONSV(\mathcal{SPEC})$. *The result of* $minLevel(\mathcal{P}_{WRT}, LCombineV)$ *(Algorithm 9) will be*

$$\begin{aligned} LCONSV(\mathcal{SPEC}) = \{ & \\ & \langle x_1 \wedge x_2 \wedge \{spec_1, spec_2\}, (01)_2 \rangle \\ & \langle \neg x_1 \wedge x_2 \wedge x_3 \wedge \{spec_2, spec_3\}, (10)_2 \rangle \\ \} & \end{aligned}$$

Note that as $\mathcal{P}_{WRT} = \{x_1, x_2\}$, *the following combinations are compared in the same group with respect to the configuration* $x_1 \wedge x_2$ *of* $\mathcal{P}_{WRT}$

$$\langle x_1 \wedge x_2 \wedge \{spec_1, spec_2\}, (01)_2 \rangle$$

$$\langle x_1 \wedge x_2 \wedge \{spec_1, spec_2\}, (10)_2 \rangle$$

$$\langle x_1 \wedge x_2 \wedge x_3 \wedge \{spec_1, spec_3\}, (10)_2 \rangle$$

$$\langle x_1 \wedge x_2 \wedge x_3 \wedge \{spec_1, spec_3\}, (11)_2 \rangle.$$

The minimization result is then only one item with respect to the configuration $x_1 \wedge x_2$ of $\mathcal{P}_{WRT}$:

$$\langle x_1 \wedge x_2 \wedge \{spec_1, spec_2\}, (01)_2 \rangle.$$

Proposition 5.10 (Complexity of maximum level labeling).
Let $LCombineV = LCONSV(\mathcal{SPEC})$ (Equation 5.21). $maxLevel(\mathcal{P}_{WRT}, LCombineV)$ (Algorithm 10) computes all consistent subsets of $\mathcal{SPEC}$ and associates to each subset the maximum preference level among the members of such a subset.

Proof. In Algorithm 9, the bits of the level encoding are examined from the most significant bit to the least significant bit. For each bit bit_k, the algorithm selects the bigger levels by conjoining the previous computed combinations $maxLCombineV$ with bit_k. On the other hand ($maxLCombineV - bigger^{\downarrow \mathcal{P}_{WRT}}$) preserves the combinations (in $\mathcal{P}_L$) and the with-respect-to frame configurations (in $\mathcal{P}_{WRT}$)) of the subsets in whose level encoding bit_k is not set. After the bigger levels are selected, the algorithm removes all other levels associated the located combinations on each truth-assignment to the frame defined by $\mathcal{P}_{WRT}$ by excluding (set minus) $bigger^{\downarrow \mathcal{P}_{WRT}}$. After going through all the bits encoding the levels, it preserves for each combination only the maximum level of the combination's members.

Example 5.6 (Maximal level consistent combinations). *Let $LCombineV = LCONSV(\mathcal{SPEC})$ of Example 5.5. The result of $maxLevel(\mathcal{P}_{WRT}, LCombineV)$ (Algorithm 10) will be*

$$\langle \neg x_1 \wedge x_2 \wedge x_3 \wedge \{spec_2, spec_3\}, (11)_2 \rangle$$
$$\langle x_1 \wedge x_2 \wedge x_3 \wedge \{spec_1, spec_3\}, (11)_2 \rangle$$

Again note that as $\mathcal{P}_{WRT} = \{x_1, x_2\}$, the following combinations are compared in the same group

with respect to the configuration $x_1 \wedge x_2$ *of* $\mathcal{P}_{WRT}$

$$\langle x_1 \wedge x_2 \wedge \{spec_1, spec_2\}, (01)_2 \rangle$$
$$\langle x_1 \wedge x_2 \wedge \{spec_1, spec_2\}, (10)_2 \rangle$$
$$\langle x_1 \wedge x_2 \wedge x_3 \wedge \{spec_1, spec_3\}, (10)_2 \rangle$$
$$\langle x_1 \wedge x_2 \wedge x_3 \wedge \{spec_1, spec_3\}, (11)_2 \rangle.$$

The maximization result is then only one item with respect to the configuration $x_1 \wedge x_2$ *of* $\mathcal{P}_{WRT}$*:*

$$\langle x_1 \wedge x_2 \wedge x_3 \wedge \{spec_1, spec_3\}, (11)_2 \rangle.$$

Proposition 5.11 (Complexity of minimal and maximal combinations). *The complexity of both* $minLevel(\mathcal{P}_{WRT}, LCombineV)$ *(Algorithm 9) and* $maxLevel(\mathcal{P}_{WRT}, LCombineV)$ *(Algorithm 10) is* $O(|\mathcal{P}_{PREF}| \cdot (|\mathcal{P}_{SAS}| + |\mathcal{SPEC}|)$ *BDD operations.*

Given a set $\mathcal{SPEC}$ *of specifications, both* $minLevel(\mathcal{P}_{WRT}, LCONSV(\mathcal{SPEC}))$ *and* $maxLevel(\mathcal{P}_{WRT}, LCONSV(\mathcal{SPEC}))$ *can be computed in* $O(|\mathcal{P}_{PREF}| \cdot (|\mathcal{P}_{SAS}| + |\mathcal{SPEC}|)$ *BDD operations.*

Proof. Both algorithms finish with a for loop of $|\mathcal{P}_{PREF}|$ steps. Each step requires 3 BDD operations and a sequence of existential quantification which implements the information projection to $\mathcal{P}_{WRT}$. The number of variables outside the set $\mathcal{P}_{WRT}$ is bounded by the number of variables in $\mathcal{P}_{SAS}$ plus the number of specification labels which is equal to the number of specifications in $|\mathcal{SPEC}|$. Therefore, both algorithms can be completed within $O(|\mathcal{P}_{PREF}| \cdot (|\mathcal{P}_{SAS}| + |\mathcal{SPEC}|))$ BDD operations.

$LCONSV(\mathcal{SPEC})$ can be computed with $O(|\mathcal{SPEC}|)$ BDD operations (Proposition 5.8). Therefore both $minLevel(\mathcal{P}_{WRT}, LCONSV(\mathcal{SPEC}))$ and $maxLevel(\mathcal{P}_{WRT}, LCONSV(\mathcal{SPEC}))$ are still within $O(|\mathcal{P}_{PREF}| \cdot (|\mathcal{P}_{SAS}| + |\mathcal{SPEC}|)$ BDD operations.

5.7 Preference vectors

In this multiagent specification system, each specification is in one of the following specification layers: multiagent interactions (IR), specialization effects (SPL), usual operator effects (OP), and static frames (FRM). This structure can be employed to differentiate the preference of the specification combinations in a hierarchical manner. At the structural level, the layout of these specification layers reflects a preference order from an architectural design point of view: inter-agent IR specifications can override local SPL specifications, the local SPL specifications can override the usual OP specifications, and the OP specifications can override the frame specifications FRM. Within each specification layer, user-defined preference levels can be applied to arbitrate the conflicting specifications inside the layer. These user-defined preference levels can come from users' subjective preferences, probabilities, utilities, trusts, and other forms of comparable mental attitude measurement (for example subjective and probabilistic measurements of trust [Tang et al., 2011a]).

Let us first define the concept of level vector on a set of specifications so that we can compare preferences over sets of specifications.

Definition 5.6 (Preference level vector). *Given a set $\mathcal{SPEC}$ of specifications, a vector of measurement levels, called* preference level vector *on $\mathcal{SPEC}$ with respect to the layers can be defined as*

$$levelVector(\mathcal{SPEC}) = \langle level_{ir}, level_{spl}, level_{op}, level_{frm} \rangle$$

where

- $level_{ir}(\mathcal{SPEC}) = minLevel(IR(\mathcal{SPEC}))$,
- $level_{spl}(\mathcal{SPEC}) = minLevel(SPL(\mathcal{SPEC}))$,
- $level_{op}(\mathcal{SPEC}) = minLevel(OP(\mathcal{SPEC}))$, *and*
- $level_{frm}(\mathcal{SPEC}) = minLevel(FRM(\mathcal{SPEC}))$.

$levelVector(\mathcal{SPEC})$ takes the minimum preference of each specification layer as a representative of such a layer. Then we can use the preference vector to compare two sets of specifications with respect to the significance of each layer. Such a comparison can be done through a cascading preference relation $\succeq$ and with *succ* defined recursively on the level vectors.

Definition 5.7 (Preference level vector comparison).
Given two vectors of levels $\mathbf{lv}_i = \langle l_{i,1}, l_{i,2}, ..., l_{i,n}\rangle$ *and* $\mathbf{lv}_j = \langle l_{j,1}, l_{j,2}, ..., l_{j,n}\rangle$, $\mathbf{lv}_i \succ \mathbf{lv}_j$ *meaning that* $\mathbf{lv}_i$ *is preferred to* $\mathbf{lv}_j$, *is defined recursively as*

- $\langle l_{i,n}\rangle \succ \langle l_{j,n}\rangle$ *iff* $l_{i,n} > l_{j,n}$,
- $\langle l_{i,k}, l_{i,k+1}, ..., l_{i,n}\rangle \succ \langle l_{j,k}, l_{j,k+1}, ..., l_{j,n}\rangle$ *iff*
  - $l_{i,k} > l_{j,k}$ *or,*
  - $l_{i,k} = l_{j,k}$ *and* $\langle l_{i,k+1}, ..., l_{i,n}\rangle \succ \langle l_{j,k+1}, ..., l_{j,n}\rangle$.

$\mathbf{lv}_i$ *is at least as preferred as* $\mathbf{lv}_j$, *denoted by* $\mathbf{lv}_i \succeq \mathbf{lv}_j$, *iff* $\mathbf{lv}_i = \mathbf{lv}_j$ *or* $\mathbf{lv}_i \succ \mathbf{lv}_j$.

Based on the comparisons of preference vectors, we are able to define preference relations $\succeq$ and $\succ$ over sets of specifications.

Definition 5.8 (Cascading preference relation). *Given two sets of specifications* $SPEC_i$ *and* $SPEC_j$, $SPEC_i$ *is preferred over* $SPEC_j$, *denoted by* $SPEC_i \succ SPEC_j$, *iff* $level(SPEC_i) \succ level(SPEC_j)$. $\succ$ *is called the* cascading preference relation *over specification sets.* $SPEC_i \succeq SPEC_j$ *iff* $SPEC_i = SPEC_j$ *or* $SPEC_i \succ SPEC_j$.

Proposition 5.12 (Maximally preferred consistent combinations). *Given a set* $\mathcal{SPEC}$ *of specifications, let* $LCombine = LCONS(\mathcal{SPEC})$. $computePreferredCombine(\mathcal{P}_{WRT}, LCombine)$ *(Algorithm 11) computes the maximally preferred combinations (with respect to the vector preference of Definition 5.8) of specifications of* $\mathcal{SPEC}$ *on each configuration of* $\mathcal{P}_{WRT}$.

Proof. $LCombine = LCONS(\mathcal{SPEC})$ collects the set of all possible consistent combinations of $\mathcal{SPEC}$. Starting with $LCombine$, $computePreferredCombine$ goes through each layer of specifications in ascending significance of layers (i.e. the order of IR, SPL, OP and FRM). For each layer, $computePreferredCombine$ first locates a subset of consistent combinations that are involved in a layer that is investigated currently (Line 3) for each configurations of the with-respect-to frame $\mathcal{P}_{WRT}$. Then the minimum preference level of consistent combinations of the current layer is computed as $layerMinCombineV$ (Line 4). For a consistent combination subset $SPEC$ of $\mathcal{SPEC}$ that contains no specification of the current layer, the preference vector $min_{layer}(SPEC)$ will be set to $-\infty$ (see Definition 5.5). This implies that all the specification combinations that contain no specifications of current layer will be less preferred than those that contain a specification of the current layer. By $layerMinCombineV \vee \left(maxLCombine - layerMinCombineV^{\downarrow \mathcal{P}_{WRT}}\right)$ (Line 5), the algorithm then removes from $maxLCombine$ all other combinations that contain no specifications of the current layer for the configurations of $\mathcal{P}_{WRT}$ on which there are combinations that contain specifications of the current layer, and preserves only the combinations that contains specifications of the current layer. As the algorithm goes from more significant layers to less significant layers, the maximally preferred combinations with respect to the vector preference of Definition 5.8 will be returned.

Proposition 5.13 (Complexity of maximally preferred consistent combinations). *Given a set $\mathcal{SPEC}$ of specifications, let $LCombine = LCONS(\mathcal{SPEC})$. $computePreferredCombine(\mathcal{P}_{WRT}$, $LCombine)$ (Algorithm 11) can be computed in $O(|\mathcal{P}_{PREF}|\cdot(|\mathcal{P}_{SAS}|+|\mathcal{SPEC}|)$ BDD operations. Together, $computePreferredCombine(\mathcal{P}_{WRT}, LCONS(\mathcal{SPEC}))$ can also be computed in $O(|\mathcal{P}_{PREF}|\cdot(|\mathcal{P}_{SAS}| + |\mathcal{SPEC}|)$ BDD operations.*

Proof. For the layers IR, SPL, OP, and FRM, 4 iterations are needed. In each iteration the complexity is dominated by $minLevel(\mathcal{P}_{WRT}, layerCombineV)$ which is bounded by $O(|\mathcal{P}_{PREF}| \cdot (|\mathcal{P}_{SAS}| + |\mathcal{SPEC}|)$ BDD operations. As the number of iterations is constant, the complexity of $computePreferredCombine$ is then bounded by $O(|\mathcal{P}_{PREF}| \cdot (|\mathcal{P}_{SAS}| + |\mathcal{SPEC}|)$ BDD operations.

As $LCONS(\mathcal{SPEC})$ can be computed in $O(|\mathcal{SPEC}|)$ BDD operations (see Proposition 5.2), the total complexity of $computePreferredCombine(\mathcal{P}_{WRT}, LCONS(\mathcal{SPEC}))$ is also $O(|\mathcal{P}_{PREF}| \cdot (|\mathcal{P}_{SAS}| + |\mathcal{SPEC}|)$ BDD operations.

Algorithm 11: $computePreferredCombine(\mathcal{P}_{WRT}, LCombine)$: Compute the most preferred combinations

Input : 1) $\mathcal{P}_{WRT}$: The set of with-respect-to frame variables over every configuration of which the maximally preferred combinations are extracted;
2) $LCombine$: A labeled set of consistent combinations of specifications in $\mathcal{SPEC}$

1 $maxLCombine \leftarrow LCombine$;
2 **for** *each layer in the order of* IR, SPL, OP, FRM **do**
  `/* Compute the min-preference level for the` $layer$ `being investigated */`
3 $layerCombineV \leftarrow maxLCombine \wedge LCONSV(layer(\mathcal{SPEC}))$;
  `/* Locate the minimum level combinations in` $layer$ `*/`
4 $layerMinCombineV \leftarrow minLevel(\mathcal{P}_{WRT}, layerCombineV)$;
  `/* Remove the combinations whose` $\mathcal{P}_{WRT}$ `frames have been specified by a combination involving the current` $layer$ `*/`
5 $maxLCombine \leftarrow layerMinCombineV \vee \left(maxLCombine - layerMinCombineV^{\downarrow \mathcal{P}_{WRT}}\right)$;
6 **end**
7 **return** $maxLCombine$;

5.8 Mutual understanding of valid state transitions

In this section, we will compute the set of mutually understood state transitions as the state transitions that can be interpreted from the maximally preferred consistent set of specification combinations for each valid state-action pair. The set of valid state-action pairs is in turn computed as the set of state-action pairs that can be interpreted from the maximally consistent set of specification combinations for each state (Algorithm 12). In Chapter 7, an argumentation-based semantics will be provided to underpin the validity behind the computation of this mutual understanding of state transitions introduced in this section.

The computation of the mutually understood state transitions is composed of two steps:

- Computing the set of valid joint actions for each joint state — the valid state-action space

- Computing the set of valid state transitions for each state-action pair in the state-action space computed in the previous step

The computation of the valid state-action space is performed by Algorithm 12. This algorithm computes the set of maximally consistent combinations of specifications of type SA for each state. Then among the maximally consistent combinations of the SA specifications for each state, the algorithm preserves only the most preferred combinations as the valid actions for such a state. With symbolic model checking techniques, the above computation of the valid most preferred maximally consistent state-action pairs for each state is carried out for all the states simultaneously in Algorithm 12.

Proposition 5.14 (Defeasible computation of state-action space). *Given a list $\mathcal{SPEC}$ of specifications, computeSA($\mathcal{SPEC}$) (Algorithm 12) computes the state-action pairs that can be interpreted out of the most preferred maximally consistent set of combinations of $\mathcal{SPEC}$ for each state, or all possible state-action pairs for the states that have no specifications in $\mathcal{SPEC}$.*

Proof. Algorithm 12 works on a list $SAList$ of state-action specifications (whose $TYPE$ are set to SA). The algorithm computes the maximal combinations with respect to the state frame $\mathcal{P}_S$ in $SAList$, and records the result in $maximalSA$. Among the combinations in $maximalSA$, the most preferred combinations are then computed and recorded in $specifiedSA$. Then the algorithm removes from the universe TRUE the states that have a specified state-action combinations to open up the possibilities for non-specified state-action pairs. The valid state-action space is returned at the end of the algorithm.

With the valid state-action space computed, we can proceed to compute the valid state transitions.

Proposition 5.15 (Defeasible computation of state transition space). *Given a list $\mathcal{SPEC}$ of specifications, computeSAS($\mathcal{SPEC}$) (Algorithm 13) computes the state transitions that can be*

Algorithm 12: $computeSA(\mathcal{SPEC})$: Compute the valid state-action space

Input : $\mathcal{SPEC}$: A list of specifications

1 $SAList \leftarrow SA(\mathcal{SPEC})$;
2 $maximalSA \leftarrow maximal(\mathcal{P}_S, SAList)$;
3 $mostPreferred \leftarrow computePreferredCombine(\mathcal{P}_S, maximalSA)$;
`/* Discard the preference information and project only to the state-action discernment frame */`
4 $specifiedSA \leftarrow mostPreferred^{\downarrow \mathcal{P}_{SA}}$;
5 $SA \leftarrow specifiedSA \vee (\text{TRUE} - specifiedSA^{\downarrow \mathcal{P}_S})$;
6 **return** SA;

interpreted out of the most preferred maximally consistent set of combinations of $\mathcal{SPEC}$ for each valid state-action pair that is computed by $computeSA(\mathcal{SPEC})$.

Proof. In Algorithm 13, the valid state-action space is first computed and recorded into SA. Then it extracts the sublist $SASList$ of state transitions specifications from the input list $\mathcal{SPEC}$. Similar to the state-action space computation, the maximal combinations of $SASList$ with respect to the state-action frame $\mathcal{P}_{SA}$ are computed and recorded in $maximalSAS$. Among the combinations in $maximalSAS$, the most preferred combinations are computed and recorded in $specifiedSAS$. Then the specified state transitions are constrained by the valid state-action space to the valid state transitions.

Proposition 5.16 (Complexity of defeasible state transition space). *Given a list $\mathcal{SPEC}$ of specifications, $computeSAS(\mathcal{SPEC})$ (Algorithm 13) can be computed in $O(|\mathcal{SPEC}|\times(|\mathcal{P}_{SA}|+|\mathcal{P}_{PREF}|))$ BDD operations.*

Proof. The computation is dominated by $computePreferredCombine(\mathcal{P}_{SA}, maximalSAS)$ which can be computed in $O(|\mathcal{SPEC}|\times(|\mathcal{P}_{SA}|+|\mathcal{P}_{PREF}|))$ BDD operations. Therefore, $computeSAS(\mathcal{SPEC})$ (Algorithm 13) can be computed in $O(|\mathcal{SPEC}| \times (|\mathcal{P}_{SA}| + |\mathcal{P}_{PREF}|))$ BDD operations.

Algorithm 13 computes a single agent's state transitions. As for the computation of the joint state transitions of a system of agents, $computeJointSAS(\mathcal{SPEC}, \mathsf{AGS})$ (Algorithm 14) does the

Algorithm 13: $computeSAS(\mathcal{SPEC})$: Compute the valid state transitions

Input : $\mathcal{SPEC}$: A list of specifications

$SA \leftarrow computeSA(\mathcal{SPEC})$;
$SASList \leftarrow SAS(\mathcal{SPEC}) \setminus FRM(\mathcal{SPEC})$;
$maximalSAS \leftarrow maximal(\mathcal{P}_{SA}, SASList)$;
$mostPreferred \leftarrow computePreferredCombine(\mathcal{P}_{SA}, maximalSAS)$;
$specifiedSAS \leftarrow mostPreferred^{\downarrow \mathcal{P}_{SAS}}$;
/* Apply frame specifications */
for *each* $frm_k \in FRM(\mathcal{SPEC})$ **do**
  $framedSAS \leftarrow specifiedSAS \wedge frm_k$;
  /* Apply a frame axiom on a state-action if it is consistent with the transitions */
  if $framedSAS \neq FALSE$ **then**
    $specifiedSAS \leftarrow framedSAS \vee (specifiedSAS - framedSAS^{\downarrow \mathcal{P}_{SA}})$;
  end
end
$SAS \leftarrow specifiedSAS \wedge SA$;
return SAS;

job. Algorithm 13 already finishes most of the computation, what we need in the computation of joint state transition is to enable every agent to act in each state transition. This can be done by having the combinations taking at least one *substantial specification* of each agent. By substantial specification, we mean the specification of multiagent interaction, specialization of single agent actions, and the definitions of operators. The frame specifications are not taken as substantial specifications for the reason that they do not substantially specify how the states are being transformed. The frame specifications only specify that the world is not changed unless some specifications say that it has.

Proposition 5.17 (Defeasible computation of joint state transition space). *$computeJointSAS(\mathcal{SPEC}, \mathsf{AGS})$ (Algorithm 14) computes the valid joint state transitions that can be interpreted out of the most preferred maximally consistent combinations of the specifications of $\mathcal{SPEC}$ for every valid joint state-action pair computed by $computeSA(\mathcal{SPEC})$, and every agent acts in each joint state transition computed.*

Proof. By Proposition 5.15, Line 1 computes the set of valid state transitions from the specifications $\mathcal{SPEC}$. Line 3 guarantees that every action will act in each joint state transition.

Proposition 5.18 (Complexity of defeasible computation of joint state transition space). *Given a list $\mathcal{SPEC}$ of specifications, computeJSAS($\mathcal{SPEC}$, AGS) (Algorithm 14) can be computed in $O(|\mathcal{SPEC}| \times (|\mathcal{P}_{SA}| + |\mathcal{P}_{PREF}|) + |\mathsf{AGS}|)$ BDD operations.*

Proof. *computeJSAS*($\mathcal{SPEC}$, AGS) is bounded by $O(|\mathcal{SPEC}| \times (|\mathcal{P}_{SA}| + |\mathcal{P}_{PREF}|))$ BDD operations. The computation corresponding to enabling every agent to act (Line 3) is bounded by $O(|\mathsf{AGS}|)$ BDD operations. In total, *computeJSAS*($\mathcal{SPEC}$, AGS) can be computed in $O(|\mathcal{SPEC}| \times (|\mathcal{P}_{SA}| + |\mathcal{P}_{PREF}|) + |\mathsf{AGS}|)$ BDD operations.

Algorithm 14: *computeJointSAS*($\mathcal{SPEC}$, AGS): Compute the valid joint state transitions for a system of agents

Input : 1) $\mathcal{SPEC}$: A list of specifications;
2) AGS: A system of agents

1 $SAS \leftarrow computeSAS(\mathcal{SPEC})$;
2 $enableAgents \leftarrow$ FALSE;
3 **for** *each* $\mathsf{Ag}_i \in \mathsf{AGS}$ **do**
4 $\quad enableAgent_i \leftarrow \bigvee_{spec_{i,k} \in \mathcal{SPEC} \text{ and } type(spec_{i,k}) \in \{IR,SPL,OP\}} (l_{spec_{i,k}})$;
5 $\quad enableAgents \leftarrow enableAgents \wedge enableAgent_i$;
6 **end**
7 $JSAS \leftarrow SAS \wedge enableAgents$;
8 **return** $JSAS$;

5.9 Summary

This chapter introduces a defeasible factored action theory for multiagent joint state transitions. The factored information has associated meta-information: types, layers and levels. This meta-information is used to deal with the three fundamental problems in single agent and multiagent action theory: the qualification problem, the ramification and the frame problem. The types and layers of the specifications correspond to an architectural design consideration for the agents

on how mutual understanding of state transitions can be derived, and on how different layers of specifications can override other layers of specifications. The type of a specification can either be a state-action space specification (SA) or a state transition specification (SAS). The usage of specification types has its root in the Markov assumptions of the state-action and the state transition space: 1) the applicability of an action depends only on the state in which it is executed for state-action space ($TYPE = SA$); 2) the effectiveness of a state transition depends only on the state and the action that has been executed in such a state for state transition space ($TYPE = SAS$). The meta-information in $TYPE$ identifies whether the system should carry out defeasible reasoning for each state to draw conclusions about state-action pairs, or carry out defeasible reasoning for each state-action pair to draw conclusions about state transitions. During defeasible reasoning for either type (SA or SAS), the meta-information of the specification layers tells the system which specifications can override which other specifications if they conflict. Within each layer, additional preference levels can be assigned to the specifications to accommodate non-architectural information (e.g. application dependent preference information) to resolve conflicts among these specifications. The defeasible reasoning is implemented using symbolic model checking techniques. Symbolic model checking techniques provide two advantages for us: 1) for each state or each state-action pair, we can simultaneously consider all possible interactions among the specifications as different combinations; 2) we can compute these considerations simultaneously at all layers and levels *simultaneously* for all states or all state-action pairs in a polynomial number of BDD operations. The capability to look at how the effects of different specifications interact in all possible combinations is a key to carry out defeasible reasoning under the concept of joint acceptability [Dung, 1995] (see Chapter 7 for the argumentation theoretical underpinning). The capability to perform the defeasible reasoning for all the states and state-action pairs in a polynomial number of BDD operations shows the acknowledge of borrowing engineering practice from symbolic model checking techniques to deal with the complexity issues in defeasible reasoning and non-monotonic reasoning in general [Dimopoulos et al., 2002].

Based on the defeasible action theory introduced in this chapter, two future directions will

be: 1) the experimental study of the computations for defeasible reasoning using symbolic model checking techniques; 2) the relaxation of the Markov assumptions (e.g. mostly Markovian state transitions but with a library of procedural state-action sequences which can override the Markov assumptions in a number of cases) and the accommodation of statistical and utility information on state transitions (see [Tang et al., 2011b,c] for our preliminary work in this latter direction).

Chapter 6

Multiagent Planning and Plan Coordination

In this chapter the centralized and decentralized planning algorithms will be revised to make use of the defeasible factored action theory introduced in Chapter 5 to plan the agents' joint behavior. Then a coordination mechanism will be introduced to enable the agents to coordinate their joint policy. If there can be more than one joint actions for the same joint state, the mechanism can be extended to enable a power relationship among the agents to resolve conflicting prescription of actions. Furthermore, the maximally consistent combination approach for factored information will be extended to reason about maximally consistent combinations of possibly inconsistent individual goals, and plan for these consistent combinations of goals together. Overall, we will be able to obtain a decentralized planning system with non-monotonic defeasible state transition information and goals, and a multiagent coordination mechanism to execute the planned policy with the capability to resolve conflicting prescription of actions with respect to a power relationship among agents.

6.1 Decentralized planning with factored specifications

First, let's start by modifying the basic ND-planning algorithm (Algorithm 1) into Algorithm 15 so that the agents don't need to share their initial states with each other.

Algorithm 15: $plan(R, I, G, computePreSA)$: Planning

Input : (1) R: The transition relation;
(2) I: The set of initial states;
(3) G: The set of goals states;
(4) $computePreSA$: Either $ComputeWeakPreSA$ or $ComputeStrongPreSA$

$Solved \leftarrow G$;
$UnSolved \leftarrow I - G$;
$FT \leftarrow G$;
$newSA \leftarrow \emptyset$;
while $newSA \neq \emptyset$ *OR* $UnSolved \neq \emptyset$ **do**
  $newSA \leftarrow computePreSA(R, FT)$;
  `/* Remove the state-action pairs whose states have already been solved */`
  $newSA \leftarrow newSA - Solved$;
  $SA \leftarrow SA \vee newSA$;
  $FT \leftarrow \exists_{\mathcal{P}_A} newSA$;
  $Solved \leftarrow Solved \vee FT$;
  $UnSolved \leftarrow UnSolved - Solved$;
end
if $UnSolved = \emptyset$ **then**
  return SA;
else
  return $\emptyset$;
end

Proposition 6.1 (Complexity for unsolved states tracking non-deterministic planning). *Let D be the maximum length of paths in a state transition relation R. $plan(R, I, G, computePreSA)$ (Algorithm 15) can be computed in $O(D \times (|\mathcal{P}_S| + |\mathcal{P}_A|)) < O(2^{|\mathcal{P}_S|} \times (|\mathcal{P}_S| + |\mathcal{P}_A|))$ BDD operations.*

Proof. The computation in each loop is the same as that of Algorithm 1 with one additional BDD operation for computing $UnSolved \leftarrow UnSolved - Solved$. Therefore the complexity analysis

Algorithm 16: $ComputeRelWRT(\mathcal{P}_{WRT}, \mathcal{SPEC}, exp)$: Compute the related frames of a given expression

Input : (1) $\mathcal{P}_{WRT}$: The frame variables on which the result is returned where relevant specifications are specified;
(1) $\mathcal{SPEC}$: A list of specifications;
(2) exp: An expression of states, next states, state-actions, or state transitions;

```
1 relWRT ← FALSE;
2 for each spec_k ∈ SPEC do
3 |   sp_k ← spec_k ∧ exp;
  |   /* Collect the specifications with resulted states overlapping with the
  |      frontier                                                          */
4 |   if sp_k ≠ ∅ then
5 |   |   relWRT ← relWRT ∨ sp_k^{↓P_WRT};
6 |   end
7 end
8 return relWRT;
```

of Algorithm 1 is also valid for Algorithm 15. It can be computed in $O(D \times (|\mathcal{P}_S| + |\mathcal{P}_A|)) < O(2^{|\mathcal{P}_S|} \times (|\mathcal{P}_S| + |\mathcal{P}_A|))$ BDD operations.

Let

$$ComputeRelS(\mathcal{SPEC}, exp) = ComputeRelWRT(\mathcal{P}_S, \mathcal{SPEC}, exp)$$

$$ComputeRelSA(\mathcal{SPEC}, exp) = ComputeRelWRT(\mathcal{P}_{SA}, \mathcal{SPEC}, exp)$$

where $ComputeRelWRT$ is defined in Algorithm 16. $ComputeRelWRT$ computes the frames that intersect with an expression.

With a list of specifications, we can have a decentralized non-deterministic planning algorithm for a multiagent systems AGS as shown in Algorithm 20. The procedure starts with shared joint goals, and then iteratively searches backwards for a synchronized joint plan. During the planning, the agents only communicate

- a restricted version of specifications that are relevant to the joint frontier states that are reachable from the joint goals, and

- the specifications that are relevant to
  - the states that are interfered with the computation of state transitions (see Algorithm 12) that are relevant to the backward reachable states from the goals, and
  - the state-actions that are interfered with the computation of state transitions (see Algorithm 13) that are relevant to the backward reachable states from the goals.

Algorithm 20 starts with all the agents sharing their frame specification. This is because the frame specifications don't depend on the agent's understanding of the external world. The frames actually depend on the agents' ontology, that is their general knowledge about the world. The frame sharing step is essentially a bootstrap step to create a common language to specify the behaviors of the systems. In each planning iteration in Algorithm 20, each agent computes its own relevant states by *ComputeRelS* which invokes Algorithm 16 by setting the output relevant frame to be the state variables $\mathcal{P}_S$, and computes its own relevant state-actions by *ComputeRelSA*. This in turn invokes Algorithm 16 by setting the output relevant frame to be the state variables $\mathcal{P}_{SA}$. Using the relevant states and state-actions, the agents can locate the lists of relevant specifications, and exchange these relevant specifications with the other agents. After receiving all the relevant specifications, the agents can compute the state transitions relevant to the given frontier by themselves.

Proposition 6.2 (Decentralized defeasible planning). *When every agent executes Algorithm 20, then*

- *if there is a valid joint plan, all the agents will compute the same plan (completeness), and*
- *the plan computed is a valid plan for all agents (soundness).*

Proof. The algorithm starts by synchronizing individual agents' goals and combining these individual goals into a joint goal. After the synchronization, every agent will hold the same joint goal and will plan to achieve this joint goal.

Thereafter, the agents follow the same non-deterministic planning algorithms to plan backward from the joint goal. This process is begun by setting the joint goal to be the initial frontier for every

agent. During the planning procedure, each agent computes the relevant states, state-actions and their specifications for the frontier, and then send the relevant specifications to each other. With the relevant specifications communicated, all the agents will combine the same set of specifications into relevant state transitions with Algorithm 13. With the same algorithm computing the pre-images, all the agents will obtain the same next frontier and the same policy decisions.

Afterward, all agents will compute their relevant specifications using the same frontier, and share these specifications with each other.

The above process continues until either no new policy segments can be obtained or the agent's initial states have all been covered. Then the agents communicate their planning status. If all the agents succeed in their planning, then the whole agent system succeeds in its planning and the joint policy is returned. Furthermore, the joint policy each agent computes individually will all be same.

Lemma 6.1 (Complexity of computing relevant *WRT* frames).
The complexity of ComputeRelWRT$(\mathcal{P}_{WRT}, \mathcal{SPEC}, exp)$ *(Algorithm 16) is bounded by* $O(|\mathcal{SPEC}| \cdot (|\mathcal{P}_{SAS}| - |\mathcal{P}_{WRT}|))$ *BDD operations.*

Proof. The major for loop goes through every specification in $\mathcal{SPEC}$. It results in $|\mathcal{SPEC}|$ loops. In each loop, the complexity is dominated by the sequence of existential quantifications for $sp_k^{\downarrow \mathcal{P}_{WRT}}$. The number of existential quantifications is bounded by $|\mathcal{P}_{SAS}| - |\mathcal{P}_{WRT}|$. In total, *ComputeRelWRT*$(\mathcal{P}_{WRT}, \mathcal{SPEC}, exp)$ can be computed in $O(|\mathcal{SPEC}| \cdot (|\mathcal{P}_{SAS}| - |\mathcal{P}_{WRT}|))$ BDD operations.

Immediately following from Lemma 6.1, *ComputeRelS*$(\mathcal{SPEC}, exp)$ = *ComputeRelWRT*$(\mathcal{P}_S, \mathcal{SPEC}, exp)$ can be computed in $O(|\mathcal{SPEC}| \cdot |\mathcal{P}_{SA}|)$ BDD operations. *ComputeRelSA*$(\mathcal{SPEC}, exp)$ = *ComputeRelWRT*$(\mathcal{P}_{SA}, \mathcal{SPEC}, exp)$ can be computed in $O(|\mathcal{SPEC}| \cdot |\mathcal{P}_S|)$.

Lemma 6.2 (Complexity for computing relevant specifications).
The complexity of ComputeRelList$(\mathcal{SPEC}, exp)$ *(Algorithm 17) is bounded by* $O(|\mathcal{SPEC}|)$ *BDD operations.*

Algorithm 17: $ComputeRelList(\mathcal{SPEC}, exp)$: Compute the specifications that intersect with an expression

Input : (1) $\mathcal{SPEC}$: A set of specifications;
(2) exp: An expression about states, next states, state-actions, next states, or state transitions about what is relevant;

```
1 relSpecList ← ∅;
2 for each spec_k ∈ SPEC do
3     if spec_k ∧ exp ≠ ∅ then
          /* Collect the specifications with current states overlapping with the
             states that can possibly leading to the frontier                  */
4         relSpecList ← relSpecList ∪ {spec_k ∧ exp};
5     end
6 end
7 return relSpecList;
```

Proof. The major for loop goes through every specification in $\mathcal{SPEC}$. Within each loop, there are a constant number of BDD operations. Therefore, $ComputeRelList(\mathcal{SPEC}, exp)$ can be computed in $O(|\mathcal{SPEC}|)$ BDD operations.

Algorithm 18: $obtainRelS\_RelSA(\mathsf{Ag}_i, \mathcal{SPEC}_i, \mathsf{AGS}, FT)$: Obtain the states and state-action pairs that are related to the frontier FT

Input : (1) Ag_i: The agent who calls the planning algorithm; (2) $\mathcal{SPEC}_i$: The list of specifications Ag_i owns; (3) AGS: The multiagent system in which agent Ag_i cooperates with; (4) FT: The frontier set of states;

```
1 relS_i ← ComputeRelS(SA(SPEC_i), FT[P_S/P_S'])
2 relSA_i ← ComputeRelSA(SAS(SPEC_i), FT[P_S/P_S'])
3 Send relS_i, relSA_i to every other agents Ag_j ∈ AGS (j ≠ i);
4 Wait for relS_j, relSA_j from the other agents Ag_j ∈ AGS (j ≠ i);
5 relS ← ⋁_{Ag_k ∈ AGS} relS_k;
6 relSA ← ⋁_{Ag_k ∈ AGS} relSA_k;
7 return ⟨relS, relSA⟩;
```

Lemma 6.3 (Complexity of obtaining relevant states and state-action pairs). *The complexity of* $obtainRelS\_RelSA(\mathsf{Ag}_i, \mathcal{SPEC}_i, \mathsf{AGS}, FT)$ *(Algorithm 18) is bounded by* $O(|\mathcal{SPEC}|)$ *BDD operations,* $O(|\mathsf{AGS}|)$ *specification list maintenance operations, and* $O(|\mathsf{AGS}|)$ *inter-agent communications where each communication is bounded by* $O(|\mathcal{SPEC}|)$ *messages.*

Proof. Immediate by counting the number of operations in Algorithm 18.

Algorithm 19: *obtainRelSpec*($\mathsf{Ag}_i, \mathcal{SPEC}_i, \mathsf{AGS}, relS, relSA$): Obtain the specifications that are related to the states $relS$ and state-actions $relSA$

Input : (1) Ag_i: The agent who calls the planning algorithm; (2) $\mathcal{SPEC}_i$: The list of specifications Ag_i owns; (3) AGS: The multiagent system in which agent Ag_i cooperates with; (4) $relS$: The set of states to specify for; (5) $relSA$: The set of state-actions to specify for;

1 $relSAList_i \leftarrow ComputeRelList(SA(\mathcal{SPEC}_i), relS)$;
2 $relSASList_i \leftarrow ComputeRelList(SAS(\mathcal{SPEC}_i) \setminus frmList, relSA)$;
3 $relSpecList_i \leftarrow relSAList_i \cup relSASList_i$;
4 Send $relSpecList_i$ to every other agents $\mathsf{Ag}_j \in \mathsf{AGS}$ $(j \neq i)$;
5 Wait for $relSpecList_j$ from the other agents $\mathsf{Ag}_j \in \mathsf{AGS}$ $(j \neq i)$;
6 $relSpecList \leftarrow \cup_{\mathsf{Ag}_j \in \mathsf{AGS}} relSpecList_j$;
7 **return** $relSpecList$;

Lemma 6.4 (Complexity of obtaining relevant specifications).

The complexity of obtainRelSpec($\mathsf{Ag}_i, \mathcal{SPEC}_i, \mathsf{AGS}, relS, relSA$) *(Algorithm 19) is bounded by* $O(|\mathcal{SPEC}|)$ *BDD operations,* $O(|\mathsf{AGS}|)$ *specification list maintenance operations, and* $O(|\mathsf{AGS}|)$ *inter-agent communications where each communication is bounded by* $O(|\mathcal{SPEC}|)$ *messages.*

Proof. Immediate by counting the number of operations in Algorithm 19.

Proposition 6.3 (Complexity of decentralized planning on defeasible factored specifications).

Let D *be the maximum length of paths in the joint state transition relation* $\mathcal{R}$ *that can be computed by Algorithm 14:*

$$computeJointSAS(\mathcal{SPEC}, \mathsf{AGS}).$$

For an agent Ag_i, *DecSpecPlan*($\mathsf{Ag}_i, \mathcal{SPEC}_i, \mathsf{AGS}, I, JG, computePreSA$) *(Algorithm 20) can be computed with* $O(D \times (|\mathcal{SPEC}| \cdot (|\mathcal{P}_{SA}| + |\mathcal{P}_{PREF}|) + |\mathsf{AGS}|))$ *BDD operations.*

In addition, the algorithm requires $O(D \cdot |\mathsf{AGS}|)$ *specification list maintenance operations, and*

Algorithm 20: $DecSpecPlan(\mathsf{Ag}_i, \mathcal{SPEC}_i, \mathsf{AGS}, I, JG, computePreSA)$: Decentralized Planning with factored specifications

Input : (1) Ag_i: The agent who calls the planning algorithm; (2) $\mathcal{SPEC}_i$: The list of specifications Ag_i owns; (3) AGS: The multiagent system in which agent Ag_i cooperates with; (4) I: The set of initial states of the local concerns; (5) JG: The set of shared goal states; (6) $computePreSA$: Either $ComputeWeakPreSA$ or $ComputeStrongPreSA$

$frmList_i \leftarrow FRM(\mathcal{SPEC}_i)$, and share $frmList_i$ to all other agents;
$frmList \leftarrow \cup_{\mathsf{Ag}_j \in \mathsf{AGS}} frmList_j$;
$Solved \leftarrow JG$;
$UnSolved \leftarrow I - JG$;
$FT \leftarrow JG$;
$newSA \leftarrow \emptyset$;
while $newSA \neq \emptyset$ *OR* $UnSolved \neq \emptyset$ **do**
  /* Exchange the relevant state, state-action information with other agents */
  $\langle relS, relSA \rangle \leftarrow obtainRelS\_RelSA(\mathsf{Ag}_i, \mathcal{SPEC}_i, \mathsf{AGS}, FT)$;
  /* Exchange the relevant specifications based on the information of related states, state-actions with other agents */
  $relSpecList \leftarrow obtainRelSpec(\mathsf{Ag}_i, \mathcal{SPEC}_i, \mathsf{AGS}, relS, relSA)$;
  /* Compute the joint state transitions with the specifications gathered from other agents */
  $SAS \leftarrow computeJointSAS(relSpecList \cup frmList, \mathsf{AGS})$;
  /* Continue with a backward planning procedure */
  $newSA \leftarrow computePreSA(SAS, FT)$;
  /* Remove the state-action pairs whose states have already been solved */
  $newSA \leftarrow newSA - Solved$;
  $SA \leftarrow SA \vee newSA$;
  $FT \leftarrow \exists_{\mathcal{P}_A} newSA$;
  $Solved \leftarrow Solved \vee FT$;
  $UnSolved \leftarrow UnSolved - Solved$;
end
if $UnSolved = \emptyset$ **then**
  return SA;
else
  return $\emptyset$;
end

$O(D \cdot |\mathsf{AGS}|)$ *inter-agent communications where each communication is bounded by* $O(|\mathcal{SPEC}|)$ *messages.*

For the multiagent system as a whole, decentralized planning with defeasible factor information can be computed with $O(|\mathsf{AGS}| \times D \times (|\mathcal{SPEC}| \cdot (|\mathcal{P}_{SA}| + |\mathcal{P}_{PREF}|) + |\mathsf{AGS}|))$ *BDD operations. During the process, the agent system require* $O(D \cdot |\mathsf{AGS}|^2)$ *specification list maintenance operations, and* $O(D \cdot |\mathsf{AGS}|^2)$ *inter-agent communications where each communication is bounded by* $O(|\mathcal{SPEC}|)$ *messages.*

Proof. The computation in each loop is the same as that of Algorithm 15 with additional computation of the joint state transitions. As in Algorithm 15, the number of loops is bounded by the maximum length D of paths of $\mathcal{R}$. Within each loop the computation is dominated by the computation of the joint state transitions which is bounded by $O(|\mathcal{SPEC}| \times (|\mathcal{P}_{SA}| + |\mathcal{P}_{PREF}|) + |\mathsf{AGS}|)$ BDD operations. In total, $DecSpecPlan(\mathsf{Ag}_i, \mathcal{SPEC}_i, \mathsf{AGS}, I, JG, computePreSA)$ can be computed with $O(D \times (|\mathcal{SPEC}| \cdot (|\mathcal{P}_{SA}| + |\mathcal{P}_{PREF}|)) + |\mathsf{AGS}|)$ BDD operations.

In addition, within each loop the computation of $obtainRelS\_RelSA(\mathsf{Ag}_i, \mathcal{SPEC}_i, \mathsf{AGS}, FT)$ (Algorithm 18) and $obtainRelSpec(\mathsf{Ag}_i, \mathcal{SPEC}_i, \mathsf{AGS}, relS, relSA)$ (Algorithm 19) requires $O(|\mathsf{AGS}|)$ specification list maintenance operations, and $O(|\mathsf{AGS}|)$ inter-agent communications where each communication is bounded by $O(|\mathcal{SPEC}|)$ messages. In total, the algorithm requires $O(D \cdot |\mathsf{AGS}|)$ specification list maintenance operations, and $O(D \cdot |\mathsf{AGS}|)$ inter-agent communications where each communication is bounded by $O(|\mathcal{SPEC}|)$ messages.

For the multiagent system as a whole, since very agent executes the same algorithm, the number of operations needed is then multiplied by $O(|\mathsf{AGS}|)$.

The communication complexity of Algorithm 18, Algorithm 19 and overall planning (Algorithm 20) is tricky to analyze because the messages will be sent in the form of BDDs, and the BDD data structure is a hash-table based data structure. Shared BDDs are cached within agents. If an additional hash table is used to enable the agents to cache each other's BDDs, then a fair amount of communication on BDDs will be cached. Further experiment study is needed to fully analyz-

ing the behaviors of the inter-agent BDD communications. The complexity analysis will be even more complicated if practical BDD techniques, such as dynamic variable re-ordering and searching heuristics, are used. With the experimental analysis, I expect that the algorithms proposed in this dissertation will need more work to fill in the gap on applying the practical symbolic model checking techniques. As this dissertation focuses on the theoretical correctness of the proposed inter-agent planning algorithms, I will leave the experimental communication complexity analysis and the application of practical symbolic model checking techniques for future work.

6.2 Coordination

In previous chapters, we have discussed how to compute joint policies for a system agents with various inputs (either consistent specifications and inconsistent specifications). To ensure that agents execute the joint policy correctly, we need to equip that with a coordination mechanism. We will discuss such a coordination mechanism in this section.

Let each agent Ag_i have a *cognitive state* that identifies what it should do next on incoming perceptions. This concept of cognitive state roughly mirrors the concept of intentions in the BDI model [Rao and Georgeff, 1995]. Let

$$cogs_i \subseteq 2^{\mathcal{P}_{SA}}$$

where $dom(cogs_i) = \mathcal{P}_{SA}$. A cognitive state can be updated by agent Ag_i's perception

$$cogs_{i,k+1} = cogs_{i,k} \wedge PERCEPT_i$$

where

$$PERCEPT \in 2^{\mathcal{P}_{i,S}}$$

with $dom(PERCEPT) = \mathcal{P}_{i,S}$. For an agent Ag_i who holds a policy π_i (see definition in Section 3.2), its cognitive state at any time step is typically a subset of π, namely $cogs_{i,k} \subseteq \pi_i$.

6.2.1 Coordinate deterministic policy

Using the concepts of cognitive state, agents with a deterministic joint policy π (see the definition of deterministic policy in Section 3.2), agents in the agent system AGS can coordinate their actions by using Algorithm 21. In the algorithm, each agent perceives the environment to obtain a perception $PERCEPT_i$, and then use this perception and the given joint policy to form a cognitive state:

$$cogs_i = \pi \wedge PERCEPT_i.$$

Then each agent tries to determine the other agents' needs concerning its local information. In other words, each agent tries to determine, given the next prescription of the joint policy, what information that it uniquely knows is required by other agents in order for them to decide what to do next. Thus agent Ag_i uses its local perception $PERCEPT_i$ to form a cognitive state of other agent Ag_j from agent Ag_i's point of view:

$$cogs_{\mathsf{Ag}_j|\mathsf{Ag}_i} = (\pi \wedge PERCEPT_i)^{\downarrow \mathsf{Ag}_j}.$$

Then $cogs_{\mathsf{Ag}_j|\mathsf{Ag}_i}$ is used to compute the set of possible actions of agent Ag_j from the agent Ag_i's point of view by:

$$nextA_j = cogs^{\downarrow \mathcal{P}_{j,A}}_{\mathsf{Ag}_j|\mathsf{Ag}_i}$$

Since all agents are acting in the same environment, if agent Ag_i perceives $PERCEPT_i$ then the set of all possible perceptions another agent Ag_j has is

$$PERCEPT_{\mathsf{Ag}_j|\mathsf{Ag}_i} = (\pi \wedge PERCEPT_i)^{\downarrow \mathcal{P}_{j,S}}.$$

In other words, from the point of view of agent Ag_i, the set of possible state-actions that Ag_j can take is then

$$\begin{aligned} nextA_{\mathsf{Ag}_j|\mathsf{Ag}_i} & \\ &= (\pi \wedge PERCEPT_i)^{\downarrow \mathcal{P}_{j,S}} \wedge \pi \\ &= (\pi \wedge PERCEPT_i)^{\downarrow \mathcal{P}_{j,SA}} \\ &= (\pi \wedge PERCEPT_i)^{\downarrow \mathsf{Ag}_j}. \end{aligned}$$

This justifies Line 4 of Algorithm 21. After the computation of another agent Ag_j's possible actions from agent Ag_i's information, if there is only one action available to agent Ag_j, then it means that agent Ag_j doesn't need agent Ag_i's information to make a decision; otherwise, agent Ag_i sends its

perception to agent Ag_j to help him make the decision about what to do. If the given joint policy is a deterministic policy (i.e. there is only one joint action for the multiagent system as whole in any given joint state), after all the agents exchange their perceptions with every other agent, all the agents will have enough information to compute a unique action for themselves to execute with respect to the given joint policy. Algorithm 21 is a naive systematic mechanism to exhaust all possible information exchanges among the agents until every agent can compute its own unique action. Its communication complexity is bounded by all agent sharing their perceptions with each other.

Algorithm 21: *CoordinateJointPolicy*($\mathsf{Ag}_i, \mathsf{AGS}, \pi$): Deterministic Policy Coordination of agent Ag_i

Input : (1) Ag_i: The agent who calls the procedure;
(2) AGS: The known agent system;
(3) π: A shared joint policy.

Perceive the environment and obtain the perception $PERCEPT_i$;
$cogs_i \leftarrow \pi \wedge PERCEPT_i$;
for *each* $\mathsf{Ag}_j \in \mathsf{AGS}$ *such that* $j \neq i$ **do**
  $nextSA_j \leftarrow (\pi \wedge PERCEPT_i)^{\downarrow \mathsf{Ag}_j}$;
  if $nextSA_j$ *contains only one entry* **then**
    Continue ;
  else
    Send $PERCEPT_i$ to agent Ag_j;
  end
end
Initialize $PERCEPT_j \leftarrow TRUE$ for every other agent Ag_j $(j \neq i)$;
repeat
  $nextAction \leftarrow cogs_i^{\downarrow \mathcal{P}_{i,A}}$;
  if *Receiving a coordination messages* $PERCEPT_j$ *from an agent* Ag_j $(j \neq i)$ **then**
    $nextAction \leftarrow nextActoin \wedge PERCEPT_j$;
  end
until *nextAction contains only one action for* Ag_i;
return *nextAction*;

Proposition 6.4 (Complexity of coordination).

CoordinateJointPolicy($\mathsf{Ag}_i, \mathsf{AGS}, \pi$) *(Algorithm 21) is bounded by* $O(|\mathsf{AGS}|)$ *BDD operations. The*

number of messages communicated of each agent is $O(|\mathsf{AGS}|)$ *and* $O(|\mathsf{AGS}|^2)$ *in the multiagent system.*

Proof. The two loops in $CoordinateJointPolicy(\mathsf{Ag}_i, \mathsf{AGS}, \pi)$ can be completed in $|\mathsf{AGS}|$ rounds. In each round of both loops, the number of BDD operations is constant. In total, $CoordinateJointPolicy$ is bounded by $O(|\mathsf{AGS}|)$ BDD operations. The messages sent are the perceptions of individual agents. The number of perceptions at any given moment is the number of agents $|\mathsf{AGS}|$, and each message needs to be sent to the other $|\mathsf{AGS}| - 1$ agents in the worst case.

6.2.2 Coordinate non-deterministic policy

If the given joint policy π is not a deterministic policy, then each agent will need to make an additional decision on which action to take. This decision is modeled by an application dependent function *decision* (used in Line 23 of Algorithm 23). As the agents make their decisions locally, these decisions may be inconsistent with each other. In this case, we can use a conflict resolving mechanism with respect to the authority level of the agents with the following constructs and with Algorithm 23 and Algorithm 22. The non-deterministic policy coordination Algorithm 23 is based on deterministic policy coordination Algorithm 21. Assume each agent Ag_i has his own decision function

$$decide_i.$$

After exchanging their perceptions, the agents then compute their local decisions on which actions to take with their own decision function $decide_i$ $(i = 1, ..., N)$ to form a decision list $decisionList = \{decision_i\}$. After the decisions are made, the agents exchange their decisions and every agent forms the same list *decisionList* of all decisions made. With the exchanged decisions, each agent Ag_i construct the set of maximal consistent decisions with the function $maximal(\emptyset, DecisionList)$ (see Section 5.5).

Because every agent computes the maximal sets of consistent decisions based on the same set of decisions gathered from other agents, each agent then computes the same set of maximal sets of

consistent decisions. Among these maximal sets of consistent decisions, Algorithm 22 is then used to select the specific set of decisions that are composed by a set of agents with the largest least command level. The *command level*s are real numbers that are ascribed to the agents, denoted by

$$\mathsf{CommandLevel}(\mathsf{Ag}_i)$$

to reflect their ability to order each other to do things. For any two agents Ag_i and Ag_j, if

$$\mathsf{CommandLevel}(\mathsf{Ag}_i) > \mathsf{CommandLevel}(\mathsf{Ag}_j)$$

then Ag_i has higher authority than agent Ag_j. In the case of conflicting decision, for example Ag_i has decision $\mathsf{decision}_i$ and Ag_j has decision $\mathsf{decision}_j$ but $\mathsf{decision}_i \wedge \mathsf{decision}_j = \text{FALSE}$, then Ag_i's decision $\mathsf{decision}_i$ will override $\mathsf{decision_j}$ of Ag_i. We assume that these command levels are shared by all the agents before the coordination mechanism begins. To simplify the mechanism, we also assume that the ordering over the command levels is strict, namely there are no equal command levels. The strict order assumption is to guarantee that every agent scan the agents' decisions in the same order for if there are agents with equal command levels then different agents might choose to scan their decisions in different order. This in turns guarantee that every agent will select the same decision at Line 8 of Algorithm 22.

Proposition 6.5 (Complexity of computing joint decision).
computeJointDecision(*decisionList*) *(Algorithm 22) can be computed in* $O(|decisionList|+|\mathcal{P}_{SAS}|+|\mathsf{AGS}| \times |\mathcal{P}_S|)$ *BDD operations.*

Proof. $maximal(\emptyset, decisionList)$ can be computed in $O(|decisionList| + |\mathcal{P}_{SAS}|)$ BDD operations (see Proposition 5.7). The two loops can both be completed within $|\mathsf{AGS}|$ rounds. Within each loop, the computation complexity is dominated by computing $ordered^{\downarrow \mathcal{P}_A}$ which can be computed by $O(|\mathcal{P}_S|)$ existential quantifications. In total, *computeJointDecision*(*decisionList*) can be computed in $O(|decisionList| + |\mathcal{P}_{SAS}| + |\mathsf{AGS}| \times |\mathcal{P}_S|)$ BDD operations.

Algorithm 22: *computeJointDecision*(*decisionList*): Compute the most preferred joint maximal consistent decisions

Input : *decisionList*: A list of decisions made by each agent.

`/* Compute the sets of maximal consistent joint decisions (see Section 5.5) */`

$LDecision \leftarrow maximal(\emptyset, decisionList)$;

`/* Label the decision combination with the agent of the least commanding level */`

for *each* $\mathsf{Ag}_j \in \mathsf{AGS}$ *($j \neq i$) in an ascending order in the agents' commanding level* **do**

  `/* Label each joint decision with the agent of the least authority */`

  $ordered \leftarrow LDecision \wedge l_{\mathsf{Ag}_j}$;

  `/* Remove all the labels without` l_{Ag_j} `from` $LDecision$ `for the set of joint actions */`

  $LDecision \leftarrow ordered \vee \left(LDecision - ordered^{\downarrow \mathcal{P}_A}\right)$;

end

$selectedDecision \leftarrow \emptyset$;

for *each* $\mathsf{Ag}_j \in \mathsf{AGS}$ *in a descending order in the agents' commanding level* **do**

  `/* Select the joint action with the maximum least commanding level */`

  $selected \leftarrow LDecision \wedge l_{\mathsf{Ag}_j}$; **if** $selected \neq FALSE$ **then**

    return *selected*;

  end

end

return $\emptyset$;

Proposition 6.6 (Non-deterministic policy coordination).
$CoordinateNDJointPolicy(\mathsf{Ag}_i, \mathsf{AGS}, \pi)$ (Algorithm 23) can be computed in $O(|\mathcal{P}_{SAS}| + |\mathsf{AGS}| \times |\mathcal{P}_S|)$ BDD operations. The number of messages communicated of each agent is in $O(|\mathsf{AGS}|)$, and $O(|\mathsf{AGS}|^2)$ for the whole multiagent system.

Proof. The computation is dominated by $computeJointDecision(decisionList)$ with $|decisionList| = |\mathsf{AGS}|$. Then by Proposition 6.5 $computeJointDecision(decisionList)$ can be computed in $O(|\mathcal{P}_{SAS}|+ |\mathsf{AGS}| \times |\mathcal{P}_S|)$ BDD operations. The rest of the computation is of the same order complexity as $CoordinateJointPolicy(\mathsf{Ag}_i, \mathsf{AGS}, \pi)$ (Algorithm 21) which is bounded by $O(|\mathsf{AGS}|)$ BDD operations. In total, $CoordinateNDJointPolicy(\mathsf{Ag}_i, \mathsf{AGS}, \pi)$ can be computed in $O(|\mathcal{P}_{SAS}| + |\mathsf{AGS}| \times |\mathcal{P}_S|)$ BDD operations.

The number of messages communicated of each agent is $2|\mathsf{AGS}|$ — at most one perception message and one decision message for each agent. In total the number of messages communicated of an agent is bounded by $O(|\mathsf{AGS}|)$, and $O(|\mathsf{AGS}|^2)$ for the multiagent system as whole.

Algorithm 23: *CoordinateNDJointPolicy*($\mathsf{Ag}_i, \mathsf{AGS}, \pi$): Non-Deterministic Policy Coordination of agent Ag_i

Input : (1) Ag_i: The agent who calls the procedure;
(2) AGS: The known agent system;
(3) π: The joint non-deterministic policy to coordinate.

for *each time step* **do**
  Perceive the environment and obtain the perception $PERCEPT_i$;
  $cogs_i \leftarrow \pi \wedge PERCEPT_i$;
  for $\mathsf{Ag}_j \in \mathsf{AGS}$ *such that* $j \neq i$ **do**
    $nextSA_j \leftarrow (\pi \wedge PERCEPT_i)^{\downarrow \mathsf{Ag}_j}$;
    if $nextSA_j$ *contains only one entry* **then**
      Continue ;
    else
      Send $PERCEPT_i$ to agent Ag_j;
    end
  end
  if $cogs_i^{\downarrow \mathcal{P}_A}$ *contains only one action* **then**
    $nextAction \leftarrow cogs_i^{\downarrow \mathcal{P}_A}$;
    Ignore all incoming coordination messages;
  else
    Initialize $PERCEPT_j \leftarrow TRUE$;
    Wait for the coordination messages $PERCEPT_j$ from Ag_j $(j \neq i)$;
    $nextAction \leftarrow cogs_i \wedge \bigwedge_{j \neq i} PERCEPT_j$
  end
  if *nextAction contains only one joint action* **then**
    $nextAction_i \leftarrow nextAction^{\downarrow \mathcal{P}_{i,A}}$;
  else
    $decicision_i \leftarrow decide_i(nextAction)$;
    Send $decision_i$ to every other agent Ag_j $(j \neq i)$;
    Wait for the coordination messages $decision_j$ from Ag_j $(j \neq i)$;
    $decisionList \leftarrow \cup_{\mathsf{Ag}_j \in \mathsf{AGS}} \{decision_j\}$;
    $jointDecision \leftarrow computeJointDecision(decisionList)$;
    $nextAction_i \leftarrow jointDecision^{\downarrow \mathcal{P}_{i,A}}$;
  end
  Execute $nextAction_i$;
end

6.3 Inconsistent goals

6.3.1 Goal specifications

Similar to the factored approach to state transition specifications introduced in Section 5.2, we can specify the goals in a factored way:

$$GSPEC_i = \{G_{i,k}\}$$

where $GSPEC_i$ is the goal specification of agent Ag_i and $G_{i,k}$ is the kth specification of agent Ag_i. Each specification $G_{i,k}$ has an associated label $l_{G_{i,k}}$. We can collect the non empty set of consistent goal specifications about the joint goal from $GSPEC = \cup_{\mathsf{Ag}_i \in \mathsf{AGS}} GSPEC_i$ by

$$LCONS(GSPEC).$$

Then we can maximally combine the goal specifications by (see Section 5.5)

$$maximal(\emptyset, GSPEC).$$

Given the set of goal specifications of all the agents $GSPEC = \{G_{i,k}\}$, we can then compute the maximal consistent sets of goals by:

$$maximal_G(LG, GSPEC) = maximal_S(LG, GSPEC).$$

Again as in Section 6.2.2 on coordinating inconsistent decisions, we can select the max-min preferred combinations of goals of all agents. As a result, we have the following procedure to compute the set of max-min preferred goals for a list of goal specifications $GSPEC$:

$$LG = LCONS(GSPEC)$$
$$maxMinLG = computeJointGoals(LG) \text{ (Algorithm 24) .}$$

Algorithm 24: *computeJointGoals*(LG): Compute the most preferred maximally consistent combinations of goals

Input : (1) LG: A non-empty consistent combinations of goal specifications.

```
   /* Compute the sets of maximal consistent joint goals                    */
```
1 $maxMinLG \leftarrow LG$;
2 $orderedGoals \leftarrow \emptyset$;
3 **for** *each* $G_{i,k} \in GSPEC$ *in an ascending order in preference* **do**
```
      /* Label each joint goals with the least authority                    */
```
4 $ordered \leftarrow maxMinLG \wedge l_{G_{i,k}}$;
```
      /* Remove from maxMinLG the goals that are already labeled with the
         current preference                                                 */
```
5 $maxMinLG \leftarrow maxMinLG - ordered^{\downarrow \mathcal{P}_S}$;
6 **end**
7 $selectedGoals \leftarrow \emptyset$;
8 **for** *each* $G_{i,k} \in GSPEC$ *in a descending order in preference* **do**
```
      /* Select the joint goals with the maximum least authority            */
```
9 $selected \leftarrow maxMinLG \wedge l_{G_{i,k}}$;
10 **if** *selected is not empty* **then**
11 **return** *selected*;
12 **end**
13 **end**
14 **return** $\emptyset$;

Proposition 6.7 (Complexity of computing joint goals). *Given a set $GSPEC$ of goal specifications, computeJointGoals($LCONS(GSPEC)$) (Algorithm 24) can be computed in $O(|GSPEC|^2)$ number of BDD operations.*

Proof. $LCONS(GSPEC)$ can be computed with $O(|GSPEC|)$ n BDD operations (see Proposition 5.2). The two loops in Algorithm 24 can both be completed in $|GSPEC|$ rounds. The computation of each round is dominated by computing $ordered^{\downarrow \mathcal{P}_S}$ which is bounded by $O(|\mathcal{P}_{L,G}|) = O(|GSPEC|)$ BDD operations. In total, the computation of *computeJointGoals* can be computed in $O(|GSPEC|^2)$ BDD operations.

6.3.2 Simultaneously planning for maximally consistent sets of multiple goals

When we do the planning, we can have a planning algorithm (Algorithm 25) keep track of which goal specifications each state-action pair in the policy are achieving by keeping the goal labeling variables.

Proposition 6.8 (Soundness and completeness of multi-goals planning). *Algorithm 25 is sound and complete.*

- *Soundness: If a policy is returned for consistent sets of goal specifications, then such the labeled subsets of the policy guarantee the achievement of each consistent set of goal specifications with the same labels in the policy with respect to the strong or weak planning solution concepts.*
- *Completeness: If there is a policy that can achieve the consistent sets of goal specifications with respect to the strong or weak planning concepts, then the algorithm will output a policy for each consistent set of goal specifications with the goal specification labels.*

Proof. Given a set $\mathcal{GSPEC}$ of goal specifications, Algorithm 25 starts with the labeled non-empty consistent sets of goal specifications $LCONS(\mathcal{GSPEC})$. During planning, the algorithm keeps track of which goal specifications have been achieved in the *Solved*. At the same time, every state in the set of initial states is labeled with all possible goal specifications by

$$LI \leftarrow I \wedge LG^{\mathcal{P}_{L,\mathcal{G}}}.$$

This means that from each state in I, we need to keep track of which goals will need to be achieved. Therefore, the unsolved states are initialized by

$$UnSolved \leftarrow LI - LG$$

meaning among the states in LI, the states that are already in LG are removed. In $UnSolved$, the unsolved goal specifications for each initial state are tracked using the goal label variables. There-

after, the algorithm goes ahead with an ordinary backward chaining planning algorithm using the weak pre-image function (see the definition of $computeWeakPreSA(R, FT)$ in Section 3.5) or strong pre-image function ($computeWeakPreSA(R, FT)$ in Section 3.5). Both pre-image computation functions can preserve the specification labels while computing the pre-image of the frontier states FT, because the computation only involve in QBF quantifications over the state and action variables, the label variables are untouched. In this way, the state-action pairs computed by the pre-image functions can label which goal specifications these state-action pairs are working towards in $newSA$ and in turn in SA. After each pre-image computation step, the solved and unsolved states with labels are updated by

$$Solved \leftarrow Solved \vee \exists_{\mathcal{P}_A} newSA$$

$$UnSolved \leftarrow UnSolved - Solved.$$

As there are only a finite number of states and a finite number of goal specifications, there will be a limited number of new frontier states to explore, and a limited number of new state-action pairs. The main loop of the algorithm is guaranteed to stop with no new frontier state related state-action pairs or no unsolved goals. After the main loop finishes, the algorithm computes the set of goal specification labels that are not achieved by

$$UnSolvedL \leftarrow \exists_{\mathcal{P}_S} UnSolved.$$

And then the algorithm removes those goals that are not achieved from the policy by:

$$SA \leftarrow SA - UnSolvedL.$$

At the end, the algorithm also computes the states that can be solved from the initial states with the goal labels by

$$Solved \leftarrow Solved - UnSolvedL.$$

Proposition 6.9 (Complexity of multiple goals planning). *Let D be the maximum length of paths in a state transition relation R. $multiGoalsPlan(R, I, G\mathcal{SPEC}, computePreSA)$ (Algorithm 25) can be computed in $O(D \times (|\mathcal{P}_S| + |\mathcal{P}_A|) + |G\mathcal{SPEC}|) < O(2^{|\mathcal{P}_S|} \times (|\mathcal{P}_S| + |\mathcal{P}_A|) + |G\mathcal{SPEC}|)$ BDD operations.*

Proof. $LCONS(G\mathcal{SPEC})$ can be computed in $O(|G\mathcal{SPEC}|)$ BDD operations (see Proposition 5.2). The rest of the computation is the same as that of $plan(R, I, G, computePreSA)$ (Algorithm 15) with the information regarding the consistent combinations of goals being carried out through the pre-image computations without changes. Therefore, the number of BDD operations needed by $multiGoalsPlan$ is $O(D \times (|\mathcal{P}_S| + |\mathcal{P}_A|) + |G\mathcal{SPEC}|)$ which is in turn bounded by $O(2^{|\mathcal{P}_S|} \times (|\mathcal{P}_S| + |\mathcal{P}_A|) + |G\mathcal{SPEC}|)$.

After we compute policies for all possible consistent goal combinations, we can then preserve in the joint policies that can achieve the maximal consistent sets of joint goals by

$$multiGoalSA = multiGoalsPlan(R, I, G\mathcal{SPEC}, computePreSA)$$
$$maxSA = maximal(\emptyset, multiGoalSA).$$

In combination with Algorithm 24 which computes the max-min preferred goals, we can compute the a joint policy that can achieve the max-min preferred goals by

$$multiGoalSA = multiGoalsPlan(R, I, G\mathcal{SPEC}, computePreSA)$$
$$maxMinSA = multiGoalSA \wedge \left(computeJointGoals(multiGoalSA^{\downarrow \mathcal{P}_L} \wedge LCONS(G\mathcal{SPEC})\right).$$

The policy $maxMinSA$ obtained is then the policy that can achieve the max-min preferred sets of goals.

6.4 Summary

This chapter introduces centralized and decentralized planning algorithms that can make use of the defeasible factored action theory introduced in Chapter 5 to produce a joint policy for the agents. Then a coordination mechanism was introduced to coordinate the behaviors of the agents so that the planned joint policy can be faithfully respected by all the agents. Thereafter, the planning algorithms were further extended to plan simultaneously for maximally consistent combinations of multiple goals. The approach is to propagate defeasible reasoning about the action theory and the specifications of goals through the state transition model of the world dynamics in the planning algorithms. Through this kind of defeasible reasoning over a state transition model of world dynamics, we are able to reasoning about the long term effects of defeasible action theories to match a defeasible theory of goals. As before, symbolic model checking techniques are employed to carry out the defeasible reasoning and its propagation through the state transition space simultaneously to obtain the polynomial complexity in terms of BDD operations.

The planning and coordination mechanisms introduced in this chapter shares the same weakness that of the defeasible action theory introduced in Chapter 5, so we need the following future directions: 1) the experimental study of the computations for planning and coordination mechanism using symbolic model checking techniques; 2) the relaxation of the Markov assumptions (e.g. mostly Markovian state transitions but with a library of procedural state-action sequences which can override the Markov assumptions in a number of cases) to see how the defeasible reasoning can propagate through a non-Markovian state transition space; 3) incorporating more domain specific knowledge of the world dynamics and handling numerical information in planning (see [Tang et al., 2011b,c] for our preliminary work in this direction).

Algorithm 25: $multiGoalsPlan(R, I, GS\mathcal{PEC}, computePreSA)$: Planning for multiple goal specifciations

Input : (1) R: The transition relation;
(2) I: The set of initial states;
(3) $GS\mathcal{PEC}$: The goal specifications;
(4) $computePreSA$: Either $ComputeWeakPreSA$ or $ComputeStrongPreSA$

$LG \leftarrow LCONS(GS\mathcal{PEC})$;
$Solved \leftarrow LG$;
$LI \leftarrow I \wedge LG^{\mathcal{P}_{L,\mathcal{G}}}$;
`/* Label the initial states with all possible goals combinations */`
$UnSolved \leftarrow LI - LG$;
$FT \leftarrow LG$;
$newSA \leftarrow \emptyset$;
while $newSA \neq \emptyset$ *OR* $UnSolved \neq \emptyset$ **do**
  $newSA \leftarrow computePreSA(R, FT)$;
  `/* Remove the state-action pairs whose states have already been solved */`
  $newSA \leftarrow newSA - Solved$;
  $SA \leftarrow SA \vee newSA$;
  $FT \leftarrow \exists_{\mathcal{P}_A} newSA$;
  $Solved \leftarrow Solved \vee FT$;
  $UnSolved \leftarrow UnSolved - Solved$;
end
`/* Compute the set labels of the goal combinations that haven't been solved */`
$UnSolvedL \leftarrow \exists_{\mathcal{P}_S} UnSolved$;
$SA \leftarrow SA - UnSolvedL$;
$Solved \leftarrow Solved - UnSolvedL$;
if $Solved \neq \emptyset$ **then**
  `/* Return the solved combinations of goals; there might some combinations are not solved */`
  return $\langle Sovled, SA \rangle$;
else
  return $\emptyset$;
end

Chapter 7

Argumentation-based reasoning on state space

In this chapter, I will introduce the theory of argumentation-based reasoning as discussed in the literature. Based on this work, I will underpin the defeasible state transition combinations in Chapter 5, the centralized and decentralized planning and coordination algorithms in Chapter 6 with an adaption of Dung's semantics which are well-accepted in the AI literature of argumentation theory [Dung, 1995].

This chapter will start with an introduction of argumentation theory in the literature (see Section 7.1). An abstract argumentation framework will be created in Section 7.2 and Section 7.3 to provide a unified framework to talk about work in the argumentation theory literature (Section 7.4.1 on argumentation semantics, Section 7.4.2 on incorporating preference into argumentation, and Section 7.4.3 on the relationship to other non-monotonic reasoning in AI) related to this work. The contributions of this dissertation will start with Section 7.5. Sections 7.5– 7.13 will introduce new concepts of arguments, defeat, preference and an adapted argumentation semantics for reasoning about actions (along with their conditions and effects), and for multiagent planning.

7.1 Argumentation theory

Argumentation theory has been proposed for both reasoning and modeling dialogues between agents. Argumentation-based reasoning is a defeasible reasoning paradigm in which reasons from premises to conclusions are recorded into a data structure called an *argument*, *conflict*s between these arguments are recorded in a *defeat relation*, and a conflict-free set of acceptable arguments can then be reached with respect to a notion of collective acceptability [Dung, 1995]. On the other hand, an argumentation-based dialogue is a model in which arguments are exchanged among multiple parities proving or disproving logical propositions [Walton, 2009] by either supporting or undermining intermediate conclusions during the course of a dialogue.

The argumentation-based reasoning mechanism can be viewed as a mathematical abstraction of the dialectical principles which are extracted from the study of human argumentation (Walton and Krabbe [Walton and Krabbe, 1995], Hamblin [Hamblin, 1970], and Toulmin [Toulmin, 1958; Toulmin et al., 1984]). The input to the reasoning scheme is knowledge represented in a formal language and sometimes an additional notion of preference over the knowledge, which can be represented inside or outside the formal language; its output is a characterization of the arguments and their conclusions. The two key components of an argumentation system are the process to construct arguments and the process to analyze the conflict pattern between these arguments. In the argument construction process, reasoning from the knowledge database to conclusions is recorded into a syntactic structure called an argument structure; then in the conflict analysis process, the conflicts between arguments will be captured and analyzed to characterize the acceptability of arguments and their conclusions. A set of arguments is *conflict-free* if there are no conflicts between any two arguments within this set. A set of arguments is collective acceptable if any argument which defeats an argument in the set is defeated by an argument in this set. The notion of collective acceptability exerts a form of stability that can resist to external challenges. The most interesting property of argumentation based reasoning is the concept of "external stability" [Dung, 1995] through which a set of coherent beliefs and reasons is characterized by the relations between

the arguments "internally" supporting the beliefs and reasons and the arguments "externally" supporting the contradictory beliefs and reasons. The concept of "external stability" is proved to correspond to the solution concept of N-person game [Dung, 1995]. This makes argumentation-based reasoning an attractive inference engine to organize the information about actions and goals in multiagent systems.

Other semantics for argumentation-based reasoning has also been proposed, such as [Caminada and Amgoud, 2007], [Cayrol and Lagasquie-Schiex, 2005], [Jakobovits and Vermeir, 1999], [Besnard and Hunter, 2001], [Pollock, 2001], which have expanded our understanding of what argumentation can be used for, and have created a bridge to possibility theory and plausibility theory in the field of reasoning about uncertainty.

7.2 Abstract argumentation frameworks

The basic concepts of existing approaches in the literature of argumentation-based reasoning can be captured by a framework with a 5-tuple:

$$\mathsf{AF} = \langle \mathcal{L}, \mathcal{RS}, \mathbf{K}, \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$$

where

1. $\mathcal{L}$ is the language to represent the input knowledge;
2. $\mathcal{RS}$ is a collection of identifiable inference schemes capturing the reasoning patterns in the language $\mathcal{L}$; typically $\mathcal{RS}$ is a list of inference rules associated with $\mathcal{L}$; in some systems, particularly those based on logic programming, a subset of $\mathcal{RS}$ is a set of knowledge-dependent inference schemes used as an efficient way to represent knowledge;
3. ARG is a collection of arguments which are structures to record reasoning from knowledge $\mathbf{K}$ represented in the language $\mathcal{L}$ using $\mathcal{RS}$ to its conclusions; the aspects of the reasoning to be recorded reflect the notion of arguments used in the system;

4. DFT is a binary relation between arguments in ARG which models a certain notion of conflict existing in the knowledge represented in $\mathcal{L}$ and $\mathcal{RS}$;

5. PREF is a preference ordering on the arguments ARG extracted from knowledge represented in $\mathcal{L}$ and $\mathcal{RS}$; PREF is also used by some systems to refine the notion of conflicts captured in DFT.

With the general framework AF, when our major concern is the defeat relation DFT, we simplify the framework notation into an *abstract argumentation framework* of Dung [1995] :

$$\mathsf{AFD} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle;$$

when our major concern is the defeat relation DFT and the preference relation PREF, we simplify the framework notation into a *preference based argumentation framework* of Amgoud and Cayrol [2002b,a]:

$$\mathsf{AFP} = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle.$$

Using the argumentation framework AF, the input knowledge $\mathbf{K}$ can be decomposed into by a triple

$$\mathbf{K} = \langle \mathsf{KL}, \mathsf{KR}, \mathsf{KPREF} \rangle$$

where

1. KL is a list of sentences that can be represented as formulae in the QBF language $\mathcal{L}$;

2. KR is the knowledge that can be represented as application dependent inference schemes in terms of $\mathcal{RS}$; and

3. KPREF is a preference ordering (most often a pre-ordering) on the knowledge represented in $\mathcal{L}$ and $\mathcal{RS}$;

An argumentation framework AF and a collection of knowledge $\mathbf{K}$ form an *argumentation system*

$$\mathsf{AS} = \langle \mathsf{AF}, \mathbf{K} \rangle$$

an argumentation can be viewed as a function

$$\underline{\mathsf{S}} = \mathsf{F}_{arg}(\mathsf{AS})$$

which, based on $\mathbf{K}$, outputs the conclusions represented in $\underline{\mathsf{S}}$, a collection of mathematical structures to represent the results, according to certain principles of argumentation. The collection $\underline{\mathsf{S}}$ can be a collection of sentences or sentence sets in $\mathcal{L}$ or a collection of arguments or argument sets in ARG each with its associated acceptability status. The acceptability status is used to characterize the plausibility of the sentences or arguments being taken into account in $\underline{\mathsf{S}}$. For example, three popular status values are "accepted", "rejected" and "unknown". The argumentation reasoning function F_{arg} is typically given as a set of mathematical constraints extracted from the study of human argumentation and their reflection in the study of nonmontonic reasoning in AI. In some systems, this set of mathematical constraints is preserved in a more constructive form as dialectical proofs and the corresponding procedure to search in the space of potential proofs. If the argumentation framework is clear in the context, the reasoning function can be simply denoted by

$$\underline{\mathsf{S}} = \mathsf{F}_{arg}(\mathbf{K}).$$

7.3 Set-theoretic arguments

In this section, I will introduce the concept of *set-theoretical argument*. The concept of set-theoretical argument is a generalized form of arguments drawn from the work of Amgoud and her colleagues [Amgoud and Cayrol, 2002a,b]. This framework will abstract away the inference procedure by which the arguments are created and only keep track of the premises that the argu-

ments are based on. Then, it will be instantiated into a definition of deductive arguments which is the definition of arguments in [Amgoud and Cayrol, 2002a,b].

Definition 7.1 (Set-theoretical argument). *A* set-theoretical argument *based on a knowledge base* $\mathsf{KL} \subseteq \mathcal{L}$ *is pair* $A = (H, h)$ *where* $H \subseteq \mathsf{KL}$ *(*$H \neq \emptyset$*) and* $h \in \mathcal{L}$*. H, denoted by* $support(A)$*, is called the support and h, denoted by* $conclusion(h)$*, is called the conclusion of the argument.* $\mathsf{ARG}(\mathsf{KL})$ *denotes the set of all arguments which can be constructed from* KL*.*

7.4 Related work

7.4.1 Dung's argumentation semantics

In the literature, Dung's abstract notion of argumentation and his extension-based semantics [Dung, 1995] is one of the most influential works in AI argumentation. Dung abstracted away the detailed structure of argument and modeled the conflicts between arguments as a binary relationship. Thereafter, he grouped arguments into different extensions with respect to the defeat relationship, and used the extensions to characterize the acceptability of the arguments inside them. In this way, the acceptability characterization of an individual argument can possess the property of "external stability" in which the status of an argument does not depend on its parts but depends on its interaction with the other arguments. Dung also proved that several well known nonmontonic reasoning approaches are instantiations of his abstract argumentation framework, albeit with different semantics, and formalized several application-level problems as instantiations of the abstract argumentation systems and showed that the solutions of these problems can be captured by the appropriate model of argumentation semantics. In this section, I will characterize Dung's abstract argumentation using the model I introduced above.

The argumentation framework is

$$\mathsf{AF} = \langle \mathcal{L}, \mathcal{RS}, \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$$

where

1. $\mathcal{L}$ and $\mathcal{RS}$ are not specified in this framework;
2. ARG is set of arguments which are primitive in this framework;
3. $\mathsf{DFT} \subseteq \mathsf{ARG} \times \mathsf{ARG}$ is a binary relationship, each element of which models a defeat relationship between two arguments;
4. PREF is empty in this framework.

As an abstract framework, the way the input knowledge $\mathbf{K}$ is represented is not specified in this work, but an appropriate way to generate the arguments ARG and defeats DFT is assumed instead. Effectively, the input of Dung's argumentation framework is

$$\mathsf{AFD} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle.$$

Example 7.1 (Dung's abstract argumentation framework).

- i_1*: My government can not negotiate with your government because your government doesn't even recognize my government*
- a*: Your government doesn't recognize my government either*
- i_2*: But your government is a terrorist government*

In the example above, we assume that there are two external components — one which can generate valid arguments: $\mathsf{ARG} = \{i_1, a, i_2\}$*, and another one which can identify the defeat relationship between arguments:* $\mathsf{DFT} = \{(i_1, a), (a, i_1), (i_2, a)\}$.

The argumentation reasoning function can be defined as follows:

$$\underline{\mathsf{S}} = \mathsf{F}_{arg}(K)$$

where

1. $\underline{\mathsf{S}} = \langle Acc_{admissible}, Acc_{preferred}, Acc_{stable}, Acc_{complete}, Acc_{grounded} \rangle$ where $Acc_{admissible}$, $Acc_{preferred}$, Acc_{stable}, $Acc_{complete}$, $Acc_{grounded}$ are subsets of ARG that characterize the status of the arguments in ARG according to intuition about human argumentation,

2. $\underline{\mathsf{S}} = \mathsf{F}_{arg}(K)$ will be defined below.

The formal definitions of $\underline{\mathsf{S}}$ will be given below. In [Dung, 1995], Dung implemented $\mathsf{F}_{arg}(K)$ by directly encoding DFT, and the formal definitions of $\underline{\mathsf{S}}$ into a logic program as a meta-interpretor of the argumentation framework.

Definition 7.2 (Defend **characteristic function of argumentation framework).** *Let* $\langle \mathsf{ARG}, \mathsf{DFT} \rangle$ *be an argumentation framework, and* $S \subseteq \mathsf{ARG}$*. An argument* A *is* defend*ed by* S *iff* $\forall B \in \mathsf{ARG}$ *if* $(B, A) \in \mathsf{DFT}$ *then* $\exists C \in S$ *such that* $(C, B) \in \mathsf{DFT}$*.*

Definition 7.3 (Conflict-free). *A set of arguments* $S \subseteq \mathsf{ARG}$ *is* conflict-free *with respect to an argumentation framework* $\mathsf{AFD} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$ *iff for any two arguments* A *and* B *in* S*,* $(A, B) \notin \mathsf{DFT}$*.*

Definition 7.4 (Admissible). *A set of arguments* $S \subseteq \mathsf{ARG}$ *is* admissible *with respect to an argumentation framework* $\mathsf{AFD} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$ *iff*

- S *is conflict-free, and*
- *for every argument* $A \in S$*, if there exists another argument* $B \in \mathsf{ARG}$ *such that* $(B, A) \in \mathsf{DFT}$*, then there exists an argument* $C \in S$ *such that* $(C, B) \in \mathsf{DFT}$*.*

Definition 7.5 (Complete extension). *With respect to an argumentation framework* $\mathsf{AFD} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$*, an admissible set* S *of arguments is a* complete extension *iff every argument in* ARG *that can be defended by* S *belongs to* S*.*

Definition 7.6 (Preferred extension). *With respect to an argumentation framework* $\mathsf{AFD} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$*, a* preferred extension *of an argumentation framework* AFD *is a maximal complete extension.*

Definition 7.7 (Grounded extension). *With respect to an argumentation framework* $\mathsf{AFD} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$*, a* grounded extension *of an argumentation framework* AFD *is a minimal complete extension.*

The complete, preferred, and grounded extensions can also be defined as functions.

Definition 7.8 (Characteristic function of argumentation framework).
Let $\mathsf{AFD} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$ *be an argumentation framework, and* $S \subseteq \mathsf{ARG}$*, a* DFD *function is defined as:*

$$\mathsf{DFD}_{\mathsf{AFD}}(S) = \{A \in \mathsf{ARG} | A \textit{ is defended by } S \textit{ with respect to } \mathsf{DFT}\}.$$

Now, for a function $F : D \to D$ where D is the domain and the range of the function, a fixed point of F is an $x \in D$ such that $x = F(x)$. When the D is associated with an ordering P — for example, P can be set inclusion over the power set D of arguments — x is a *least fixpoint* of F if x is a least element of D with respect to P and x is a fixed point.

Definition 7.9 (Fixed point extensions). *Let* $\mathsf{AFD} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$ *be an argumentation framework, and* $S \subseteq \mathsf{ARG}$*.*

- S *is a* complete extension *of* AFD *iff* S *is a fixed point of* $\mathsf{DFD}_{\mathsf{AFD}}$*.*
- S *is a* grounded extension *of* AFD *iff* S *is a least fixed point of* $\mathsf{DFD}_{\mathsf{AFD}}$*.*
- S *is a* preferred extension *of* AFD *iff* S *is a largest fixed point of* $\mathsf{DFD}_{\mathsf{AFD}}$*.*

Definition 7.10 (Stable extension). *In an argumentation framework* $\mathsf{AFD} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$*, a set of arguments* $\mathsf{Args} \subseteq \mathsf{ARG}$ *is a* stable extension *iff* Args *is conflict-free and defeats any arguments that are not in* Args*. Namely,*

- Args *is conflict free, and*
- *for every argument* $B \notin Args$ *there exits an argument* $A \in$ ARG *that defeats* B.

These notions of acceptability are different ways to characterize argument status based on different intuitions about human argumentation in terms of the interaction among the arguments through the abstract concept of defeat. The $Acc_{admissible}$ (admissible extension) is a set of arguments that have no defeats inside the set (it may have defeaters outside the set) and so can defend themselves against the defeaters; $Acc_{preferred}$ (preferred extension) is a maximally admissible set of arguments (maximality is w.r.t to set inclusion); Acc_{stable} (stable extension) is a set of arguments that have no defeat inside the set and defeat any other arguments outside the set; $Acc_{complete}$ (complete extension) is a set of arguments that have no defeats inside the set and contain all the arguments which can be defended by the arguments in the set; $Acc_{grounded}$ (grounded extension) is the smallest of complete sets of arguments (w.r.t to set inclusion)[1]. In the above definitions, a basic requirement is that the extension should be conflict-free. It models the intuition about human argumentation that the acceptable set of one's arguments should not contain any two arguments that defeat each other. Without further information other than the defeat relationship, DFT, aside from the basic requirement for the arguments to be conflict-free, it is the concept of "external stability" which makes the acceptable set of arguments together collectively stable with respect to the arguments outside the extension. The idea of a preferred extension adds an additional level of "external stability" in terms of maximality on the collectively defendable arguments; the idea of a stable extension adds "external stability" in terms of defeating not only the arguments defeating an argument inside but also any other arguments outside the extension; the idea of a complete extension adds "external stability" in terms of containing all the arguments which the extension can collectively defend; and finally, the idea of a grounded extension adds "external stability" in terms of minimality of the complete extensions.

[1]Please note that, depending on the properties of defeat relationship between arguments, there may be more than one admissible extension, preferred extension, stable extension, complete extension and grounded extension.

Dung showed that when the argumentation framework is what he called *finitary*, i.e. there is no infinite sequence of arguments $A_1, A_2, \ldots$ such that A_{i+1} defeats A_i, the framework will have exactly one complete extension which is also a preferred, stable, grounded extension. In an argumentation framework with a finite number of arguments, this result means that when there is no cycle of defeats, the acceptable extensions coincide and become unique.

In this work, Dung also showed that the abstract argumentation framework can be used to investigate the solution concepts of practical problems using the N-person game and stable marriage problem as two examples. Here I will summarize Dung's N-person game example for similar game structures are present in many multiagent systems. In Dung's demonstration, arguments are imputations, defeats are the domination relationship between imputations, and then the stable, preferred, and $F(\emptyset)$ (the first step toward the grounded extension) corresponds to game theorectical solution concepts of "established order of society", the preferred solution concept, and the "core imputations" concept respectively. In Dung's demonstration, an argument is an imputation which is a vector of payoffs the players can receive after playing the game depending on how these players form coalitions; defeats are relationships where one imputation dominates another one: an imputation $\vec{p_1}$ is dominated by another imputation $\vec{p_2}$ if it is the case that some players can behave differently from the imputation $\vec{p_1}$ by forming a different configuration of coalitions so that a new imputation $\vec{p_2}$ is obtained and some players under $\vec{p_2}$ can receive better payoffs. The stable extension is the "established order of society" from which is impossible for the players to deviate; the preferred extension is a maximal set which contains the imputations that can withstand any attacks from outside; the arguments in $F(\emptyset)$ are the imputations which are not dominated by any other imputations. This result has two important indications. First, it justifies the correctness of Dung's abstract argumentation framework by showing that the framework captures the important logical structure of complex problems such as the N-person game. Second, it inspires us that an argument should not be limited to take the form of a logical proof or a deductive proof — it can be something that can not be expressed by a known logic but some collection of information with partially known structure, provided that the structure contains enough information to extract an

appropriate concept of defeat between two arguments.

Dung also showed that abstract argumentation and its semantics can capture the logical structure of many nonmontonic reasoning systems. This idea was further developed in [Bondarenko et al., 1997] by Dung and his colleagues.

An important contribution of Dung's work is that he isolated the detailed structure of an argument from the argumentation reasoning model. In this way, Dung provided us with a clear view of how to model the reasoning behind human argumentation and grounded much work on argumentation based reasoning within AI in the past 10 years. Dung's major ideas are inherited and extended in the works described in [Amgoud and Cayrol, 2002b], [Bondarenko et al., 1997], [Dung et al., 2006], and so on. (Dung's work also showed the equivalency between his framework and the systems that inherited Pollock's status assignment of [Pollock, 1992].)

7.4.2 Amgoud and Cayrol's preference based argumentation framework

Amgoud and Cayrol [2002a,b] instantiated Dung's argumentation framework [Dung, 1995], extended it with a notion of preference, adopted Dung's grounded (fixed-point) semantics and proposed a constructive dialectical proof theory for the semantics. The preferences over arguments are translated from preferences over knowledge. The defeat relationships are formal notions of conclusion rebutting, premise undercutting, and sub-argument contradicting. In this work, the argumentation framework is as follows:

$$\mathsf{AF} = \langle \mathcal{L}, \mathcal{RS}, \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$$

where

1. $\mathcal{L}$ is a logical language with the standard connectives $\wedge, \vee, \neg, \equiv$, which is finite or countably infinite;

2. $\mathcal{RS}$ is entailment $\vdash$ defined in the standard way over $\mathcal{L}$;

3. ARG is as defined in Definition 7.11 below;

4. DFT is as defined in Definition 7.12 below;

5. PREF is as defined in Definition 7.13 below;

Definition 7.11 (Deductive argument). *A* deductive argument *based on* $\mathsf{KL} \subseteq \mathcal{L}$ *is set-theoretical argument* $A = (H, h)$ *following Definition 7.1 such that*

1. H *is consistent with respect to* $\mathcal{L}$,

2. $H \vdash h$, *and*

3. H *is minimal in the sense of set inclusion.*

This definition of an argument can be understood as a set of constraints on how information can be clustered as arguments. Condition (1) ensures that an argument is coherent. The coherence of an agent's information is defined in terms of the consistency of the language $\mathcal{L}$ in which the information is written. Condition (2) can be understood as insisting that the conclusion of an argument should be supported by a set of information in the sense of inference in the language $\mathcal{L}$. From a model theoretical point of view, condition (2) means that $\mathcal{I}(H) \subseteq \mathcal{I}(h)$ ($\mathcal{I}$ is an interpretation function following Definition 2.2). Condition (3) can be understood as saying that no redundant information should appear in an argument.

Definition 7.12 (Logical defeats). *Let* (H_1, h_1), (H_2, h_2) *be two arguments of* ARG(KL).

1. (H_1, h_1) rebut*s* (H_2, h_2) *iff* $h_1 \equiv \neg h_2$.

2. (H_1, h_1) undercut*s* (H_2, h_2) *iff* $\exists h \in H_2$ *such that* $h_1 \equiv \neg h$.

3. (H_1, h_1) contradict*s* (H_2, h_2) *iff* (H_1, h_1) *rebuts a subargument of* (H_2, h_2). *An argument* (H', h') *is a* subargument *of another argument* (H, h) *iff* $H' \subseteq H$.

The binary relations rebut, undercut, *and* contradict *gather all pairs of arguments satisfying conditions (1), (2) and (3) respectively.*

Definitions of rebut, undercut, and contradict will be given below and we will collectively refer to the relations as DFT if no distinction is necessary or we are describing them collectively. Infix notations such as $\mathsf{rebut}((H_1, h_1), (H_2, h_2))$, $\mathsf{undercut}((H_1, h_1), (H_2, h_2))$, $\mathsf{contradict}((H_1, h_1), (H_2, h_2))$ and $\mathsf{DFT}((H_1, h_1), (H_2, h_2))$ are also used to denote defeats between two arguments.

Translation of preferences

Definition 7.13 (Knowledge preference order). *A* preference order PREF over the knowledge $\mathsf{KL} \subseteq \mathcal{L}$, *denoted by* $\geq$, *is a preorder defined as*

$a \geq b$ *means* $a \in \mathsf{KL}$ *is at least as "good" as* $b \in \mathsf{KL}$.

If $\geq$ *is a total order on* $\mathsf{KL} \subseteq \mathcal{L}$, *then* KL *can be stratified into* $\mathsf{KL}_1 \cup \ldots \cup \mathsf{KL}_n$ *such that all beliefs in* KL_i *are equally strong and are stronger than all beliefs in* KL_j *where* $j > i$.

Preference over arguments is defined based on the preference over the supports of arguments as follows:

Definition 7.14 (Argument preference order). *Let* PREF *be a (partial or total) preordering on subsets of* KL, *and let* $(H, h), (H', h')$ *be two arguments of* $\mathsf{ARG}(\mathsf{KL})$. (H, h) *is* PREF*-preferred to* (H', h') *iff* H *is preferred to* H' *w.r.t.* PREF.

Amgoud and Cayrol first gave a notion of levels for total order preference:

Definition 7.15 (Support level). *For a non-empty subset* $H \subseteq \Sigma$, *let* $H_i = H \cap \Sigma_i$,

$$level(H) = MIN\{i | 1 \leq i \leq n \text{ and } H_{i+1} \cup \ldots \cup H_n = \emptyset\}. \tag{7.1}$$

It means that the level of a set H of formulae is determined by the formula of the highest belief level in H (namely the least preferred element in H since the higher the level the less it is preferred according to Definition 7.13). With this notion, Amgoud and Cayrol offered three ways of translating the preference on KL into the preference on subsets of $\mathcal{L}$; they are BDP, ELI, and $WBDP$, and defined as follows.

Definition 7.16 (Preference translations).

1. *BDP*

   Let H and H' be two consistent subsets of Σ.

   - *If $\geq$ on Σ is a total order, H is preferred to H' iff $level(H) \leq level(H')$.*
   - *If $\geq$ on Σ is a partial order, H is preferred to H' iff $\forall k \in H$, $\exists k' \in H'$ such that $k > k'$ (i.e. $k \geq k'$ and not $k' \geq k$).*

2. *ELI*

   Let H and H' be two consistent subsets of Σ.

   - *If $\geq$ on Σ is a total order, H is preferred to H' iff $level(H - H') < level(H' - H)$.*
   - *If $\geq$ on Σ is a partial order, H is preferred to H' iff $\forall k \in H - H'$, $\exists k' \in H' - H$ such that $k > k'$ (i.e. $k \geq k'$ and not $k' \geq k$).*

3. *WBDP (only for a total order over subsets of Σ)*

   - *Firstly, let $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \ldots \cup \Sigma_n$ be a stratified belief base and H a consistent subset of Σ. For each $1 \leq k \leq n$, let us define*

     $$level_k(H) = level(H_1 \cup \ldots \cup H_k) = MIN\{i | 1 \leq i \leq k \text{ and } H_{i+1} \cup \ldots \cup H_k = \emptyset\}$$

     (with $MIN\emptyset$ taken equal to k). Then, $level(H) = level_n(H)$.

   - *Let H and H' be two consistent subsets of Σ. For a total ordering $\geq$ on Σ, H is preferred to H' iff $\exists k$ such that $1 \leq k \leq n$, $level_k(H) < level_k(H')$ and for each $j > k$, $level_j(H) = level_j(H')$.*

The properties of a preference translation $P \in \{BDP, ELI, WBDP\}$ can be partially understood in terms of the following properties:

Minimality P satisfies "Minimality" for set inclusion iff $\forall H, H' \subseteq \Sigma$, if $H' \subseteq H$, then H' is preferred to H w.r.t PREF.

And P satisfies "And" iff for any H, H', H'' subsets of the belief base Σ, if H is preferred to H' w.r.t. PREF and H'' is preferred to H' w.r.t. PREF, then $H \cup H'$ is preferred to H' w.r.t. P.

Monotonicity P satisfies "Monotonicity" iff for any H, H', H'' subsets of the belief base Σ such that $H' \cap H'' = \emptyset$. If H is preferred to H', then $H \cup H'$ is preferred to $H' \cup H''$.

The properties of $BDP, ELI, WBDP$ are summarized in table 7.1.

Properties	Minimality	And	Monotonicity
BDP	•	•	∘
ELI(total order)	•	∘	•
$WBDP$	•	∘	∘

Table 7.1: Summary of preference properties. Symbols: •, for success and ∘, for failure

With the elements of the argumentation framework instantiated, the knowledge available to the agent can be instantiated as follows:

$$K = \langle \mathsf{KL}, \mathsf{KR}, \mathsf{KPREF} \rangle$$

where

1. KL is a subset of sentences in $\mathcal{L}$;
2. KR is empty;
3. KPREF is translated from PREF.

Semantics

Different notions of preference translation P give different argumentation reasoning functions for this system.

$$\underline{\mathsf{S}}^P = \mathsf{F}^P_{arg}(K)$$

where

1. $\underline{\mathsf{S}}^P = \langle Acc^P_{Rebut}, Acc^P_{Undercut} \rangle$;

2. $\underline{\mathsf{S}}^P = \mathsf{F}^P_{arg}(K)$ is defined as follows

$$Acc^P_x = \cup \mathcal{F}^{i>0}(\emptyset) = C^P_x \cup [\cup \mathcal{F}^{i>1}(C^P_x)]$$

for $x \in \{Rebut, Undercut\}$ where

$$\mathcal{F}(S) = \{A \in \mathcal{A} | A \text{ is defended by } S \subseteq \mathsf{ARG} \text{ with respect to } x \text{ and } P\};$$

an argument A is said to be *defended by* S iff $\forall B \in \mathcal{A}$, if $(B, A) \in x$ and $(A, B) \notin P$ then $\exists C \in S$ such that $(C, B) \in x$ and $(B, C) \notin P$; $C^P_x \subseteq \mathsf{ARG}$ denotes the set of arguments defending themselves, which is essentially an alias for Acc^P_x defined above.

In summary, the acceptable arguments output by $\mathsf{F}^P_{arg}(K)$ are the ones which defend themselves against their defeaters and also the arguments which are defended (directly or indirectly) by the arguments in C^P_x. In [Dung, 1995], Dung showed that if the argumentation framework is finitary (i.e., for each argument A there are only a finite number of arguments that defeat A), and the function $\mathcal{F}$ is continuous, then its least fixpoint can be obtained by iterative application of $\mathcal{F}$ to the empty set. The definition of argument and defeat in this system guarantee that the condition (only finitary many points to rebut or undercut) holds.

7.4.3 Bondarenko et al.'s assumption-based argumentation

Bondarenko et al. [1997] proposed an abstract assumption-based argumentation-theoretic approach, which I will call the "BDKT approach", to various kinds of nonmonotonic reasoning in AI, such as default logic, circumscription and so on. It is based on the notion of a deduction system, the use of assumptions to extend the initial theory using the deduction system, a notion of defeat among sets of assumptions, and several notions of semantics based on the defeats between sets of assumptions.

The argumentation-based framework is instantiated as follows:

$$\mathsf{AF} = \langle \mathcal{L}, \mathcal{RS}, \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$$

where

1. $\mathcal{L}$ is a formal language consisting of countably many sentences; and associated with $\mathcal{L}$, there is an additional language processor, denoted by $^-$, which is a mapping from assumptions $Ab \subseteq \mathcal{L}$ into $\mathcal{L}$, where $\overline{\alpha}$ denotes the contrary of α in addition to the negation of α (if any) defined in $\mathcal{L}$;

2. $\mathcal{RS}$ is a set of inference rules of the form

$$\frac{\alpha_1, \ldots, \alpha_n}{\alpha}$$

   where $\alpha, \alpha_1, \ldots, \alpha_n \in \mathcal{L}$ and $n \geq 0$; and by overloading the notation, I will also use $\mathcal{RS}$ to refer to the associated notion of deduction (Definition 7.17 below);

3. $\mathsf{ARG} = 2^{Ab}$: There is no formal definition of an argument in this system, but an equivalent concept is given as subsets of assumptions Ab;

4. DFT is defined as "attack" in Definition 7.18 below;

5. PREF is empty (is not used in this system).

Definition 7.17 (Deduction). *A deduction from a theory $T \subseteq \mathcal{L}$ is a sequence $\beta_1, \ldots, \beta_m$, where $m > 0$, such that for all $i = 1, \ldots, m$,*

- *$\beta_i \in T$, or*
- *there exists $\frac{\alpha_1, \ldots, \alpha_n}{\beta_i}$ in $\mathcal{R}$ such that $\alpha_1, \ldots, \alpha_n \in \{\beta_1, \ldots, \beta_{i-1}\}$.*

$T \vdash \alpha$ means that there is a deduction from T whose last element is α. $Th(T)$ is the set $\{\alpha \in \mathcal{L} | T \vdash \alpha\}$.

This definition can be viewed as an instantiation of the notions of $\mathcal{L}$ and the corresponding entailment $\vdash$ in both Elvang-Goransson and Hunter's system [Elvang-Gøransson and Hunter, 1995] and Amgoud and Cayrol's work [Amgoud and Cayrol, 2002b].

Definition 7.18 (BDTK-defeats). *Given an assumption-based framework* $\langle T, Ab, \bar{} \rangle$,

- *a set of assumptions* $\Delta \subseteq Ab$ *attacks an assumption* $\alpha \in Ab$ *if and only if* $T \cup \Delta \vdash \overline{\alpha}$,
- *a set of assumptions* $\Delta \subseteq Ab$ *attacks a set of assumptions* $\Delta' \subseteq Ab$ *if and only if* Δ *attacks some assumption* $\alpha \in \Delta'$.

With the argumentation part of the framework instantiated, the knowledge representation is then instantiated as follows:

$$K = \langle \mathsf{KL}, \mathsf{KR}, \mathsf{PREF} \rangle$$

where

1. $\mathsf{KL} = T \cup Ab$ is composed of two disjoint sets $T, Ab \subseteq \mathcal{L}$ and $Ab \neq \emptyset$ are the input assumptions; T is to express a given set of beliefs, and Ab is a set of assumptions (the initial understanding of possibilities) that can be used to extend T;
2. KR may be given as a set of application dependent inference rules;
3. PREF is empty.

Therefore in this system, K is a tuple $\langle T, Ab, \bar{} \rangle$ defined as follows:

Definition 7.19 (Assumption-based argumentation framework). *Given a deductive system* $(\mathcal{L}, \mathcal{R})$, *an assumption-based framework with respect to* $(\mathcal{L}, \mathcal{R})$ *is a tuple* $\langle T, Ab, \bar{} \rangle$ *where*

- $T, Ab \subseteq \mathcal{L}$ *and* $Ab \neq 0$,
- $\bar{}$ *is a mapping from* Ab *into* $\mathcal{L}$, *where* $\bar{\alpha}$ *denoted the contrary of* α.

The following concepts are needed to define the argumentation reasoning functions:

Definition 7.20 (BDTK conflict-free). *Given an assumption-based framework* $\langle T, Ab, {}^{-} \rangle$ *and* $\Delta \subseteq Ab$,

- Δ *is conflict-free if and only if for all* $\alpha \in Ab$, $T \cup Ab \nvdash \alpha, \overline{\alpha}$,
- Δ *is maximal conflict-free if and only if* Δ *is conflict-free and there is no conflict-free* $\Delta' \supset \Delta$.

Definition 7.21 (BDTK closed). *Given an assumption-based framework* $\langle T, Ab, {}^{-} \rangle$, *a set of assumptions* $\Delta \subseteq Ab$ *is closed iff* $\Delta = \{\alpha \in Ab | T \cup \Delta \vdash \alpha\}$.

Definition 7.22 (BDTK defend). *A set of assumptions* Δ *defends an assumption* α *iff for each closed set of assumptions* Δ', *if* Δ' *attacks* α *then* Δ *attacks* $\Delta' - \Delta$.

Definition 7.23 (BDTK argumentation characteristic function Def**).** *Given an assumption-based framework* $\langle T, Ab, {}^{-} \rangle$ *and a set of assumptions* $\Delta \subseteq Ab$, $Def(\Delta) = \{\alpha | \Delta$ *defends* $\alpha\}$.

Now we can instantiate the argumentation reasoning function as follows:

$$\underline{\mathsf{S}} = \mathsf{F}_{arg}(K)$$

where

1. $\underline{\mathsf{S}} = \langle Acc_{naive}, Acc_{stable}, Acc_{admissible}, Acc_{preferred}, Acc_{complete}, Acc_{well\text{-}founded} \rangle$
2. $\underline{\mathsf{S}} = \mathsf{F}_{arg}(K)$ is defined as follows
   (a) A set $\Delta \subseteq Ab$, denoted by Acc_{naive}, is a *naive extension* iff
      - Δ is conflict-free, and
      - Δ is maximal, namely there is no conflict-free $\Delta' \supset \Delta$.
   (b) A set $\Delta \subseteq Ab$, denoted by Acc_{stable}, is a *stable extension* iff
      - Δ is closed,
      - Δ does not attack itself, and

- Δ attacks each assumption $\alpha \notin \Delta$.

(c) A closed set of assumptions $\Delta \subseteq Ab$, denoted by $Acc_{admissible}$, is an *admissible extension* iff

- Δ does not attack itself, and
- for each closed set of assumptions $\Delta' \subseteq Ab$, if Δ' attacks Δ then Δ attacks Δ'.

(d) A set of assumptions $\Delta \subseteq Ab$, denoted by $Acc_{preferred}$, is a *preferred* extension iff Δ is maximal (w.r.t. set inclusion) admissible.

(e) A set of assumptions $\Delta \subseteq Ab$, denoted by $Acc_{complete}$, is a *complete* extension if and only if

- Δ is closed, and
- $\Delta = Def(\Delta)$.

(f) A set of assumptions Δ, denoted by $Acc_{well\text{-}founded}$, is a *well-founded extension* if and only if Δ is the intersection of all complete sets of assumptions.

The relation to other kind of nonmontonic reasoning

The above abstract assumption-based framework $\langle T, Ab, {}^{-} \rangle$ can be further instantiated to capture the forms and semantics of several non-monotonic logics (taken from [Bondarenko et al., 1997])

- Theorist [Poole et al., 1987] is the framework $\langle T, Ab, {}^{-} \rangle$ defined as follows:
  - A deductive system $(\mathcal{L}, \mathcal{R})$
  - $T \subseteq \mathcal{L}$ is consistent and $Ab \subseteq \mathcal{L}$
  - ${}^{-}$ is defined as $\bar{\alpha} = \neg\alpha$ for each $\alpha \in Ab$
- Logic programming [Eshghi and Kowalski, 1989] is the framework $\langle T, HB_{not}, {}^{-} \rangle$ defined as follows:

- $\mathcal{L} = Lit \cup \{\alpha \leftarrow \beta_1, \ldots, \beta_n | \alpha \in \mathcal{HB}$ and $\beta_1, \ldots, \beta_n \in Lit$ and $n \geq 0\}$ where $\mathcal{HB}$ stands for the Herbrand base, i.e. the set of all ground atoms constructible from the constant symbols and function symbols of the language, $\mathcal{HB}_{not}$ stands for the set $\{not\ \alpha | \alpha \in \mathcal{HB}\}$, and lit stands for $\mathcal{HB} \cup \mathcal{HB}_{not}$
- T is a normal logic program
- $\mathcal{R}$ is the set of all inference rules of the form

$$\frac{\alpha \leftarrow \beta_1, \ldots, \beta_n \quad \beta_1, \ldots, \beta_n}{\alpha}$$

  where $\alpha \in \mathcal{HB}$ and $\beta_1, \ldots, \beta_n \in Lit$ and $n \geq 0$
- $\overline{not\ \alpha} = \alpha$, for each $not\ \alpha \in \mathcal{HB}_{not}$

- Circumscription [McCarthy, 1987] is the framework $\langle T, Ab, \bar{} \rangle$ defined as follows
  - T is a theory in a first-order language $\mathcal{L}$ whose predicate symbols are divided into three sets: P is the set of predicate symbols whose interpretation is to be minimized; Z is the set of predicate symbols who interpretation is to be varied; Q is the set of predicate symbols whose interpretation is to be fixed.
  - $Ab = \mathcal{HB}^{\mathsf{P}}_{\neg} \cup \mathcal{HB}^{\mathsf{Q}}_{\neg} \cup \mathcal{HB}^{\mathsf{Q}}$ where
    - $\mathcal{HB}^{\mathsf{P}}_{\neg}$ is the set of all sentences of the form $\neg p(t_1, ..., t_m)$ with $p \in \mathsf{P}$
    - $\mathcal{HB}^{\mathsf{Q}}_{\neg}$ is the set of all sentences of the form $\neg q(t_1, ..., t_m)$ with $q \in \mathsf{Q}$
    - $\mathcal{HB}^{\mathsf{Q}}$ is the set of all sentences of the form $q(t_1, ..., t_m)$ with $q \in \mathsf{Q}$
  - $\bar{}$ is defined as $\bar{\beta} = \neg\beta$ for $\beta \in \mathcal{L}$.

  In circumscription, the reasoning is done by minimizing the interpretation of the predicates in P while the interpretation of predicates in Z can vary. Sentences that can be derived from a theory T, denoted by $CIRC(T, \mathsf{P}, \mathsf{Z})$, are in the intersections of all (P, Z)-minimal models

of T. The minimization, so called circumscription, corresponds to the maximal conflict-free extension (preferred extension) of $\langle T, Ab, \bar{} \rangle$ in argumentation theory. $CIRC(T, \mathsf{P}, \mathsf{Z})$ corresponds to the intersection of all preferred extensions.

- Default logic [Reiter, 1980] is the framework $\langle T, Ab, \bar{} \rangle$ based on a deductive system $(\mathcal{L}, \mathcal{R})$ defined as follows:
  - $(\mathcal{L}_0, \mathcal{R}_0)$ is a deductive system for classical first-order logic.
  - $\mathcal{L} = \mathcal{L}_0 \cup \{M\alpha | \alpha \in \mathcal{L}_0\}$,
  - $\mathcal{R} = \mathcal{R}_0 \cup D$ where D is a set of default rules of the form

    $$\frac{\alpha : M\beta_1, \ldots, M\beta_n}{\gamma}$$

    where $\alpha, \beta_1, \ldots, \beta_n, \gamma \in \mathcal{L}_0$, and $n \geq 0$
  - $T \subseteq \mathcal{L}_0$
  - $Ab = \{M\beta | \beta \in \mathcal{L}_0$ and $M\beta$ appears in one of the default rules in $D\}$
  - $\overline{M\alpha} = \neg\alpha$
- Autoepistemic logic [Moore, 1985] is the framework $\langle T, Ab, \bar{} \rangle$ based on a deductive system $(\mathcal{L}, \mathcal{R})$ defined as follow
  - $\mathcal{L}$ is a modal language containing a modal operator L
  - $\mathcal{R}$ is some set of inference rules for classical logic for the language $\mathcal{L}$
  - $Ab = \{L\alpha | \alpha \in \mathcal{L}\} \cup \{\neg L\alpha | \alpha \in \mathcal{L}\}$
  - $\overline{\neg L\alpha} = \alpha$ and $\overline{L\alpha} = \neg L\alpha$ for each $\alpha \in \mathcal{L}$
- McDermott's non-monotonic modal logics [McDermott, 1982] are instances of the the framework $\langle T, Ab, \bar{} \rangle$ based on a deductive system $(\mathcal{L}, \mathcal{R})$ defined as follow
  - $\mathcal{L}$ is a modal language containing a modal operator L

- $\mathcal{R}$ is some set of inference rules for classical logic for the language $\mathcal{L}$
- $Ab = \{\neg L\alpha | \alpha \in \mathcal{L}\}$
- $\overline{\neg L\alpha} = \alpha$

With the above assumption-based framework, the similarity and difference between different non-monotonic systems become clear. Along the dimension of what to assume, Theorist allows any subset of the language as assumptions, logic programming takes everything as non-provable by default and makes them as assumptions, default logic takes all the formulas of the form $M\alpha$ ($M\alpha$ is to be read as "it is consistent to assume α") that appear in D as assumptions, autoepistemic logic assumes every $\alpha \in \mathcal{L}$ to be both learnable, $L\alpha \in Ab$, and non-learnable, $\neg L\alpha \in Ab$ [2], and finally non-monotonic modal logic takes every $\alpha \in \mathcal{L}$ as non-learnable, $\neg L\alpha \in Ab$. Along the dimension of what semantics to take, using the above mapping between an assumption-based framework and the elements of the nonmonotonic reasoning systems, the existing semantics of these nonmonotonic systems can be captured by the semantics introduced above for the assumption based framework. For example, the stable, admissible, and preferred semantics of logic programming can be captured as the stable, admissible, and preferred semantics of the assumption-based framework respectively. Similar things happen to the Theorist, default logic, autoepistemic logic and nonmonotonic modal logic as well. In this way, BDKT's argumentation gives us a unified approach to nonmonotonic reasoning and shines light on generalizing them.

7.5 Combination arguments

Definition 7.24 (Combination argument). *An* combination argument *based on* $\mathsf{KL} \subseteq \mathcal{L}$ *is an argument* $A = (H, h)$ *following Definition 7.11 with an additional constraint:*

$$h \stackrel{def}{=} \bigwedge_{h_k \in H} h_k.$$

[2]This approach puts the two controversial sentences $L\alpha$ and $\neg L\alpha$ for any element α in the language into the assumption base without any pre-judgment, then it relies on the semantics to determine which one is acceptable.

In a combination argument, the conclusion is directly the conjunction of all elements in its support. In a set-theoretic argument $A = (H, h)$ following Definition 7.11, it requires that $H \vdash h$. Semantically this means that $\mathcal{I}(H) \subseteq \mathcal{I}(h)$ (recall that $\mathcal{I}$ is the interpretation function of Definition 2.2). Namely, an additional semantic choice is made in a set-theoretic argument to select a superset of the models of its support as a conclusion. On the other hand, a combination argument doesn't make such a choice. Its conclusion fully respects the models of its support without making any decisions on concluding on which superset of models that are entailed by its supporting models. The notion of combination argument will help us precisely conclude the meaning of a specification in the defeasible action theory in terms of the models of the arguments' supports.

For notational convenience, I will overload the notation of *support*. Let A be an argument $\langle H, h\rangle$, $support(A)$ also refer to the following conjunction:

$$support(A) = \bigwedge_{h_k \in H} h_k.$$

Definition 7.25 (Combination defeat). *Let $A = (H_A, h_A)$ and $B = (H_B, h_B)$ be two combination arguments that can be constructed from an knowledge base $\mathsf{KL} \subseteq \mathcal{L}$. A is said to be a* combination-defeat *to B iff*

$$conclusion(A) \wedge conclusion(B) = FALSE.$$

It is straightforward to see from the definition the combination-defeat relation is symmetric.

The conflicts among the arguments that can be picked by defeats of Definition 7.12 depends on the capability of the argument construction mechanism to construct the negated premises, negated material implications of the inference rules, and the negated intermediate conclusions. The more knowledge and inference rules that are available to construct these negated conclusions, the more relevant defeats can be picked up. Combination defeat is a special version of rebut which can pick up all possible conflicts between any two combination arguments.

Definition 7.26 (Combination argumentation framework). *A* combination argumentation framework AFD *can be constructed on a knowledge base* KL*:*

$$\mathsf{AFD} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$$

where

- ARG *is a set of combination arguments constructed from* KL*, and*
- DFT *is a combination defeat relation on* ARG*.*

Definition 7.27 (Combination conflict-free). *A set of arguments* $S \subseteq \mathsf{ARG}$ *is* combination conflict-free *iff*

$$\bigwedge_{A_i \in S} conclusion(A_i) \neq FALSE.$$

Here, an non-standard concept of conflict-free is used. In the literature (e.g. [Dung, 1995]), a conflict-free set of arguments is usually defined as a set of arguments with no defeat relationship between any two arguments in the set. However, this definition is not appropriate in reasoning about the knowledge combination in terms of interpretations. For example, Let $A_1 = \{I_1, I_2\}$, $A_2 = \{I_2, I_3\}$, and $A_3 = \{I_3, I_1\}$. While there is no defeat between any two arguments: $support(A_1) \wedge support(A_2) \neq$ FALSE, $support(A_2) \wedge support(A_1) \neq$ FALSE, and $support(A_1) \wedge support(A_3) \neq$ FALSE, $support(A_1) \wedge support(A_2) \wedge support(A_3) =$ FALSE. In argumentation-based reasoning about state transitions and AI planning, we would like to avoid having a set of conflict-free arguments which have an empty set of interpretations. This motivates us to adopt Definition 7.27 instead of the one in Definition 7.3 (following Dung's abstract framework [Dung, 1995]).

Proposition 7.1 (Combination conflict-free). *In an argumentation framework* $\mathsf{AF} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$*, if a set* $\mathsf{Args} \subseteq \mathsf{ARG}$ *of arguments is combination conflict-free, then* Args *is also conflict-free.*

In an argumentation framework $\mathsf{AF} = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$*, if a set* $\mathsf{Args} \subseteq \mathsf{ARG}$ *of arguments is conflict-free, there are no effective defeats between any two arguments in* Args*.*

Proof. Proof by contradiction. Let Args be a combination conflict-free set of arguments. Assume that there are two arguments $A, B \in$ Args such that A combination-defeats B, we have $support(A) \wedge support(B) =$ FALSE. This implies $\bigwedge_{A_i \in Args} support(A_i) =$ FALSE. We reach a contradiction to that Args is combination conflict-free.

If there are no defeats in Args, then there will no effective defeats in Args as the relation of effective defeats is a sub-relation of defeats.

In this work, I will focus on the notion combination conflict-free. Below, I will refer to combination conflict-free simply as conflict-free unless the two concepts need to be distinguished. All reference to concept of conflict-free in the semantics defined in Section 7.4.1 will be replaced by the concept of combination conflict-free.

7.6 Argumentation for state transitions

Given a knowledge base KL based on an agent specification model (see Definition 5.1), we can construct arguments for state-actions and state transitions. Assuming the Markov property, we can construct state-action arguments for every encountered state and state transitions for every state-action pair.

Definition 7.28 (Agent specification knowledge base). *Let* KL *be an* agent specification knowledge base*, and*

$$A = \langle H, h \rangle$$

be a combination argument (see Definition 7.24).

- *A is called a* state-action argument *if* $support(A) \subseteq SA(\mathsf{KL})$.
- *A is called a* state transition argument *if* $support(A) \subseteq SAS(\mathsf{KL})$.

The set of state-action arguments that can be constructed from KL is denoted by $\mathsf{ARG}_{SA}(\mathsf{KL})$. The set of transition arguments that can be constructed from KL is denoted by $\mathsf{ARG}_{SAS}(\mathsf{KL})$. If the

agent specification knowledge base KL is clear in the context, $\mathsf{ARG}_{SA}(\mathsf{KL})$ and $\mathsf{ARG}_{SAS}(\mathsf{KL})$ are also referred to as ARG_{SA} and ARG_{SAS} respectively.

Definition 7.29 (Compatible argument). *Let ξ be an formula in the language $\mathcal{L}$. An argument A is called an argument compatible with ξ, denoted by ξ-compatible, iff $\xi \wedge support(A) \neq FALSE$.*

As a state s and a set of states S can always be represented by formulae in the language $\mathcal{L}(\mathcal{P}_{SAS})$, we can denote an argument that is compatible with s and S to be s-compatible and S-compatible respectively. Similarly, for a state-action pair $\langle s, a\rangle$ and a set of state-action pairs SA, we denote an argument that is compatible with $\langle s, a\rangle$ and SA to be $\langle s, a\rangle$-compatible and SA-compatible respectively. With this notation, we have the following:

- The set of state-action arguments that are compatible with a state s is denoted by $\mathsf{ARG}_{SA}(s)$.
- The set of state-action arguments that are compatible with a set of states S is denoted by $\mathsf{ARG}_{SA}(S)$.
- The set of transition arguments that are compatible with a state-action pair $\langle s, a\rangle$ is denoted by $\mathsf{ARG}_{SAS}(s, a)$.
- The set of transition arguments that are compatible with a set of state-action pairs SA is denoted by $\mathsf{ARG}_{SAS}(SA)$.

7.7 Preference based argumentation framework for state transitions

Recall that a *preference-based argumentation framework* (see Section 7.2 and Section 7.4.2) is a triple

$$\mathsf{AFP} = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF}\rangle$$

where ARG is a set of arguments, DFT is the binary relation over the arguments, and PREF is a binary preference relation over ARG.

The preference relation PREF over an argumentation space ARG can be derived from the preferences over the knowledge KL on which the arguments are constructed. In order to underpin the reasoning about state transition combinations, I will construct preferences over arguments based on the knowledge structure over the agent specification in Section 5.2 and preferences over subsets of specifications defined in Section 5.6. The structure of knowledge in agent specifications is repeated here for convenience. Each piece of information $spec_{i,k}$ in a knowledge base KL was associated with

- a *specification type*, denoted by $type(spec_{i,k}) \in TYPE$,
- an *specification layer*, denoted by $\mathsf{layer}(spec_{i,k}) \in LAYER$, and
- a *preference level*, denoted by $\mathsf{level}(spec_{i,k}) \in LEVEL$.

The $TYPE$ takes the values of state-action specification (SA) and state transition specification SAS. The $LAYER$ takes the values of interaction (IR), specialization (SPL), operator (OP) and frame axioms (FRM). $LEVEL$ can take any real value in $\Re$. The specification type function type, specification layer function layer, and the preference level function level together are denoted by

$$\mathsf{KM} = \langle \mathsf{type}, \mathsf{layer}, \mathsf{level} \rangle$$

as *meta knowledge* on the knowledge base KL. Correspondingly, the meta information of a piece of information ϕ and a set of information $\mathcal{SPEC}$ is denoted by

$$\mathsf{KM}(\phi) = \langle \mathsf{type}(\phi), \mathsf{layer}(\phi), \mathsf{level}(\phi) \rangle$$

and

$$\mathsf{KM}(\mathcal{SPEC}) = \langle \mathsf{type}(\mathcal{SPEC}), \mathsf{layer}(\mathcal{SPEC}), \mathsf{level}(\mathcal{SPEC}) \rangle.$$

Together, we have a full knowledge base $\mathbf{K}$ for an agent as a pair of agent specification and meta-knowledge

$$\mathbf{K} = \langle \mathsf{KL}, \mathsf{KM} \rangle.$$

Given a set $\mathcal{SPEC}$ of specifications, we can define the minimum level of $\mathcal{SPEC}$ to be

$$minLevel(\mathcal{SPEC}) = \begin{cases} min\{\mathsf{level}(spec) \mid spec \in \mathcal{SPEC}\} & \text{if } \mathcal{SPEC} \neq \emptyset \\ -\infty & \text{if } \mathcal{SPEC} = \emptyset \end{cases}$$

Given a set $\mathcal{SPEC}$ of specifications, we can refer a sub-layer of the specifications by:

- $IR(\mathcal{SPEC}) = \{spec \mid spec \in \mathcal{SPEC} \text{ and } \mathsf{layer}(spec) = IR\}$,
- $SPL(\mathcal{SPEC}) = \{spec \mid spec \in \mathcal{SPEC} \text{ and } \mathsf{layer}(spec) = SPL\}$,
- $OP(\mathcal{SPEC}) = \{spec \mid spec \in \mathcal{SPEC} \text{ and } \mathsf{layer}(spec) = OP\}$, and
- $FRM(\mathcal{SPEC}) = \{spec \mid spec \in \mathcal{SPEC} \text{ and } \mathsf{layer}(spec) = FRM\}$.

We can also refer to a sub-type by:

- $SA(\mathcal{SPEC}) = \{spec \mid spec \in SPEC \text{ and } \mathsf{type}(spec) = SA\}$, and
- $SAS(\mathcal{SPEC}) = \{spec \mid spec \in SPEC \text{ and } \mathsf{type}(spec) = SAS\}$.

To integrate reasoning about valid state-action spaces and valid state transitions together into an argumentation framework, the preference level vector defined in Definition 5.6 is extended to distinguish state-action specifications and transition specifications. On a specification set $\mathcal{SPEC}$, we can also build up a vector measure of preference level as

$$\mathsf{levelVector}(\mathcal{SPEC}) = \langle \mathsf{level}_{ir}, \mathsf{level}_{spl}, \mathsf{level}_{op}, \mathsf{level}_{frm} \rangle$$

where

- $\mathsf{level}_{ir}(\mathcal{SPEC}) = minLevel(IR(\mathcal{SPEC}))$,
- $\mathsf{level}_{spl}(\mathcal{SPEC}) = minLevel(SPL(\mathcal{SPEC}))$,
- $\mathsf{level}_{op}(\mathcal{SPEC}) = minLevel(OP(\mathcal{SPEC}))$, and

- $\mathsf{level}_{frm}(\mathcal{SPEC}) = minLevel(FRM(\mathcal{SPEC}))$.

To argue about state transitions and execution structures in the state transitions, we need a notion of vector concatenation in addition to the level vector definitions introduced in Section 5.2.

Definition 7.30 (Level vector concatenation). *Two level vectors* $\mathbf{lv}_i = \langle l_{i,1}, l_{i,2}, ..., l_{i,m}\rangle$ *and* $\mathbf{lv}_j = \langle l_{j,1}, l_{j,2}, ..., l_{j,n}\rangle$ *can be concatenated into a new level vector, denoted by* $\mathbf{lv}_i \oplus \mathbf{lv}_j$*:*

$$\mathbf{lv}_i \oplus \mathbf{lv}_j = \langle l_{i,1}, l_{i,2}, ..., l_{i,m}, l_{j,1}, l_{j,2}, ..., l_{j,n}\rangle$$

For example,

$$\begin{aligned}\mathsf{level}_{SA,SAS}(\mathcal{SPEC}) =& \mathsf{level}_{\mathsf{layer}}(SA(\mathcal{SPEC})) \oplus \mathsf{level}_{\mathsf{layer}}(SAS(\mathcal{SPEC}))\\ =& \langle \mathsf{level}_{SA,ir}, \mathsf{level}_{SA,spl}, \mathsf{level}_{SA,op}, \mathsf{level}_{SA,frm},\\ & \mathsf{level}_{SAS,ir}, \mathsf{level}_{SAS,spl}, \mathsf{level}_{SAS,op}, \mathsf{level}_{SAS,frm}\rangle\end{aligned}$$

where

- $\mathsf{level}_{SA,ir}(\mathcal{SPEC}) = minLevel(IR(SA(\mathcal{SPEC})))$,
- $\mathsf{level}_{SA,spl}(\mathcal{SPEC}) = minLevel(SPL(SA(\mathcal{SPEC})))$,
- $\mathsf{level}_{SA,op}(\mathcal{SPEC}) = minLevel(OP(SA(\mathcal{SPEC})))$,
- $\mathsf{level}_{SA,frm}(\mathcal{SPEC}) = minLevel(FRM(SA(\mathcal{SPEC})))$,
- $\mathsf{level}_{SAS,ir}(\mathcal{SPEC}) = minLevel(IR(SAS(\mathcal{SPEC})))$,
- $\mathsf{level}_{SAS,spl}(\mathcal{SPEC}) = minLevel(SPL(SAS(\mathcal{SPEC})))$,
- $\mathsf{level}_{SAS,op}(\mathcal{SPEC}) = minLevel(OP(SAS(\mathcal{SPEC})))$,
- $\mathsf{level}_{SAS,frm}(\mathcal{SPEC}) = minLevel(FRM(SAS(\mathcal{SPEC})))$.

Here the state-action specifications take a higher significant order than the state transition specifications to reflect that state transitions should sanction the valid state-action specifications if such specifications exist.

In Definition 5.8, we further define a cascading preference relation over two sets $SPEC_i$ and $SPEC_j$ of specifications as

$$SPEC_i \succ SPEC_j \text{ iff } \mathsf{level}(SPEC_i) \succ \mathsf{level}(SPEC_j).$$

Two vectors of levels $\mathbf{lv}_i = \langle l_{i,1}, l_{i,2}, ..., l_{i,n}\rangle$ and $\mathbf{lv}_j = \langle l_{j,1}, l_{j,2}, ..., l_{j,n}\rangle$ can be compared in a cascading manner (see Definition 5.7): $\mathbf{lv}_i \succ \mathbf{lv}_j$ iff

- $\langle l_{i,n}\rangle \succ \langle l_{j,n}\rangle$ iff $l_{i,n} > l_{j,n}$,
- $\langle l_{i,k}, l_{i,k+1}, ..., l_{i,n}\rangle \succ \langle l_{j,k}, l_{j,k+1}, ..., l_{j,n}\rangle$ iff
  - $l_{i,k} > l_{j,k}$ or,
  - $l_{i,k} = l_{j,k}$ and $\langle l_{i,k+1}, ..., l_{i,n}\rangle \succ \langle l_{j,k+1}, ..., l_{j,n}\rangle$.

Now we are able to compare preferences over two arguments.

Definition 7.31 (Cascading preference). *Let* KL *be a knowledge base of agent specification (see Definition 5.1). Let two arguments be* $A_1 = (H_1, h_1)$ *and* $A_2 = (H_2, h_2)$ *on a knowledge base* KL. *A preference relation* PREF *over arguments can be defined as*

$$(A_1, A_2) \in \mathsf{PREF} \textit{ iff } \mathsf{level}(H_1) \succ \mathsf{level}(H_2).$$

For notational consistency, $(A_1, A_2) \in \mathsf{PREF}$ *is also denoted by* $A_1 \succ A_2$.

As $\succ$ is asymmetric, it easy to see that PREF is also asymmetric.

Definition 7.32 (Preference-refined defeat relation). *Let* PREF *be a preference relation and* DFT *be a defeat relation on a set* ARG *of arguments. A* preference-refined defeat relation PDFT

can be defined as the following for any two arguments A_1 *and* A_2 *in* ARG

$$(A_1, A_2) \in \mathsf{PDFT} \textit{ iff } (A_1, A_2) \in \mathsf{DFT} \textit{ but } (A_2, A_1) \notin \mathsf{PREF}.$$

With the concept of preference-refined defeat relation PDFT, we can translate a preference-based argumentation framework $\mathsf{AFP} = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$ into an argumentation framework

$$\mathsf{AFD}(\mathsf{AFP}) = \langle \mathsf{ARG}, \mathsf{PDFT} \rangle.$$

7.8 Sub-arguments and Super-arguments

Definition 7.33 (Combination sub-argument). *A combination argument* A *is said to be a* combination sub-argument *of another combination argument* B *iff* $support(A) \subseteq support(B)$*. Conversely,* B *is said to be a* combination super-argument *of* A*.*

In this work, as we will focus on combination sub-arguments and combination super-arguments, combination sub-arguments and combination super-arguments will be referred to simply as sub-arguments and super-arguments respectively.

Lemma 7.1 (Sub-argument conflict-free). *Given an argument* A *and a set* S *of combination conflict-free arguments, if all sub-arguments of* A *are in* S*, then* $A \cup \{A\}$ *is also conflict-free.*

Proof. It is obvious that $support(A) \subseteq \cup_{A_i \in S} support(A_i)$. $\bigwedge_{A_i \in S} support(A_i) \neq$ FALSE implies $\bigwedge_{A_i \in S} support(A_i) \wedge support(A) \neq$ FALSE. Therefore $S \cup \{A\}$ is also conflict-free.

Proposition 7.2 (Super argument conflict-free). *If a set* S *of arguments is combination conflict-free, then the set of arguments whose sub-arguments are in* S *is also combination conflict-free. Formally*

$$\{A \mid A \in S \textit{ or all sub-arguments of } A \textit{ is in } S\}$$

is combination conflict-free.

Proof. Directly derived from Lemma 7.1.

Proposition 7.3 (Sub-argument admissible extension). *Let S be an admissible extension of a combination argumentation framework $\mathsf{AF} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$ based on a knowledge base KL, then*

$$\underline{S} = S \cup \{A \mid \textit{there exists some } B \in S \textit{ such that } support(A) \subseteq support(B)\}$$

is also an admissible extension of AF.

Proof. Let A be any argument in $\underline{S}$. If $A \in S$, then as S is an admissible extension, if there is an argument $B \in \mathsf{ARG}$ such that $(B, A) \in \mathsf{DFT}$, there must be some argument in S which can defend A against B. If $A \notin S$, then there must be an argument $D \in S$ such that A is a sub-argument of D. For any argument $E \in \mathsf{ARG}$ such that $(E, A) \in \mathsf{DFT}$, namely $support(E) \wedge support(A) = \text{FALSE}$, as $support(A) \subseteq support(D)$, it must be the case that $support(E) \wedge support(D) = \text{FALSE}$, namely $(E, D) \in \mathsf{DFT}$. As S is an admissible extension, there must be some argument in S which can defend D against E. Such an argument also defends A against E. This concludes that $\underline{S}$ is also an admissible extension.

Proposition 7.4 (Preference sub-argument admissible extension). *Let S be an admissible extension of a preference-based combination argumentation framework $\mathsf{AFP} = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$ based on a knowledge base KL, then*

$$\underline{S} = S \cup \{A \mid \textit{there exists some } B \in S \textit{ such that } support(A) \subseteq support(B)\}$$

is also an admissible extension of AFP.

Proof. Let A be any argument in $\underline{S}$. If $A \in S$, then as S is an admissible extension, if there is an argument $B \in \mathsf{ARG}$ such that $(B, A) \in \mathsf{PDFT}$, there must be some argument in S which can defend A against B. If $A \notin S$, then there must be an argument $D \in S$ such that A is a sub-argument of D. For any argument $E \in \mathsf{ARG}$ such that $(E, A) \in \mathsf{PDFT}$, then it must be the case that $(E, A) \in \mathsf{DFT}$

(see Definition 7.32). Since D is a super-argument of A, it is also the case that $(E, D) \in \mathsf{DFT}$. As S is an admissible extension, if $(E, D) \in \mathsf{PDFT}$, then there must be some argument F in S which can defend D against E. Such an argument also defends A against E. If $(E, D) \notin \mathsf{PDFT}$, then it must be the case that $(D, E) \in \mathsf{PREF}$. Since PREF is asymmetric, $(E, D) \notin \mathsf{PREF}$. As $(D, E) \in \mathsf{DFT}$ and $(E, D) \notin \mathsf{PREF}$, we have $(D, E) \in \mathsf{PDFT}$, namely D defends A against E. Thus we conclude that $\underline{S}$ is also an admissible extension of AFP.

Proposition 7.3 and Proposition 7.4 give us a hint as to how we go about computing a preferred extension of an combination argumentation framework and its preference-based extension. We can compute an admissible extension which contains only one argument with maximal consistent set of supports in KL.

Proposition 7.5 (Sub-argument preferred extension). *Let* $\mathsf{AFD} = \langle \mathsf{ARG}, \mathsf{DFT} \rangle$ *be a combination argumentation framework based on a knowledge base* KL. *Let* $A = (H, h)$ *be a combination argument with maximally consistent subset of supports. If* $\{A\}$ *is an admissible extension of* AFD*, then all the sub-arguments of* A *form a preferred extension.*

Proof. Let S be the set of combination arguments whose supports are a subset of H. As $\{A\}$ is an admissible extension, by Proposition 7.3, S is also an admissible extension. Since there is no $H' \subseteq \mathsf{KL}$ such that $H \subseteq H'$ and H' is consistent. There will be no argument B in ARG such that $B \notin S$ and $support(A) \wedge support(B) \neq \text{FALSE}$. Namely, S is a maximally conflict-free extension of AFD.

Proposition 7.6 (Preference-based sub-argument preferred extension).
Let $\mathsf{AFP} = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$ *be a combination preference-based argumentation framework based on a knowledge base* KL. *Let* $A = (H, h)$ *be a combination argument with maximally consistent subset of supports. If* $\{A\}$ *is an admissible extension of* AFD*, then all the sub-arguments of* A *form a preferred extension.*

Proof. Let S be the set of combination arguments whose supports are a subset of H. As $\{A\}$ is an admissible extension, by Proposition 7.4, S is also an admissible extension. Since there is no $H' \subseteq \mathsf{KL}$ such that $H \subseteq H'$ and H' is consistent. There will be no argument B in ARG such that $B \notin S$ and $support(A) \wedge support(B) \neq \text{FALSE}$. Namely, S is a maximally conflict-free extension of AFD.

7.9 Implicitly arguing for all states simultaneously

Definition 7.34 (Preference-based state-action argumentation framework). *Given an agent specification knowledge base* KL*, a preference-based state-action argumentation framework* $\mathsf{AFP}_{SA}(\mathsf{KL})$ *is defined as*

$$\mathsf{AFP}_{SA}(\mathsf{KL}) = \langle \mathsf{ARG}_{SA}, \mathsf{DFT}_{SA}, \mathsf{PREF}_{SA} \rangle$$

where

- ARG_{SA} *is a set of state-action arguments following Definition 7.28.*
- DFT_{SA} *is a combination defeat relation that can be built on* ARG_{SA}*.*
- PREF_{SA} *is a preference relation on* ARG_{SA} *following Definition 7.32.*

Given a state s, an s-compatible preference-based state-action argumentation framework $\mathsf{AFP}_{SA}(\mathsf{KL}, s) = \langle \mathsf{ARG}_{SA}, \mathsf{DFT}_{SA}, \mathsf{PREF}_{SA} \rangle$ *is a preference-based state-action argumentation framework on* KL *with an additional constrain on* ARG_{SA} *being a set of s-compatible state-action arguments.*

Definition 7.35 (Preference-based state transition argumentation framework). *Given an agent specification knowledge base* KL*, a preference-based state-action argumentation framework* $\mathsf{AFP}_{SAS}(\mathsf{KL})$ *is defined as*

$$\mathsf{AFP}_{SAS}(\mathsf{KL}) = \langle \mathsf{ARG}_{SAS}, \mathsf{DFT}_{SAS}, \mathsf{PREF}_{SAS} \rangle$$

where

- ARG_{SAS} *is a set of state transition arguments following Definition 7.28.*
- DFT_{SAS} *is a combination defeat relation that can be built on* ARG_{SA}*.*
- PREF_{SAS} *is a preference relation on* ARG_{SAS} *following Definition 7.32.*

Given a state-action pair $\langle s, a\rangle$*, an* $\langle s, a\rangle$*-compatible preference-based state-action argumentation framework* $\mathsf{AFP}_{SA}(\mathsf{KL}, s, a) = \langle \mathsf{ARG}_{SAS}, \mathsf{DFT}_{SAS}, \mathsf{PREF}_{SAS}\rangle$ *is a preference-based state transition argumentation framework on* KL *with an additional constraint on* ARG_{SAS} *being a set of* $\langle s, a\rangle$*-compatible state-action arguments.*

The algorithms introduced in Section 5.8 for computing valid state-action spaces and valid state transitions can simultaneously compute the preferred extensions of the preference-based state-action argumentation frameworks on every state and transition argumentation frameworks on every state-action pair. The internal hash data structures used to manipulate BDDs are then a mechanism for all these argumentation computation to share as much reasoning as possible.

Proposition 7.7 (State-action argumentation). *Let* KL *be an agent specification.*

- *For every state* $s \in 2^{\mathcal{P}_S}$ *such that there exist* s*-compatible state-action arguments constructed from* KL*,* $computeSA(\mathsf{KL})$ *(see Algorithm 12 on page 126) computes the preferred extensions for the preference-based state-action argumentation framework that is compatible with* s*.*
- *For every state* $s \in 2^{\mathcal{P}_S}$ *such that there exist* s*-compatible state-action arguments constructed from* KL*,* $computeSA(\mathsf{KL})$ *(see Algorithm 12 on page 126) returns the set of all possible state-actions on* s*.*

Proof. Algorithm 12 starts by computing the maximally consistent state-action arguments in line 2. Then, in line 3, $computePreferredCombine(\mathcal{P}_S, maximalSA)$ (see Algorithm 11 on page 124) is called to remove the arguments that can be defeated with respect to PDFT_{SA} on each state. In this way, for every state s, only the maximally consistent arguments that can defend themselves will be preserved. Thereafter, in line 5 the algorithm makes an open-world assumption by

allowing any state-action in the states where no state-action arguments can be constructed by $SA \leftarrow specifiedSA \vee (\text{TRUE} - specifiedSA^{\downarrow \mathcal{P}_S})$.

Proposition 7.8 (State transition argumentation). *Let* KL *be an agent specification. For every state-action pair* $\langle s, a \rangle \in 2^{\mathcal{P}_{SA}}$ *such that*

- $\langle s, a \rangle$ *is compatible with any maximally preferred argument computed in* $computeSA(\mathsf{KL})$ *(see Algorithm 12 on page 126) on* s*, or*
- *there is no* s*-compatible state-action arguments that can be constructed on* KL

$computeSAS(\mathsf{KL})$ *(see Algorithm 13 on page 127) computes the preferred extensions for the preference-based state transition argumentation framework that is based on* KL *and is compatible with* $\langle s, a \rangle$*.*

Proof. Algorithm 12 starts by computing the valid state-action space in line 1 (see Algorithm 11 on page 124), and computing the maximally consistent state-action arguments with the state transition specifications in KL that are not frame-axioms. For every state-action pair, line 3 (see Algorithm 11 on page 124) computes the maximally consistent state transition arguments. Then $computePreferredCombine(\mathcal{P}_S, maximalSA)$ (see Algorithm 11, line 4) is invoked to remove the arguments that can be defeated with respect to PDFT_{SAS} on each state-action pair. In this way, for every state $\langle s, a \rangle$, only the maximally consistent arguments that can defend themselves will be preserved. Thereafter, in loop starting by line 6, the algorithm combines the frame axioms that are consistent with the maximal state-transition arguments computed in line 3 one-by-one. As any two frame axioms are consistent with each other, the order in which the frame axioms are added into the maximally consistent state transition arguments doesn't matter. This guarantees that, after the loop started by line 3 the maximally consistent state transition arguments also contains maximally consistent frame axioms. This in turn guarantees that the state transition arguments computed for every valid state-action pair are maximal arguments with frame axioms taken into account.

Note that not taking the frame axioms into account during the computation of maximally state transition arguments but having an extra loop to combine the frame axioms is an efficiency consideration. We can have the algorithm consider the frame axioms during the computation of maximal arguments in line 3 and remove the loop in line 6. This will output the same result.

Proposition 7.9 (Joint state transition argumentation). *Let* KL *be an agent specification for a multiagent system* $\mathsf{AGS} = \{\mathsf{Ag}_1, ..., \mathsf{Ag}_N\}$. *For every joint state-action pair* $\langle s, a\rangle \in 2^{\mathcal{P}_{SA}}$ *such that*

- $\langle s, a\rangle$ *is compatible with any maximally preferred argument computed in* $computeSA(\mathsf{KL})$ *(see Algorithm 12 on page 126) on* s*, or*
- *there is no* s*-compatible state-action arguments that can be constructed on* KL

$computeJointSAS(\mathsf{KL}, \mathsf{AGS})$ *(see Algorithm 14 on page 128) computes the preferred extensions for the preference-based state transition argumentation framework that is*

- *based on* KL*,*
- *compatible with* $\langle s, a\rangle$*, and*
- *compatible with the criteria that every agent can act in each joint transition.*

Proof. Algorithm 14 calls $computeSAS(\mathsf{KL})$ (see Algorithm 13 on page 127). From Proposition 7.8, we know that the output is the set of preferred extensions of the preference-based state transition argumentation framework compatible with each $\langle s, a\rangle$ with the properties stated in the corollary. The loop starting with line 3 of Algorithm 14 guarantees that every agent will act according to its specification.

7.10 Arguing over the frame of discernment

Algorithm 12 (on page 126), Algorithm 13 (on page 127) and Algorithm 14 (on page 128) implicitly compute the arguments for every state, state-action, and joint state transition. The usage of implicit

argumentation is for algorithmic and efficiently considerations. Now we are ready to capture the idea of arguing for every state and every state-action pair together into the argumentation framework implicitly through the concept of frame of discernment arguments.

Definition 7.36 (Frame arguments). *A* frame argument *on a set $\mathcal{P}$ of proposition variables is a combination argument A where the support encodes exactly one truth assignment for $\mathcal{P}$:*

$$support(A) = \{\bigwedge_{p_i \in \mathcal{P}, f_i \text{ is either } p_i \text{ or } \neg p_i} f_i\}.$$

Definition 7.37 (State frame argument). *A* state frame argument *on a set $\mathcal{P}_S$ of state variables is a frame argument on $\mathcal{P}_S$.*

Definition 7.38 (State-action frame argument). *A* state-action frame argument *on a set $\mathcal{P}_{SA}$ of state variables is a frame argument on $\mathcal{P}_{SA}$.*

Definition 7.39 (Preference-based frame state-action argumentation framework). *Given an agent specification* KL*, a preference-based state-action argumentation framework extended with frame of state discernment is a state-action argumentation framework:*

$$\mathsf{AFP}_{SA}(\mathsf{KL}) = \langle \mathsf{ARG}_{SA}, \mathsf{DFT}_{SA}, \mathsf{PREF}_{SA} \rangle$$

where

- ARG_{SA} *is a set of state-action argumentsextended with state frame arguments.*
- DFT_{SA} *is a combination defeat relation that can be built on* ARG_{SA}*.*
- PREF_{SA} *is a preference relation on* ARG_{SA} *following Definition 7.32.*

Definition 7.40 (Preference-based frame state transition argumentation framework). *Given an agent specification* KL*, a preference-based state transition argumentation framework extended with frame of state-action discernment is a preference-based state-action argumentation*

framework:

$$\mathsf{AFP}_{SAS}(\mathsf{KL}) = \langle \mathsf{ARG}_{SAS}, \mathsf{DFT}_{SAS}, \mathsf{PREF}_{SAS} \rangle$$

where

- ARG_{SAS} *is a set of state transition arguments extended with state-action frame arguments.*
- DFT_{SAS} *is a combination defeat relation that can be built on* ARG_{SA}*.*
- PREF_{SAS} *is a preference relation on* ARG_{SAS}*.*

Proposition 7.10 (Computing preferred extensions of state-action argumentation). *Let* KL *be an agent specification knowledge and*

$$\mathsf{AFP}_{SA}(\mathsf{KL}) = \langle \mathsf{ARG}_{SA}, \mathsf{DFT}_{SA}, \mathsf{PREF}_{SA} \rangle$$

be an preference-based argumentation framework for state-actions extended with frame of state discernment. $computeSA(\mathsf{KL})$ *(see Algorithm 12 on page 126) computes the set of preferred extensions for* $\mathsf{AFP}_{SA}(\mathsf{KL})$*.*

Proof. Following directly from the definition of preference-based frame state-action argumentation framework (Definition 7.39) and the proof of Proposition 7.7.

Proposition 7.11 (Computing preferred extensions of state transition argumentation). *Let* KL *be an agent specification and*

$$\mathsf{AFP}_{SAS}(\mathsf{KL}) = \langle \mathsf{ARG}_{SAS}, \mathsf{DFT}_{SAS}, \mathsf{PREF}_{SAS} \rangle$$

be an preference-based argumentation framework for state-actions extended with frame of state-action discernment. $computeSAS(\mathsf{KL})$ *(see Algorithm 13 on page 127) computes the set of preferred extensions for* $\mathsf{AFP}_{SAS}(\mathsf{KL})$ *excluding the preferred extensions with no state transition arguments.*

Proof. Following directly from the definition of preference-based frame state transition argumentation framework (Definition 7.40) and the proof of Proposition 7.8.

Proposition 7.12 (Preferred extensions of the joint state transition argumentation). *Let* KL *be an agent specification and*

$$\mathsf{AFP}_{JSAS}(\mathsf{KL}) = \langle \mathsf{ARG}_{JSAS}, \mathsf{DFT}_{JSAS}, \mathsf{PREF}_{JSAS} \rangle$$

be an preference-based argumentation framework for state-actions extended with frame of joint state-action discernment. compute$JSAS$(KL) (see Algorithm 14 on page 128) computes the set of preferred extensions for $\mathsf{AFP}_{JSAS}(\mathsf{KL})$ *excluding the preferred extensions with no state transition arguments.*

Proof. Following directly from the definition of preference-based frame joint state transition argumentation framework (Definition 7.40) and the proof of Proposition 7.9.

7.11 Process arguments

Now we are ready to extend argumentation based reasoning for one step state transitions to argumentation-based reasoning for an execution structure. The extension will effectively propagate argumentation through the underlying state transition dynamics.

An execution path *exec* (see Definition 3.1) of n action steps in a state space $\mathcal{R}$ is a sequence of state-action pairs ending with a final state. We can capture an execution path as an argument by having n copies of the action domains: $\mathcal{P}_{0,A}, ..., \mathcal{P}_{n-1,A}$ and $n+1$ copies of the state domains: $\mathcal{P}_{0,S}, ..., \mathcal{P}_{n,S}$. With these domains, we can have a domain of process with length n:

$$\mathcal{P}_{process}(\overline{n}) \stackrel{\text{def}}{=} \mathcal{P}_{0,S} \cup \mathcal{P}_{0,A} \cup \mathcal{P}_{1,S} \cup ... \cup \mathcal{P}_{n-1,A} \cup \mathcal{P}_{n,S}$$

which creates structure for recording states and actions of policy executions.

An agent specification KL can be extended into a *forward process knowledge base* $\mathsf{KL}_{forward}(\overline{n})$ or a *backward process knowledge base* $\mathsf{KL}_{backward}(\overline{n})$ of length n:

$$\mathsf{KL}_{forward}(k) = \mathsf{KL}[\mathcal{P}_S/\mathcal{P}_{k,S}, \mathcal{P}_A/\mathcal{P}_{k,A}, \mathcal{P}_{S'}/\mathcal{P}_{k+1,S}]$$
$$\mathsf{KL}_{forward}(\overline{n}) = \cup_{k=0,...,n-1}\mathsf{KL}(k)$$
$$\mathsf{KL}_{backward}(k) = \mathsf{KL}[\mathcal{P}_S/\mathcal{P}_{k+1,S}, \mathcal{P}_A/\mathcal{P}_{k,A}, \mathcal{P}_{S'}/\mathcal{P}_{k,S}]$$
$$\mathsf{KL}_{backward}(\overline{n}) = \cup_{k=0,...,n-1}\mathsf{KL}(k)$$

Derived from the same agent specification, the difference between a forward process knowledge base and a backward process knowledge base is in the way it duplicates the state domain variables. In a forward process knowledge base, as the number of the state domains $\mathcal{P}_{k,S}$ and action domains $\mathcal{P}_A$ increase from the initial states towards goal states, one would expect given that the above definition of forward process knowledge base is in the forward chaining style. In a backward process knowledge base, the kth segment of the knowledge is obtained by copying the next state domain variables $\mathcal{P}_{S'}$ into $\mathcal{P}_{k,S}$ and the current state domain variables $\mathcal{P}_S$ are copied into $\mathcal{P}_{k+1,S}$.

Definition 7.41 (Forward process argument). *A* forward process argument *of length n is a combination argument*

$$A = \langle H, h \rangle$$

such that

- *it is based on a forward process knowledge base $\mathsf{KL}_{forward}(\overline{n})$, and*
- *it contains at least one piece of knowledge for every step, namely $support(A) \cap \mathsf{KL}_{forward}(k) \neq \emptyset$ for every $k = 0, ..., n-1$.*

n is called the process length of argument A, denoted by $length(A)$.

To match the forward argument process, we can have the following backward arguments for processes.

Definition 7.42 (Backward process argument). *A* backward process argument *of length n is a combination argument*

$$A = \langle H, h \rangle$$

such that

- *it is based on a backward process knowledge base $\mathsf{KL}_{backward}(\overline{n})$, and*
- *it contains at least one piece of knowledge for every step, namely $support(A) \cap \mathsf{KL}(k) \neq \emptyset$ for every $k = 0, ..., n-1$.*

n is called the process length of argument A, denoted by $length(A)$.

The sets of forward and backward arguments are denoted by $\mathsf{ARG}_{process,forward}$ and $\mathsf{ARG}_{process,backward}$ respectively. Note that the forward process arguments and backward process arguments represent exactly the same set of processes but number symbols differently.

As the planning algorithms (such as Algorithm 1 on page 67) I use are in a backward chaining style, I will use the following notions

$$\mathsf{KL}(\overline{n}) = \mathsf{KL}_{backward}(\overline{n})$$

$$\mathsf{ARG}_{process} = \mathsf{ARG}_{backward,process}$$

to refer to the process knowledge and process arguments. The set of arguments for all processes of length k is defined as

$$\mathsf{ARG}_{process}(\mathsf{KL}, k) = \mathsf{ARG}_{process}(\mathsf{KL}, k).$$

The set of arguments for all processes of length up to n is defined as

$$\mathsf{ARG}_{process}(\mathsf{KL}, \overline{n}) = \mathsf{ARG}_{process}(\mathsf{KL}, 1) \cup \mathsf{ARG}_{process}(\mathsf{KL}, 2) \cup ... \cup \mathsf{ARG}_{process}(\mathsf{KL}, n).$$

If forward and backward style process arguments need to be distinguished, we denote forward

process arguments of length less than n to be $\mathsf{ARG}_{process,forward}(\mathsf{KL}, \overline{n})$, and backward process arguments of length less than n to be $\mathsf{ARG}_{process,backward}(\mathsf{KL}, \overline{n})$. If the process length is not known ahead, we have $\mathsf{ARG}_{process}(\mathsf{KL})$ denote the set of all finite step process arguments of concern.

Definition 7.43 (Process defeat). *Let A and B be two process arguments of $\mathsf{ARG}_{process}$ from an agent specification KL. A* process defeat relation $\mathsf{DFT}_{process}$ *is defined as: $(A, B) \in \mathsf{DFT}_{process}$ iff $support(A) \wedge support(B) = FALSE$ following the definition of combination defeat (see Definition 7.25).*

Definition 7.44 (Process step projection). *Let $A = \langle H, h \rangle$ be a process argument of length n, then the k-step projection argument ($k < n$) of A is*

$$A^{\downarrow k} = \langle H^{\downarrow k}, h^{\downarrow k} \rangle$$

where

- $H^{\downarrow k} = \{spec^{\downarrow \mathcal{P}_{k,S} \cup \mathcal{P}_{k,A} \cup \mathcal{P}_{k+1,S}} \mid spec \in H\}$, *and*
- $h = \bigwedge_{spec \in H^{\downarrow k}} spec$.

Corresponding to the re-interpretation of an agent specification knowledge KL into every step to be $\mathsf{KL}(k)$, the meta-knowledge KM of specification types, layers and levels (see Section 7.7) on KL will also re-interpretation into every step to be $\mathsf{KM}(k)$.

Definition 7.45 (Process argument preference). *Let A and B be two process arguments. A is preferred to B, denoted by $(A, B) \in \mathsf{PREF}_{process}$ iff for every $k = 0, ..., min(length(A), length(B))$ $(A^{\downarrow k}, B^{\downarrow k}) \in \mathsf{PREF}$ according to the meta-knowledge in $\mathsf{KM}(k)$.*

The process preference $\mathsf{PREF}_{process}$ enables a process argument defend against another process argument only when it can defend itself in every step.

Definition 7.46 (Process argumentation framework). *Let* KL *be an agent specification knowledge base and* KM *be its meta-knowledge. A preference-based argumentation framework* $\mathsf{AFP}_{process}(n)$ *for processes up to length* n *can be defined as*

$$\mathsf{AFP}_{process}(n) = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$$

where

- ARG *is the set of combination arguments that can be constructed on the process knowledge base* $\mathsf{KL}(\overline{n})$ *of length* n*;*
- DFT *is the combination defeat relation on* ARG*;*
- PREF *is the union of the process preference* $\mathsf{PREF}_{process}$ *and the preference on* ARG *derived from* $\mathsf{KM}(k)$ *for every step* $k = 0, ..., n-1$*.*

To frame argumentation over processes, we need to define arguments for state sequence frame and arguments for state-action sequence frames as the following.

Definition 7.47 (State sequence frame argument). *A* state sequence frame argument *of length* $n-1$ *is a frame argument on a sequence of state domains* $\langle \mathcal{P}_{0,S}, \mathcal{P}_{1,S}, ..., \mathcal{P}_{n-1,S} \rangle$ *is a frame argument on the variable set* $\mathcal{P}_{0,S} \cup \mathcal{P}_{1,S} \cup ... \cup \mathcal{P}_{n-1,S}$*.*

Definition 7.48 (State-action sequence frame argument). *A* state-action sequence frame argument *of length* $n-1$ *is a frame argument on a sequence of state-action domains* $\langle \mathcal{P}_{0,S}, \mathcal{P}_{0,A}, \mathcal{P}_{1,S}, \mathcal{P}_{1,A}, ..., \mathcal{P}_{n-1,S}, \mathcal{P}_{n-1,A}, \rangle$ *is a frame argument on the variable set* $\mathcal{P}_{0,S} \cup \mathcal{P}_{0,A} \cup \mathcal{P}_{1,S} \cup \mathcal{P}_{1,A} \cup ... \cup \mathcal{P}_{n-1,S} \cup \mathcal{P}_{n-1,A}$*.*

Definition 7.49 (Preference-based process argumentation framework). *Given an agent specification* KL*, a* preference-based process argumentation framework *extended with state-action se-*

quence frame is a preference-based process argumentation framework:

$$\mathsf{AFP}_{process}(\mathsf{KL}, n) = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$$

where

- ARG *is the set of combination arguments that can be constructed on the process knowledge base* $\mathsf{KL}(\overline{n})$ *of length* n *derived from* $myKL$ *extended with state sequence frame arguments and state-action sequence frame arguments of length* $n-1$.
- DFT *is a combination defeat relation that can be built on* ARG.
- PREF *is the union of the process preference* $\mathsf{PREF}_{process}$ *and the preference on* ARG *derived from* $\mathsf{KM}(k)$ *for every step* $k = 0, ..., n-1$.

Proposition 7.13 (Preferred extensions of process argumentation framework). *Let* S *be a preferred extension of a process argumentation framework* $\mathsf{AFP}_{process} = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$ *extended with arguments of state sequence and state-action sequence discernment frame, then*

- *each preferred extension of* $\mathsf{AFP}_{process}$ *with least one process argument corresponds to an execution path conforming to the argumentation for state transitions on every step;*
- *the set of preferred extensions of* $\mathsf{AFP}_{process}$ *each of which contains at least one process argument corresponding to the set of all possible execution paths conforming to the argumentation for state transitions on every step.*

Proof. It is obvious that the state sequence frame argument of length $n-1$ enables a sequence of state-action argumentation on every step and preserves the set of valid state-actions in the preferred extensions on the state of every step. The state-action sequence frame argument of length $n-1$ enables a sequence of state transition argumentation on every decision step (every state-action pair in the sequence) and preserves the set of valid state transitions in the preferred

extensions on the state-action of every step. From these observations, a preferred extension in the process argumentation framework with least one process argument corresponds to an execution path conforming to the argumentation for state transitions on every step. Therefore, the set of preferred extensions of $\mathsf{AFP}_{process}$ each of which contains at least one process argument corresponds to the set of all possible execution paths conforming to the argumentation for state transitions on every step.

7.12 Policy argumentation framework

Policy arguments can be produced in the argument space to extract the set of acceptable arguments with respect to the initial and goal states, as well as with respect to the execution capability of the agents.

Definition 7.50 (Policy argument). *A* policy argument *is a combination argument*

$$A_{policy} = \langle H, h \rangle.$$

where $H \subseteq \mathcal{L}$ *whose domain is the state and action variables* $\mathcal{P}_{SA}$.

A policy argument can be cast into the process argument space $\mathsf{ARG}(n)$ of length n by renaming the state and action domain variables into each segment of the process state and action argument space:

$$A_{policy}(n) = \bigwedge_{k=1,\ldots,n} A_{policy}[\mathcal{P}_{SA}/\mathcal{P}_{S,k} \cup \mathcal{P}_{A,k}].$$

Definition 7.51 (Execution argumentation framework). *Let* KL *be an agent specification,* KM *be its meta-knowledge and* A_π *be a policy argument of a policy* π *(whose process re-interpretation up to* n *step is* $A_\pi(n)$*). An execution argumentation framework* $\mathsf{AFP}_{exec}(n)$ *for processes up to length* n *can be defined as*

$$\mathsf{AFP}_{exec}(\pi, \overline{n}) = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$$

where

- ARG *is* $\{A_\pi(n)\} \cup \mathsf{ARG}_{process}(\overline{n})$ *(the set of combination arguments that can be constructed on the process knowledge base* $\mathsf{KL}(\overline{n})$*), and;*
- DFT*, the combination defeat relation on* ARG*;*
- PREF*, the union of the process preference* $\mathsf{PREF}_{process}$ *and the preference on* ARG *derived from* $\mathsf{KM}(k)$ *for every step* $k = 0, ..., n-1$*.*

Proposition 7.14 (Execution argumentation preferred extensions). *Given an execution argumentation framework* $\mathsf{AFP}_{exec}(\pi, \overline{n})$ *which is based on an agent specification knowledge base* KL*, its meta-knowledge* KM*, and a policy argument* A_π*, the preferred extensions of* $\mathsf{AFP}_{exec}(n)$ *correspond to the execution structures that conform to the state transition relations extracted from preferred extensions of the state transition argumentation framework* $\mathsf{AFP}_{sas}(\mathsf{KL})$ *(see Proposition 7.11).*

Proof. By construction.

7.13 Solution argumentation

To incorporate argumentation for processes with initial and goal state specification, we will extend the agent specification knowledge base with a *situation base* KI — a knowledge base which gives the initial states that the agents are believed to be in —, and a *goal base* KG — a knowledge base which gives the goals that the agents are designated to achieve.

Definition 7.52 (Situation base). *For an agent system* AGS*, a set of specification* $\mathsf{KI} \subseteq \mathcal{L}$ *using only state variables* $\mathcal{P}_S$ *is called a* situation base *to specify which initial states an agent system* AGS *is believed to be in.*

The situation base KI can be re-interpreted into process knowledge base of n steps in a backward chaining or forward chaining manner as the following:

$$\mathsf{KI}(k) = \mathsf{KI}[\mathcal{P}_S/\mathcal{P}_{k,S}]$$
$$\mathsf{KI}_{process,backward}(\overline{n}) = \mathsf{KI}(0) \cup ... \cup \mathsf{KI}(n-1)$$
$$\mathsf{KI}_{process,forward}(\overline{n}) = \mathsf{KI}(0)$$

In backward chaining process knowledge, the initial states can be reached in any step during the state space searching. In contrast, in the forward chaining process knowledge, the initial states are always step 0. As the algorithms for state transition argumentation and planning introduced in this dissertation are in a backward chaining style, we assume that $\mathsf{KI}_{process} = \mathsf{KI}_{process,backward}$. For the algorithms which are in a forward chaining style, the process knowledge on initial states should employ $\mathsf{KI}_{process,forward}$.

Definition 7.53 (Goal base). *For an agent system* AGS*, a set of specification* $\mathsf{KG} \subseteq \mathcal{L}$ *using only state variables* $\mathcal{P}_S$ *is called a* goal base *to specify which goals states an agent system* AGS *is intended to achieve.*

The goal base KG can be re-interpreted into a process knowledge base of n steps in a backward chaining manner as follows:

$$\mathsf{KG}(k) = \mathsf{KG}[\mathcal{P}_S/\mathcal{P}_{k,S}]$$
$$\mathsf{KG}_{process,backward}(\overline{n}) = \mathsf{KG}(0)$$
$$h\mathsf{KG}_{process,backward}(\overline{n}) = \mathsf{KG}(0) \cup ... \cup \mathsf{KG}(n-1)$$

In forward chaining with process knowledge, the goal states can be reached in any step during the state space search. While in backward chaining, the goal states are always at the first step (step 0). As the algorithms for state transition argumentation and planning introduced are in a backward

chaining style, we assume that $\mathsf{KG}_{process} = \mathsf{KI}_{process,backward}$. For the algorithms which are in a forward chaining style, the process knowledge on goal states should employ $\mathsf{KG}_{process,forward}$.

Definition 7.54 (Initial state argument). *Given a situation base* KI*, an* initial state argument *is a combination argument*

$$A = \langle H, h \rangle$$

where $H \subseteq \mathsf{KI}$.

Definition 7.55 (Initial state argument in process form). *Given a situation base* KI *and its process re-interpretation* $\mathsf{KI}_{process}(n)$*, an* initial state argument in process form *is a combination argument*

$$A = \langle H, h \rangle$$

where $H \subseteq \mathsf{KI}(\overline{n})$.

In the definition of an initial state argument in process form, the support of an initial state argument is restricted to the re-interpretation of the initial situation specifications to exactly one step of the process. This process form of initial state argument works for both forward and backward process knowledge.

Definition 7.56 (Goal argument). *Given a goal base* KG*, a* goal argument *is a combination argument*

$$A = \langle H, h \rangle$$

where $H \subseteq \mathsf{KG}$.

Definition 7.57 (Goal state argument in process form). *Given a goal base* KG *and its process re-interpretation* $\mathsf{KG}_{process}(n)$*, a* goal state argument in process form *is a combination argument*

$$A = \langle H, h \rangle$$

where $H \subseteq \mathsf{KG}(\overline{n})$.

In the definition of a goal state argument in process form, the support of a goal state argument is restricted to the re-interpretation of the goal specifications to exactly one step of the process. This process form of a goal state argument works for both backward and forward process knowledge.

Definition 7.58 (Solution process argument). *Given an agent specification* KL*, a situation base* KI*, and a goal base* KG*, a* solution process argument *is a process argument based on* KL *with*

- *exactly one initial state sub-argument in process form based on* KI*; and*
- *exactly one goal state sub-argument in process form based on* KG.

Definition 7.59 (Solution argumentation framework). *Given an agent specification* KL*, a situation base* KI*, and a goal base* KG*, a* solution argumentation framework *is a preference-based argumentation framework* AFP *defined as:*

$$\mathsf{AFP}_{solution}(n) = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$$

where

- ARG *is the set of combination arguments that can be constructed on the process knowledge base* $\mathsf{KL}(\overline{n})$ *of length* n*, the situation base* $\mathsf{KI}(\overline{n})$*, and the goal base* $\mathsf{KG}(\overline{n})$*;*
- DFT *is the combination defeat relation on* ARG*;*
- PREF *is the union of the process preference* $\mathsf{PREF}_{solution}$ *and the preference on* ARG *derived from* $\mathsf{KM}(k)$ *for every step* $k = 0, ..., n-1$.

Definition 7.60 (Solution argumentation extensions). *A* solution argumentation framework *is a preference-based argumentation framework* $\mathsf{AFP}_{solution}(n) = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$ *extended with frame of state-action discernment. A preferred extension of situated goal processes is a preferred extension of* $\mathsf{AFP}_{solution}$ *that contains at least one situated goal process argument.*

Now we are ready to characterize the semantics of the weak policy and a strong policy planning algorithms on a set of inconsistent agent specifications, initial state specifications, and goal specifications in terms of process argumentation framework.

Proposition 7.15 (Planning argumentation). *Let* $\mathsf{AFP}_{solution}(\overline{n}) = \langle \mathsf{ARG}, \mathsf{DFT}, \mathsf{PREF} \rangle$ *be a solution argumentation framework based on an agent specification* KL*, its meta-knowledge* KM*, a situation base* KI *and a goal base* KG*. The preferred extensions of the execution argumentation framework* $\mathsf{AFP}_{exec}(\pi, \overline{n})$ *based on* KL*,* KM*, and a policy argument* A_π *that are computed by the planning algorithms (e.g. Algorithm 1 on page 67)*

- *are contained in the preferred situated goal extensions of* $\mathsf{AFP}_{solution}(n)$ *if a strong solution is computed;*
- *intersect with the preferred situated goal extensions of* $\mathsf{AFP}_{solution}(n)$ *if a weak solution is computed.*

Proof. Proposition 7.13 guarantees that every step of a preferred extension of the process argumentation framework on KL and KM sanctions the valid state transition relation. The correctness of Algorithm 1 and its variants guarantee that the policy π that is computed for the given set of initial states and goal states is either a strong or weak solution to the planning problem. Combining the policy argument A_π into the process argumentation framework produces the execution argumentation framework $\mathsf{AFP}_{exec}(\pi, \overline{n})$. Algorithm 1 guarantees that the initial states and goal states are covered by the solutions in a weak solution manner and a strong solution manner. This in turn guarantees that the family of preferred extensions of the execution argumentation framework $\mathsf{AFP}_{exec}(\pi, \overline{n})$ are contained in the preferred situated goal extensions of $\mathsf{AFP}_{solution}(n)$ if a strong solution is computed, and such a family of extensions intersect with the preferred situated goal extensions of $\mathsf{AFP}_{solution}(n)$ if a weak solution is computed.

7.14 Related work and discussions

Atkinson and Bench-Capon [2007] proposed a presumptive argumentation system for practical reasoning based on argumentation schemes [Walton, 1996]. Argumentation schemes are a set of distinct structures that can be instantiated to match against knowledge and belief input to generate argumentation. It consists of argument schemes, which are a set of abstract structures (or templates) that capture the reasoning (either deductive or non-deductive reasoning) within an argument, and critical questions (defeat schemes), which are a set of abstract structures that capture defeat relations between arguments. Argument schemes can be seen as *macro* arguments comprised of multiple lower level arguments that, when combined, are used for a particular purpose. These schemes can thus capture complex arguments concisely, structuring them and facilitating its comprehension by human users.

In [Atkinson and Bench-Capon, 2007], an alternative state transition system is used to underpin the argumentation system. States are explicitly referred to in the language used in the argumentation schemes [Walton, 1996]. These argumentation schemes are essentially a form of knowledge engineering template which need to be specified explicitly. Valuation of state aspects is incorporated into the language, and a qualitative approach of promoting and demoting values can be associated with the actions. However, the formalism doesn't handle non-deterministic state transitions. It doesn't explicitly account for how argumentation-based practical reasoning can be propagated through the state transition system, and it doesn't provide algorithms matching the argument schemes against the input and how to argue and make decisions in the alternative state transition system.

The defeasible action theory proposed in this dissertation can be viewed as an automatic reasoning mechanism to pick the reasoning that can match the argument schemes and critical questions proposed in [Atkinson and Bench-Capon, 2007] instead of specifying them explicitly, and also the basic mechanism can pick up more practical reasoning and conflicts among these reasoning as it is based on the inconsistency in the underlying model theory. The formalism that are presented in this

dissertation are devised so that they can eventually be incorporated into another line of work on the probabilistic HTNs planning [Tang et al., 2011b] to be capable of propagating of probabilities and utilities through argumentation over action theory and planning.

Following [Searle, 2001], Atkinson and Bench-Capon [2007] identified three characteristics of practical reasoning:

(a) The reasoning is defeasible and must be evaluated in the context of the arguments that have been made in the particular situation. Additional arguments which can potentially lead to re-evaluation are always possible.

(b) The reasoning is inherently subjective, because it concerns choice. Different agents have different values, interests and aspirations, and this may lead them to accept different arguments.

(c) The preferences expressed by an agent should be capable of being a product of rather than an input to practical reasoning.

This dissertation accepts the proposed characteristics (a) and (c) in practical reasoning and implement them with an argumentation on both action theory and planning based on the action theory. However, this dissertation accepts in part characteristics (b) with the following modification:

(b) The reasoning should be a combination of subjective and objective reasoning, because it concerns both choices and the objective nature of the external environment. Different agents have different perceptions of the environment, values, interests and aspirations, and this may lead them to accept different arguments.

The subjective characteristics come from the agents and their users' observation of the objective world is subjective. However, there is a objective aspect of the external environment behind these subjective views. The reasoning mechanism should not only reflect subjective views of perceptions, interests and aspiration but also reflect how these subjective information can propagate through the information and reasoning regarding to the objective nature of the external world.

As defeasible action theory is concerned, there is work on using argumentation for defeasible action theory [Vo and Foo, 2005] with the focus on solving three fundamental problem in action theory: the qualification problem, the ramification problem and the frame problem. In [Vo and Foo, 2005], the assumption-based argumentation framework [Bondarenko et al., 1997] (see Section 7.4.3) is used to carry out defeasible reasoning over an action theory. The action theory is represented in a formal language is proposed with explicit symbols to represent the applicability of the qualification and frame conditions, and explicit symbols to represent time and duration. Similar to the approach in this dissertation, [Vo and Foo, 2005] also apply preferred extension semantics (see Section 7.4.1) to characterize the plausible outcome of the defeasible action theory. The major difference is that [Vo and Foo, 2005] only considers one preferred extension but the argumentation semantics in this dissertation take into account all the preferred extensions and translate the multiple preferred extensions in non-deterministism in the state transitions. And this non-determinism in state transitions and its planning argumentation counter-part is considered in the planning algorithms. The planning algorithms (Chapter 6) and the corresponding argumentation for planning present in this Chapter make another major difference between this work and that of [Vo and Foo, 2005]. [Vo and Foo, 2005] is mainly concerned about defeasible reasoning on actions not about planning. Also [Vo and Foo, 2005] has the argumentation based on a very expressive language, the deductive system needed by assumption based argumentation framework [Bondarenko et al., 1997] will prevent it from being an implementable tool making it only a theoretical analysis tool.

As far as planning is concerned, García et al. [2008] proposed an argumentation formalism for partial order planning (a form of plan space planning [Chapman, 1987]). The argumentation is based on defeasible logic programming [Garcia and Simari, 2004]. Defeasible rules are used to simulate the applicability and outcomes of the action operators (similar to that of Chapter 3), and to simulate the operations to add an action and promote and demote the order of an action. The planning begins with an empty sequence of action. Then a dialectical semantics, which is equivalent to the grounded semantics, is used to derive undefeated arguments for a plan. The actions in this formalism are deterministic, and the planning algorithms output a sequence of actions (or a partial

order of actions) that can achieve the goals with respect to the defeasible action theory. The work in this dissertation differs from that of [García et al., 2008] in the following two aspects:

- formalisms of argumentation are provided to explicitly capture the reasoning about one step action, policy as well as execution structures so as to capture more essential aspects of an action theory and its applications in a planning process, and
- multiple preferred extensions in argumentation semantics are translated into a non-deterministic action theory (and its Kripke structure interpretation), and the planning algorithms and the argumentation framework are able to reason about this non-deterministism.

7.15 Summary

In this chapter, argumentation-based reasoning is introduced to underpin the computation of state transitions extracted from the defeasible factored information and its planning algorithms.

The argumentation for processes introduced in this chapter can be viewed as a way to argue over a family of Kripke structures that can be derived from the given specifications. In the state transition specifications, the Markov property is assumed. The Markov property means that the next state only depends on the current state and the current action. This is also called the first-order Markov property because the resulting next state only depends the current state and action not depend on the history before the current state. In general a k-order Markov property means that the resulting next state only depends on the k previous steps of state-action pairs and has nothing to do with the steps before them. It is a future direction to extend the work in this dissertation to argue about defeasible information about k-order Markov property.

Chapter 8

Related work

This chapter will survey the related work in action theories, planning, multiagent planning, and argumentation theory.

8.1 Action theories

In [Thielscher, 2000], an action theory is defined informally as

> An action theory consists of a formal language that allows adequate specifications of action domains and scenarios, and it tells us precisely what conclusions can be drawn from these specifications.

In the literature, there are two schools of formal languages used in action theory: STRIPS [Fikes and Nilsson, 1971] and situation calculus [McCarthy, 1963; McCarthy and Hayes, 1969].

Situation calculus and its descendents, such as fluent calculus [Thielscher, 1998, 1999] and event calculus [Kowalski and Sergot, 1986], represent properties of states, actions and how actions affect properties of states in first order logic formulae, and then use a theorem prover to draw conclusions on the outcomes of actions. In situation calculus, situations are characterized by fluents, and changes in the world are characterized by axioms that describe how fluents are changed by actions.

A *fluent* is a predicate or function whose truthfulness or value changes relative to time. There is also a special function $do(a, s)$ to denote the situation resulting from performing action a on situation s. For example,

$$empty(A, s) \rightarrow (on(B, A, do(putOn(B, A), s)) \wedge \neg empty(A, do(putOn(B, A), s)))$$

represents a piece of knowledge saying that, when a block A is empty, performing an action $putOn(B, A)$ will result in a new situation $do(putOn(B, A), s)$, and in such a situation B will be on top of A, represented by $on(B, A, do(putOn(B, A), s))$, and the block A will not be empty represented by $\neg empty(A, do(putOn(B, A), s))$. Since theorem proving in first order logic is undecidable, situation calculus is mostly used to formalize theories and scenarios involving actions and effects, and to prove formal properties of these theories and scenarios. It is rarely implemented and applied to AI planning.

The STRIPS language uses a list of operators which correspond to actions that change the world. Each operator is associated with a precondition, an add list and a delete list. The precondition specifies the condition under which the operator can apply. The add list specifies which new properties of the world that the operator can bring into. The delete list specifies which old properties of the world the operator will remove. An example from [Fikes and Nilsson, 1971] is to describe how a robot can move objects in the physical world. It uses a predicate $ATR(x)$ to denote that the robot is at location x, and another predicate $AT(k, x)$ to denote an object k is at location x [1]. For example, an operator $push(k, x, y)$ to denote the robot push object k from location x to location y:

$$\text{Operator} : push(k, x, y)$$

$$\text{Precondition} : ATR(x), At(k, x)$$

$$\text{Add} : ATR(y), At(k, y)$$

[1] Influenced by situation calculus, in its original form [Fikes and Nilsson, 1971], each fluent predicate has a situation parameter, such as $ATR(x, s)$ and $AT(k, x, s)$ to denote the location of the robot and the location of an object in the situation s. In later versions of STRIPS, the situation parameter is removed.

Delete : $ATR(x), At(k, x)$

Essentially the STRIPS language describes the transformation that an action performs in terms of the description of states. By limiting the form of expressions associated with the operators, STRIPS can simplify the reasoning process by checking whether a predicate is in the list instead of running a full first order theorem prover so as to make the formalism efficient in practice. Following the direction of STRIPS, many other languages have been developed, notably PDDL [Ghallab et al., 1998] which acts as a standard language for describing the testbed problems for AI planning, Action Description Language (ADL) [Pednault, 1989, 1994] which grounds the semantics on state transitions, and so on [Gelfond and Lifschitz, 1998; Giunchiglia and Lifschitz, 1998]. [Gelfond and Lifschitz, 1998] provides a comprehensive review of the action languages following the line of STRIPS and characterize the relation between these languages based on the state transition model.

The development of action theory focuses on solving the three fundamental problems: the frame problem [McCarthy and Hayes, 1969], the qualification problem [McCarthy, 1977] , the ramification problem [Lin and Reiter, 1994]. The frame problem refers to the need to specify what is not changed from situation to situation. For example, from the situation s to the situation $do(putOn(B, A), s))$ (the situation after performing an action $putOn(B, A)$), one needs to specify for each influent predicate whether it will changed by the action or not. The qualification problem refers to need to specify precisely the conditions under which the effects of the action can take place. The difficulty lies in the fact that there are possibly infinitely many such conditions. For example, to start a car, one need to check whether the key is the right one, whether the mechanics is working well, whether the gas tank is not empty, and so on. It is impractical to specify all of these conditions. The ramification problem refers to the need to specify precisely the indirect effects of an action derived from the dependency among various world aspects. For example, moving a box from one location to another not only changes the location of the box but also changes the location of the contents of the box. As we can see, these three problems are interrelated, a full solution to any of them involves solving the other two.

Solutions to these three fundamental problems fall into two categories: the monotonic approach and non-monotonic approach. On monotonic side, STRIPS solves the frame problem by syntactically manipulating the description of states instead of going through classical theorem proving. Others [Reiter, 1991; Castilho et al., 1999] try to solve the frame problem through acclimatization of the positive, negative effects and frames using conditionals in the first order language. However, when the frame problem is combined with the qualification and ramification problem, the solution is inherently non-monotonic as the unexpected qualification and ramification effects of an action will force the frame axioms related to such effects to be retracted [Vo and Foo, 2005]. The non-monotonic formalisms that have been applied to these fundamental problems includes truth maintenance system [Doyle, 1987], default logic [Reiter, 1980], circumscription [McCarthy, 1987], modal non-monotonic logic [McDermott and Doyle, 1987], and argumentation-based reasoning [Vo and Foo, 2005].

Specially, the argumentation-based reasoning approach [Vo and Foo, 2005] to the fundamental problem is closely related to the formalism developed in this dissertation. [Vo and Foo, 2005] surveys all the previous approaches to the fundamental problems in action theory, and argues that all of them can only solve the problems to some extent and none can solve all the three problems together. [Vo and Foo, 2005] solves the three fundamental problems simultaneously by introducing an action language with symbols to represent whether the qualification assumptions and the frame assumptions for each fluent can apply or not, and then applies assumption-based argumentation [Bondarenko et al., 1997] reviewed in Section 7.4.3. [Vo and Foo, 2005] solves the problems by explicitly asserting the qualification and frame assumptions into the assumption base, and having a set of axioms which specify the conditions that can support or defeat the qualification and frame assumptions. The ramification problem is automatically solved through the set of ordinary axioms on describing the effects and indirect effects. The introduction of the qualification and frame assumption symbols in this approach is similar to the label symbols introduced for the specifications in this dissertation. The difference lies in the kind of support and defeat that are available in the system. While [Vo and Foo, 2005] reasons about the applicability of qualifications and frames

on individual fluents, Chapter 7 on the other hand reasons about the combined applicability of all possible consistent specifications. This shifts the focus from applicability of fluents to the application of interaction between different specifications so as to extend it to multiagent scenarios.

This dissertation is not directly concerned with the three fundamental problems in action theory. Rather it focuses on how specifications from multiple agents can be put together to form a mutually agreed and plausible understanding of the way the world changes. Since the specifications of individual agents are not aware of the specifications of other agents, the three fundamental problems in action theory become more complex in a multiagent setting. In the mutiagent setting, whether a part of the world, represented by a predicate, is changing or not depends not only on the local perceived part of the world and actions but also depends on the other agents' perceptions and actions. This requires the reasoning mechanism used by the agents to consider the frame problem, the qualification and ramification problem together along with the major effects of the joint actions. The argumentation reasoning approach used in this dissertation, puts all possible combinations of the related information regarding actions into all consistent arguments which support the set of all possible state transitions. The components of these arguments consider all appropriate information related to the major effects, the frame problem, the qualification problem, and the ramification problem, and puts them together if together they can generate a consistent state transition model. Then the inconsistency between these combinations of specifications is captured with the notation of "defeat" in argumentation, so as to extract the most plausible consistent extensions of state transitions as the output. Multiple extensions are considered as non-deterministic properties of the state transition models. In summary, the approach proposed this dissertation, although it did not set out to solve the three major problems in action theory, provides a new approach to consider all these problems together leading to a solution for multiagent action theory that describes how different agents' knowledge about the actions can come into play together.

8.2 AI Planning

Planning is an essential component for the agents to decide what to do to achieve their goals in a long run. AI planning is concerned with how, given a model of the external world and the capability of the agents looking for a plan, it is possible to construct a plan to achieve a desired state of the world. A plan is captured by a data structure over a set of actions of the agents. so that a desired state of world can be achieved. Starting with STRIPS [Fikes and Nilsson, 1971], classical planning has used a state space model. The world is modeled as being in states with a factored representation — by a set of formulae in a logical language or by a collection of variables, and actions (operators) are available to change the world from states to states. The planning problem is then to look for a sequence of actions that can lead the agent towards the desired goal states from an initial state. Various representation languages [Fikes and Nilsson, 1971; Ghallab et al., 1998], algorithms [Fikes and Nilsson, 1971], and semantics [Lifschitz, 1987] have been studied within the state space setting.

Another planning setting is plan space planning [Chapman, 1987; McAllester and Rosenblitt, 1991; Soderland and Weld, 1991; Penberthy and Weld, 1992] in which the planning processes can be abstracted as refining a given partial plan into a detailed plan that can achieve the desired goals. A partial plan is usually composed of an ordering over on the actions along with additional logical constraints over the actions. Hierarchical Task Networks [Erol et al., 1994] is one such representation in which nodes represent actions, edges between nodes represent an ordering between the actions. In task networks, logical expressions can be attached to nodes and edges to represent the properties of the states on which the actions can be carried and the properties the actions should preserve respectively. The representation of a partial plan essentially captures a collection of plans instead of one specific plan. Plan space planning then starts with a general partial plan and refines the plan incrementally into more detailed partial plan(s) until an executable plan (such as a sequence of actions or a detailed partial plan which will be refined during execution) that can achieve the desired goals is found.

While state space planning and plan space planning deals with deterministic actions — executing

an action in a state results in a deterministic state — recent development in planning consider non-deterministic or probabilistic domains and producing plans that can take into account sensing information during the planning. Non-determinsitic planning [Cimatti and Roveri, 2000; Cimatti et al., 2003] and decision theoretical planning in stochastic environments [Boutilier et al., 1999] are part of this new trend.

The complexity of planning is in general (both state space planning and plan space planning) PSPACE-hard [Bylander, 1992]. To conquer the complexity problem in practice, heuristics are used to guide the planning processes to try the actions that are most likely to lead to the goals as soon as possible. These heuristics are usually admissible — an admissible heuristic never underestimates the costs towards the goals in search for a plan. Among the heuristics, a plan graph [Blum and Furst, 1997] is a general data structure that can provide such an admissible heuristic.

This dissertation follows the non-deterministic planning approach [Cimatti and Roveri, 2000; Cimatti et al., 2003], and extends it to a mutliagent system. To the best of my knowledge, this is the first work to extend non-deterministic planning based on model checking to multiagent systems. In parallel to the work in this dissertation, I am also working with other colleagues on incorporating HTNs and decision theoretical planning to the formalism in this dissertation (see [Tang et al., 2011c,b; Meneguzzi et al., 2011, 2010]) to accommodate probabilities and utility information.

8.3 Multiagent planning

When planning meets multiagent systems, the problems of distributed information and coordination arise. The problem of distributed information requires solving the problems of information integration and inference. The problem of coordination requires solving the problems of action interaction and communication. The problems related to distributed information are usually investigated by looking at the characteristics of interaction situations (scenarios) [Jennings, 1996]. The problem of coordination is usually investigated by looking at the interactions between actions [Decker and Lesser, 1995]: the choices of action combinations that can affect the overall outcomes,

the order of the actions, and the resources and time constraints over action combinations and the order of the actions. In the literature [Durfee, 1999; de Weerdt et al., 2005], the multiagent planning process can be divided into the following phases:

- Refine the global goals or tasks into subtasks that can be executed by individual agents
- Allocate the tasks to agents
- Define rules or constraints for individual agents to prevent conflicting plans
- Individual agents plan to reach its goals
- Coordinate individual plans
- Execute plans and synthesize outcomes of tasks

A naive way to tackle these problems is to collect all the information in a centralized manner, represent all possible combinations of the actions of the agents, and specify their inferences and interactions for each combinations. However, the combinations of simultaneous actions are exponential in the number of actions.

Various approaches have proposed to solve these problems such as TÆMS [Decker and Li, 2000], GPGP [Lesser et al., 2004] and DEC-POMDPs [Bernstein et al., 2000; Petrik and Zilberstein, 2009].

TÆMS[Decker and Li, 2000] proposed a blackboard approach to model the global constraints, such as resources, time, utilities and so on, shared by all the agents. It is the basic data structures used by GPGP [Lesser et al., 2004]. GPGP uses a goal hierarchy (HTN style knowledge) and partial ordering planning to guide the propagation of the shared constraints recorded in the TÆMS data structure as to solve the underlying complicated constraint optimization problems. By reducing the problem to constraint programming, a distributed satisfaction approach can be applied for multiagent plan coordination [Cox et al., 2005]. Following the approach of planning as constraint satisfaction in [Do and Kambhampati, 2001], [Cox et al., 2005] models multiagent plan as partial plans composed of steps (operators), and causal links and temporal links. Threats are defined with

respect of causal links flaws, open pre-condition flaws, and parallel step flaws. Variables are created to represent these flaws explicitly in the constraint satisfaction problem, and then the partial order planning step requirements are captured as constraints between the variables corresponding to the flaws. Thereafter, plan coordination (which can be viewed as a plan refinement procedure after individual agents produce their own plans) can be achieved through solving a distributed constraint satisfaction problem.

DEC-POMDPs [Bernstein et al., 2000; Petrik and Zilberstein, 2009] extend decision theoretical planning of single agent [Boutilier et al., 1999] to multi-agent planning. To the best of my knowledge, the DEC-POMDP approach does not consider communication among the agents. Instead, the approach models interactions among the agents in the same way as the interactions between the agents and the environment. It models both types of interactions as an observation problem in the environment using Decentralized Partial Observable Markov Decisions Processes (DEC-POMDPS).

8.4 Argumentation theories

Argumentation theory has been proposed for both reasoning and modeling dialogues between agents. Argumentation-based reasoning is a defeasible reasoning paradigm in which reasons from premises to conclusions are recorded into a data structure called an *argument*, *conflict*s between these arguments are detected into a *defeat relation*, and a conflict-free set of acceptable arguments can then be reached with respect to a notion of collective acceptability [Dung, 1995]. On the other hand, an argumentation-based dialogue is a model in which arguments are exchanged among multiple parities proving or disproving logical propositions [Walton, 2009] by either supporting or undermining intermediate conclusions during the course of a dialogue.

The argumentation-based reasoning mechanism can be viewed as a mathematical abstraction of the dialectical principles which are extracted from the study of human argumentation (Walton and Krabbe [Walton and Krabbe, 1995], Hamblin [Hamblin, 1970], and Toulmin [Toulmin, 1958; Toulmin et al., 1984]). The input to the reasoning scheme is knowledge represented in a formal

language and sometimes an additional notion of preference over the knowledge, which can be represented inside or outside the formal language; its output is a characterization of the arguments and their conclusions. The two key components of an argumentation system are the process to construct arguments and the process to analyze the conflict pattern between these arguments. In the argument construction process, reasoning from the knowledge database to conclusions is recorded into a syntactic structure called an argument structure; then in the conflict analysis process, the conflicts between arguments will be captured and analyzed to characterize the acceptability of arguments and their conclusions. A set of arguments is *conflict-free* if there are no conflicts between any two arguments within this set. A set of arguments is collectively acceptable if any argument which defeats an argument in the set is defeated by an argument in this set. The notion of collective acceptability exerts a form of stability that can resist to external challenges. The most interesting property of argumentation based reasoning is the concept of "external stability" [Dung, 1995] through which a set of coherent beliefs and reasons is characterized by the relations between the arguments "internally" supporting the beliefs and reasons and the arguments "externally" supporting the contradictory beliefs and reasons. The concept of "external stability" is proved to correspond to the solution concept of an N-person game [Dung, 1995]. This makes argumentation-based reasoning an attractive inference engine to organize the information on actions and goals in multiagent systems. Other semantics of argumentation-based reasoning have also been proposed, such as [Caminada and Amgoud, 2007], [Cayrol and Lagasquie-Schiex, 2005] , [Jakobovits and Vermeir, 1999], [Besnard and Hunter, 2001], [Pollock, 2001], which have expanded our understanding of what argumentation can be used for, and have created a bridge to possibility theory and plausibility theory in the field of reasoning about uncertainty.

An influential model of human dialogues is the typology of primary dialogue types of argumentation theorists Doug Walton and Erik Krabbe [Walton and Krabbe, 1995]. This categorization is based on the information the participants have at the commencement of a dialogue (of relevance to the topic of discussion), their individual goals for the dialogue, and the goals the participants share. A list of dialogue types in Walton and Krabbe's categorization are:

- Information-seeking dialogues: One participant seeks answer(s) to some question(s) from another participant who is believed by the first participant to know the answer(s).
- Inquiry dialogues: The participants collaborate together to answer some question(s) whose answer(s) are not known to any individual participant.
- Persuasion dialogues: One participant seeks to persuade another to accept his own proposition on some issues (for which the other participant may have a different opinion).
- Negotiation dialogues: The participants bargain over the division of some scarce resources. The goal of the dialogue — a division of the scarce resources — may be in conflict with individual participants' goals.
- Deliberation dialogues: Participants collaborate to decide what action or course of actions should be taken in certain situations. Here, participants share the responsibility to decide the course of actions.
- Eristic Dialogues: Participants quarrel verbally as a substitute for physical fighting, aiming to vent perceived grievances.

Following this categorization of dialogues, researchers in multiagent system have proposed computational multiagent dialogue systems for information seeking [Amgoud et al., 2000b], inquiry [Amgoud et al., 2000a], negotiation [Amgoud et al., 2000b; Parsons et al., 1998], persuasion [Prakken, 2000] and deliberation [Hitchcock et al., 2001; Parsons et al., 1998; Tang and Parsons, 2005]. (Eristic dialogues are rarely important in computational multiagent dialogue.)

8.5 Summary

Action theories study how information regarding the external world can be put together to draw conclusions about the applicability and outcomes of actions and plans. To solve the three fundamental problems in action theories, non-monotonic reasoning is unavoidable. Planning techniques are

mostly based on STRIPS-style action languages which are not able to solve all the three fundamental problem. Non-deterministic and decision-theoretic planning open new ways to make planning work in an uncertain environment. Non-deterministic planning offers us a logical structure for the planning problem, while decision theoretical planning offers us a stochastic and utility-based view of the planning problem. Argumentation theory offers us a unified view of different kinds of non-monotonic reasoning, and this view is dialectical and is readily available as a reasoning mechanism among multiple agents.

The approach proposed in this dissertation combines defeasible action theories, argumentation-based reasoning and non-deterministic planning to enable multiple agents to plan coherent and effective behaviors in their problem solving activities while these agents are given defeasible knowledge and are situated in a non-deterministic environment. In parallel to work present in this dissertation, we have worked on incorporating decision theoretic planning into the formalism proposed here (see [Tang et al., 2011b] for preliminary result). The work in this dissertation will provide a logical structure to which stochastic information can ascribed.

Chapter 9

Conclusion

In this chapter Section 9.1 will first summarize the contributions of this dissertation and Section 9.2 will outline possible future directions.

9.1 Contributions

This dissertation addresses the following challenges in constructing multiagent systems:

1. Acquired knowledge (either from human or from machine learning algorithms) about the agents' actions and the inter-agent interactions are distributed, inconsistent and uncertain.

2. The pre-set goals of the agents, as well as the intermediate generated goals during the execution, are also distributed, inconsistent and uncertain.

3. Given the above challenges, it is a complex process to plan coherent long-term behaviors with necessary coordination for a system of agents leading to the pre-set objectives of these agents.

In response to the above challenges, I have proposed a set of algorithms to symbolically explore the multiagent state space and to reason about valid state transitions and the policy to control transitions in the state space. The approach is based on explicitly encoding the state space, the action space and the state transitions with QBFs/BDDs. The multiagent state space is produced by

an argumentation-based reasoning process about combining distributed, inconsistent, and uncertain specifications on the state space. Non-deterministic planning concepts — weak, strong, strong cyclic plans — are employed to generate joint plans for the agents to behave in the joint state space. With the power of argumentation-based reasoning, inconsistent goals can be resolved to achieve maximally preferred extensions of the goals of all the agents. With the same mechanism, the non-deterministic policy generated for the agents can be coordinated by an argumentation-based coordination mechanism. Overall this dissertation has following major contributions:

1. **Symbolic techniques for multiagent state space — the model, decentralized planning and interaction analysis** [Chapter 4]

   Symbolic techniques are introduced to explore the multiagent state space and analyze the interaction among the agents in this state space.

2. **Defeasible factored action theory and its symbolic implementation** [Chapter 5]

   A defeasible factored action theory is proposed to specify the state space for individual agents and a multiagent system as a whole.

3. **Non-deterministic decentralized multiagent planning and coordination on defeasible factored information** [Chapter 6]

   A decentralized planning system is proposed to make use of the non-monotonic defeasible factored information, and this system is able to plan and coordinate multiple inconsistent goals.

4. **Argumentation theory for defeasible reasoning about actions, planning and execution of planning** [Chapter 7]

   The reasoning procedures and the non-deterministic planning procedures are re-formalized to be underpinned by an argumentation theory.

5. **Introducing formalisms and techniques from other areas to AI planning, argumentation, and MAS research**

   Symbolic checking techniques is used in combination with the notation information algebra [Kohlas, 2003a] which can provide a well-defined mathematical framework for understanding multi-agent behaviors [see Chapter 2]. Symbolic techniques are also employed to enable the implicit exploration of a large number of argumentation for the planning processes all together simultaneously [see Chapter 7].

9.2 Future directions

The work present in this dissertation focused on the logical structure of the state space, the planning procedure, and the produced policy. The next step of this work is to

- extend the QBFs/BDDs encoding to accommodate more general numerical/algebraic information than the preference levels so that
  - probability information can be accommodated to enable the agents to learn from past experiences (see [Tang et al., 2011b,c] for our preliminary work),
  - value and cost information can be accommodated so that agents can identify plans with maximal expected utilities (see [Tang et al., 2011b,c] for our preliminary work),
  - trust measures among the agents can be accommodated and propagated through the agents and the state transitions in the state space (see [Tang et al., 2010a; Parsons et al., 2011a,b; Tang et al., 2011a] for our preliminary work)
- apply hierarchical task networks to abstract high level procedure specifications for the agents so that the system so that the system can accommodate carefully engineered plan library for the agents (see [Tang et al., 2011b,c] for our preliminary work)
- devise multi-agent dialogue formalisms to better organize the communication among the agents for the planning process as well as for plan execution (see [Tang and Parsons, 2005,

2006, 2008; Tang et al., 2008; Tang and Parsons, 2009; Tang et al., 2009; Tang, 2010] for our preliminary work)

- bridge to plan recognition and machine learning (see [Tang et al., 2009] for our preliminary work)

The ultimate goal is to enable to agents to automatically develop social intelligence — the decision making procedures, the communication, and the coordination necessary to reach a level of coherent behaviors — given that these agents' specifications, perceptions and goals are possibly inconsistent, incomplete, and uncertain.

Index

Bibliography

L. Amgoud and C. Cayrol. Inferring from inconsistency in preference-based argumentation frameworks. *Journal of Automated Reasoning*, 29(2):125–169, 2002a. 7.2, 7.3, 7.4.2

L. Amgoud and C. Cayrol. A reasoning model based on the production of acceptable arguments. *Annals of Mathematics and Artificial Intelligence*, 34(1-3):197–215, 2002b. ISSN 1012-2443. 7.2, 7.3, 7.4.1, 7.4.2, 7.4.3

L. Amgoud, N. Maudet, and S. Parsons. Modeling dialogues using argumentation. In *ICMAS '00: Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, pages 31–38. IEEE Computer Society, 2000a. ISBN 0-7695-0625-9. 8.4

L. Amgoud, S. Parsons, and N. Maudet. Arguments, dialogue, and negotiation. In W. Horn, editor, *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence*, pages 338–342, Berlin, Germany, August 20-25 2000b. 8.4

K. Atkinson and T. Bench-Capon. Practical reasoning as presumptive argumentation using action based alternating transition systems. *Journal of Artificial Intelligence*, 171(10-15):855–874, 2007. 1.1.3, 1.3, 7.14

D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of markov decision processes. In *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 32–37, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-709-9. 1.1.2, 8.3

P. Besnard and A. Hunter. A logic-based theory of deductive arguments. *Artificial Intelligence*, 128 (1-2):203–235, 2001. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(01)00071-6. 7.1, 8.4

A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, February 1997. ISSN 0004-3702. doi: 10.1016/S0004-3702(96)00047-1. URL `http://dl.acm.org/citation.cfm?id=249379.249386`. 8.2

J. Blythe. An overview of planning under uncertainty. In M. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today*, volume 1600 of *Lecture Notes in Computer Science*, pages 85–110. Springer Berlin / Heidelberg, 1999. URL `http://dx.doi.org/10.1007/3-540-48317-9_4`. 1.1.1

B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-Complete. *IEEE Transations on Computers*, 45(9):993–1002, 1996. ISSN 0018-9340. doi: http://dx.doi.org/10.1109/12.537122. 2.2.1

A. Bondarenko, P. M. Dung, R. A. Kowalski, and F. Toni. An abstract argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93:63–101, 1997. 1.1.3, 1.3, 7.4.1, 7.4.3, 7.4.3, 7.14, 8.1

C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999. 1.1.2, 3.2, 8.2, 8.3

R. Brachman and H. Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. ISBN 1558609326. 2.4.2

M. E. Bratman. What is intention? In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 15–32. MIT Press, Cambridge, MA, USA, 1990. 1.1.1, 5.1

R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14 – 23, mar 1986. ISSN 0882-4967. doi: 10.1109/JRA.1986.1087032. 1.1.1, 5.1

R. A. Brooks. Intelligence without reason. In *COMPUTERS AND THOUGHT, IJCAI-91*, pages 569–595. Morgan Kaufmann, 1991a. 5.1

R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, February 1991b. ISSN 0004-3702. doi: 10.1016/0004-3702(91)90053-M. URL `http://dl.acm.org/citation.cfm?id=110289.110295`. 5.1

R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986. 1.2, 2.2.1, 1

R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/136035.136043. 2.1, 2.2, 2.2.1, 18

J. Burch, E. Clarke, K. McMillan, D. Dill, L. H. J. R. Burch, E. M. Clarke, K. L. Mcmilla, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press. 2.2.1, 16, 18

J. R. Burch. Using BDDs to verify multipliers. In *DAC '91: Proceedings of the 28th Conference on ACM/IEEE Design Automation*, pages 408–412, New York, NY, USA, 1991. ACM. ISBN 0-89791-395-7. doi: http://doi.acm.org/10.1145/127601.127703. 2.2.1

J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. In *Proceedings of International Conference on Very Large Scale Integration*, pages 49–58. North-Holland, 1991. 2.2.1

J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13:401–424, 1994. 2.2.1

C. Burnett, D. Masato, M. McCallum, T. J. Norman, J. Giampapa, M. J. Kollingbaum, and K. Sycara. Agent support for mission planning under policy constraints. In *Proceedings of the Second Annual Conference of the ITA*, Imperial College, London, 2008. 3.1.2

T. Bylander. Complexity results for serial decomposability. In *AAAI*, pages 729–734, 1992. 18, 8.2

G. Cabodi, P. Camurati, L. Lavagno, and S. Quer. Disjunctive partitioning and partial iterative squaring: an effective approach for symbolic traversal of large circuits. In *DAC '97: Proceedings of the 34th Annual Conference on Design Automation*, pages 728–733, New York, NY, USA, 1997. ACM. ISBN 0-89791-920-3. doi: http://doi.acm.org/10.1145/266021.266355. 2.2.1

M. Caminada and L. Amgoud. On the evaluation of argumentation formalisms. *Artificial Intelligence*, 171(56):286 – 310, 2007. ISSN 0004-3702. doi: 10.1016/j.artint.2007.02.003. URL `http://www.sciencedirect.com/science/article/pii/S0004370207000410`. 7.1, 8.4

M. Castilho, O. Gasquet, and A. Herzig. Formalizing action and change in modal logic i: the frame problem. *Journal of Logic and Computation*, 9(5):701–735, 1999. doi: 10.1093/logcom/9.5.701. URL `http://logcom.oxfordjournals.org/content/9/5/701.abstract`. 8.1

C. Cayrol and M.-C. Lagasquie-Schiex. Graduality in argumentation. *Journal of Artificial Intelligence Research*, 23:245–297, 2005. 7.1, 8.4

D. Chapman. Planning for conjunctive goals. *Artif. Intell.*, 32:333–377, July 1987. ISSN 0004-3702. doi: 10.1016/0004-3702(87)90092-0. URL `http://dl.acm.org/citation.cfm?id=25677.25679`. 1.1.2, 1.1.3, 5.1, 7.14, 8.2

P. Chauhan, E. M. Clarke, S. Jha, J. Kukula, T. Shiple, H. Veith, and D. Wang. Non-linear quantification scheduling in image computation. In *ICCAD '01: Proceedings of the 2001 IEEE/ACM International Conference on Computer-aided Design*, pages 293–298, Piscataway, NJ, USA, 2001. IEEE Press. ISBN 0-7803-7249-2. 2.2.1

A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000. 1.1.2, 5.1, 8.2

A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(02)00374-0. 1.1.2, 5, 2.2.1, 3, 3.2, 3.3, 3.5, 16, 3.6, 3.7, 5.1, 8.2

P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42 (2–3):213–261, 1990. doi: 10.1016/0004-3702(90)90055-5. 1.1.1, 5.1

O. Coudert and J. C. Madre. The implicit set paradigm: a new approach to finite state system verification. *Formal Methods in System Design*, 6(2):133–145, 1995. ISSN 0925-9856. doi: http://dx.doi.org/10.1007/BF01383965. 2.1, 2.2.1, 18

O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In *Automatic Verification Methods for Finite State Systems*, pages 365–373, 1989. 2.2.1

J. S. Cox, E. H. Durfee, and T. Bartold. A distributed framework for solving the multiagent plan coordination problem. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, AAMAS '05, pages 821–827, New York, NY, USA, 2005. ACM. ISBN 1-59593-093-0. doi: http://doi.acm.org/10.1145/1082473.1082598. URL `http://doi.acm.org/10.1145/1082473.1082598`. 8.3

M. de Weerdt, A. ter Mors, and C. Witteveen. Multi-agent planning: An introduction to planning and coordination (handout). http://www.st.ewi.tudelft.nl/ mathijs/publications/easss05.pdf, 2005. 8.3

K. Decker and V. Lesser. Designing a family of coordination algorithms. In V. Lesser, editor, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 73–80, San Francisco, CA, USA, 1995. The MIT Press: Cambridge, MA, USA. 1.1.2, 8.3

K. Decker and J. Li. Coordinating mutually exclusive resources using GPGP. *Autonomous Agents and Multi-Agent Systems*, 3:133–157, June 2000. ISSN 1387-2532. doi: 10.1023/A:1010074611407. URL `http://dl.acm.org/citation.cfm?id=608603.608650`. 1.1.2, 8.3

Y. Dimopoulos, B. Nebel, and F. Toni. On the computational complexity of assumption-based argumentation for default reasoning. *Artif. Intell.*, 141(1):57–78, 2002. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(02)00245-X. 5.9

M. B. Do and S. Kambhampati. Planning as constraint satisfaction: solving the planning graph by compiling it into csp. *Artif. Intell.*, 132:151–182, November 2001. ISSN 0004-3702. doi: 10.1016/S0004-3702(01)00128-X. URL `http://dl.acm.org/citation.cfm?id=505928.505930`. 8.3

J. Doyle. *A truth maintenance system*, pages 259–279. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987. ISBN 0-934613-45-1. URL `http://dl.acm.org/citation.cfm?id=42641.42661`. 8.1

R. Drechsler, N. Drechsler, and W. Günther. Fast exact minimization of BDDs. In *DAC '98: Proceedings of the 35th Annual Conference on Design Automation*, pages 200–205, New York, NY, USA, 1998. ACM. ISBN 0-89791-964-5. doi: http://doi.acm.org/10.1145/277044.277099. 2.2.1

P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77:321–357, September 1995. ISSN 0004-3702. doi: 10.1016/0004-3702(94)00041-X. URL `http://dl.acm.org/citation.cfm?id=220193.220197`. 1.1.3, 5.9, 7, 7.1, 7.2, 7.4.1, 7.4.1, 7.4.2, 7.4.2, 7.5, 8.4

P. M. Dung, R. Kowalski, and F. Toni. Dialectic proof procedures for assumption-based, admissible argumentation. *Journal of Artificial Intelligence*, 170(2):114–159, 2006. 7.4.1

E. H. Durfee. Distributed problem solving and planning. In G. Weiß, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 121–164. MIT Press, Cambridge, MA, USA, 1999. 8.3

E. H. Durfee and V. R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, - 1991. 1.1.2

S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *Recent Advances in AI Planning. 5th European Conference on Planning (ECP'99)*, volume 1809 of *Lecture Notes in Artificial Intelligence*, pages 135–147, New York, 1999. Springer-Verlag. 2.4.3, 2.4.4, 3.1.2, 1

M. Elvang-Gøransson and A. Hunter. Argumentative logics: Reasoning with classically inconsistent information. *Data Knowledge Engineering*, 16(2):125–145, 1995. 7.4.3

E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of theoretical computer science (vol. B)*, chapter Temporal and modal logic, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990. ISBN 0-444-88074-7. URL `http://portal.acm.org/citation.cfm?id=114891.114907`. 3.4

K. Erol, J. Hendler, and D. S. Nau. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1123–1128, Seattle, Washington, USA, 1994. AAAI Press/MIT Press. ISBN 0-262-51078-2. 1.1.2, 5.1, 8.2

K. Eshghi and R. A. Kowalski. Abduction compared with negation by failure. In *Logic Programming, Proceedings of the Sixth International Conference*, pages 234–254, 1989. 7.4.3

I. A. Ferguson. TouringMachines: An architecture for dynamic, rational, mobile agents. Technical Report UCAM-CL-TR-273, University of Cambridge, Computer Laboratory, Nov. 1992. URL `http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-273.pdf`. 1.1.1, 5.1

R. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 608–620, 1971. 1.1.1, 1.1.2, 5.1, 5.2, 8.1, 1, 8.2

A. J. Garcia and G. R. Simari. Defeasible logic programming: an argumentative approach. *Theory and Practice of Logic Programming*, 4(2):95–138, 2004. ISSN 1471-0684. doi: http://dx.doi.org/10.1017/S1471068403001674. 1.1.3, 7.14

D. R. García, A. J. García, and G. R. Simari. Defeasible reasoning and partial order planning. In *Proceedings of the 5th international conference on Foundations of information and knowledge*

systems, FoIKS'08, pages 311–328, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-77683-4, 978-3-540-77683-3. URL `http://dl.acm.org/citation.cfm?id=1786094.1786117`. 1.1.3, 1.3, 5.1, 7.14

M. Gelfond and V. Lifschitz. Action languages. *Electronic Transactions on AI*, 3, 1998. 8.1

M. R. Genesereth and N. J. Nilsson. *Logical foundations of artificial intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987. ISBN 0-934613-31-1. 1.1.1, 5.1

M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *AAAI*, pages 677–682, 1987. 5.1

M. Ghallab, C. K. Isi, S. Penberthy, D. E. Smith, Y. Sun, and D. Weld. PDDL - the planning domain definition language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998. 1.1.2, 8.1, 8.2

E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: preliminary report. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, pages 623–630. AAAI Press, 1998. 8.1

O. Grumberg, S. Livne, and S. Markovitch. Learning to order BDD variables in verification. *Journal of Artificial Intelligence Research (JAIR)*, 18:83–116, 2003. 2.2.1, 1

C. L. Hamblin. *Fallacies*. Methuen, London, 1970. 1.1.3, 7.1, 8.4

D. Harel. *First-Order Dynamic Logic*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1979. ISBN 0387092374. 2.4.3

D. Hitchcock, P. McBurney, and S. Parsons. A framework for deliberation dialogues. In *Proceedings of the Fourth Biennial Conference of the Ontario Society for the Study of Argumentation*, 2001. 8.4

R. Hojati, S. C. Krishnan, and R. K. Brayton. Early quantification and partitioned transition relations. In *ICCD '96: Proceedings of the 1996 International Conference on Computer Design, VLSI in Computers and Processors*, pages 12–19, Washington, DC, USA, 1996. IEEE Computer Society. ISBN 0-8186-7554-3. 2.2.1, 18

J. Jain, W. Adams, and M. Fujita. Sampling schemes for computing OBDD variable orderings. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM International Conference on Computer-aided Design*, pages 631–638, New York, NY, USA, 1998. ACM. ISBN 1-58113-008-2. doi: http://doi.acm.org/10.1145/288548.289099. 2.2.1

H. Jakobovits and D. Vermeir. Robust semantics for argumentation frameworks. *Journal of Logic and Computation*, 9(2):215–261, 1999. 7.1, 8.4

N. R. Jennings. *Coordination techniques for distributed artificial intelligence*, pages 187–210. John Wiley & Sons, Inc., New York, NY, USA, 1996. ISBN 0-471-006750. URL `http://dl.acm.org/citation.cfm?id=239297.239311`. 1.1.2, 8.3

R. M. Jensen. *Efficient BDD-Based Planning for Non-Deterministic, Fault-Tolerant, and Adversarial Domains*. PhD thesis, Carnegie Mellon University, June 2003. 3.4, 16, 3.6

R. M. Jensen, R. E. Bryant, and M. M. Veloso. An efficient BDD-based A* algorithm. In *Proceedings of AIPS-02 Workshop on Planning via Model Checking*, 2002a. 2.2.1

R. M. Jensen, R. E. Bryant, and M. M. Veloso. SetA*: An efficient BDD-based heuristic search algorithm. In *Proceedings of 18th National Conference on Artificial Intelligence (AAAI02)*, pages 668–673, 2002b. 2.2.1

R. M. Jensen, M. M. Veloso, and R. E. Bryant. State-set branching: Leveraging BDDs for heuristic search. *Artificial Intelligence*, 172(2-3):103–139, 2008. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/j.artint.2007.05.009. 2.2.1

R. Jhala and R. Majumdar. Software model checking. *ACM Comput. Surv.*, 41(4):1–54, 2009. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/1592434.1592438. 2.2.1

L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. *Robotics and Autonomous Systems(1&2), June 1990*, 6:35–48, 1990. 5.1

J. Kohlas. *Information Algebras: Generic Structures for Inference*. Discrete Mathematics and Theoretical Computer Science. Springer-Verlag, London, Berlin, Heidelberg, 2003a. 5, 2.5, 5

J. Kohlas. Probabilistic argumentation systems: A new way to combine logic with probability. *Journal of Applied Logic*, 1(3-4):225–253, 2003b. 2.5

R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1): 67–95, 1986. ISSN 0288-3635. 1.1.2, 5.1, 8.1

R. E. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal on Computing*, 6(3):467–480, 1977. doi: 10.1137/0206033. 2.2.1, 18

J. Langel and J. Kohlas. A semantic theory of propositional information. Technical Report 05-15, Department of Informatics, University of Fribourg, 2005. URL `http://diuf.unifr.ch/tcs/publications/ps/langelkohlas05.pdf`. 2.5, 2.5.4, 2.5.5

V. R. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. E. Neiman, R. M. Podorozhny, M. V. N. Prasad, A. Raja, R. Vincent, P. Xuan, and X. Zhang. Evolution of the GPGP/TÆMS domain-independent coordination framework. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):87–143, 2004. 1.1.2, 8.3

H. J. Levesque. A logic of implicit and explicit belief. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 198–202, 1984. 1.1.1, 5.1

V. Lifschitz. On the semantics of strips. In M. Georgeff, Lansky, and Amy, editors, *Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufmann, San Mateo, CA, 1987. 1.1.2, 8.2

F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5):655–677, 1994. doi: 10.1093/logcom/4.5.655. URL `http://logcom.oxfordjournals.org/content/4/5/655.abstract`. 1.1.2, 5.1, 8.1

D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the ninth National conference on Artificial intelligence - Volume 2*, AAAI'91, pages 634–639. AAAI Press, 1991. ISBN 0-262-51059-6. URL `http://dl.acm.org/citation.cfm?id=1865756.1865775`. 1.1.2, 5.1, 8.2

J. McCarthy. Situations, actions, and causal laws. Technical Report AIM-2, Artificial Intelligence Project, Stanford University, 1963. 1.1.2, 5.1, 8.1

J. McCarthy. Epistemological problems of artificial intelligence. In *Proceedings of the 5th international joint conference on Artificial intelligence - Volume 2*, pages 1038–1044, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc. URL `http://dl.acm.org/citation.cfm?id=1622943.1623044`. 1.1.2, 5.1, 8.1

J. McCarthy. *Circumscription — a form of non-monotonic reasoning*, pages 145–152. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987. ISBN 0-934613-45-1. URL `http://dl.acm.org/citation.cfm?id=42641.42653`. 1.3, 5.1, 7.4.3, 8.1

J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969. reprinted in McC90. 1.1.2, 5.1, 8.1

D. McDermott. Nonmonotonic logic II: Nonmonotonic modal theories. *Journal of the ACM (JACM)*, 29(1):33–57, 1982. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/322290.322293. 7.4.3

D. McDermott and J. Doyle. *Non-monotonic logic I*, pages 111–126. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987. ISBN 0-934613-45-1. URL `http://dl.acm.org/citation.cfm?id=42641.42650`. 8.1

C. Meinel and T. Theobald. *Algorithms and Data Structures in VLSI Design*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998. ISBN 3540644865. 2.2.1

F. Meneguzzi, Y. Tang, K. Sycara, and S. Parsons. On representing planning domains under uncertainty. In *Proceedings of the Third Annual Conference of the ITA*, Imperial College, London, 2010. 8.2

F. Meneguzzi, Y. Tang, K. Sycara, and S. Parsons. An approach to generate MDPs using HTN representations. In *IJCAI Workshop on Decision Making in Partially Observable Uncertain Worlds: Exploring Insights from Multiple Communities*, Barcelona, Spain, July 2011. 8.2

I.-H. Moon, J. H. Kukula, K. Ravi, and F. Somenzi. To split or to conjoin: the question in image computation. In *DAC '00: Proceedings of the 37th conference on Design automation*, pages 23–28, New York, NY, USA, 2000. ACM. ISBN 1-58113-187-9. doi: http://doi.acm.org/10.1145/337292.337305. 2.2.1

R. C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94, 1985. 7.4.3

J. P. Mller and M. Pischel. Modelling interacting agents in dynamic environments. In *ECAI*, pages 709–713, 1994. 5.1

V. Nannen. A short introduction to Kolmogorov complexity, April 2003. URL `http://volker.nannen.com/pdf/short_introduction_to_kolmogorov_complexity.pdf`. 2.2.1

D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. ISBN 1558608567. 3.1.2, 3.6

A. L. Oliveira and A. Sangiovanni-Vincentelli. Using the minimum description length principle to infer reduced ordered decision graphs. *Machine Learning*, 25(1):23–50, 1996. ISSN 0885-6125. doi: http://dx.doi.org/10.1023/A:1018344122010. 2.2.1

S. Panda, F. Somenzi, and B. F. Plessier. Symmetry detection and dynamic variable ordering of decision diagrams. In *ICCAD '94: Proceedings of the 1994 IEEE/ACM International Conference on Computer-aided Design*, pages 628–631, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press. ISBN 0-89791-690-5. 2.2.1

S. Parsons, C. Sierra, and N. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998. 8.4

S. Parsons, Y. Tang, K. Cai, E. Sklar, and P. McBurney. Some thoughts on using argumentation to handle trust. In *Proceedings of the 12th International Workshop on Computational Logic in Multi-Agent Systems*, Barcelona, 2011a. 9.2

S. Parsons, Y. Tang, E. Sklar, K. Cai, and P. McBurney. Argumentation-based reasoning in agents with varying degrees of trust. In *Proceedings of the Tenth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2011b. 9.2

J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984. ISBN 0-201-05594-5. 2.2.1

E. P. D. Pednault. Formulating Multi-Agent Dynamic-World problems in the classical planning framework. In M. P. Georgeff and A. L. Lansky, editors, *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop*, pages 47–82, San Mateo, CA, 1987. Morgan Kaufmann Publishers. 2.4.3

E. P. D. Pednault. ADL: Exploring the middle ground between strips and the situation calculus. In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, pages 324–332, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. ISBN 1-55860-032-9. 8.1

E. P. D. Pednault. ADL and the state-transition model of action. *Journal of Logic and Computation*, 4(5):467–512, 1994. URL `http://logcom.oxfordjournals.org/content/4/5/467`. 8.1

J. S. Penberthy and D. S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In B. Nebel, C. Rich, and W. Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 103–114. Morgan Kaufmann, San Mateo, California, 1992. 1.1.2, 5.1, 8.2

M. Petrik and S. Zilberstein. A bilinear programming approach for multiagent planning. *Journal of Artificial Intelligence Research*, 35:235–274, 2009. 1.1.2, 8.3

J. L. Pollock. How to reason defeasibly. *Artificial Intelligence*, 57(1):1–42, 1992. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/0004-3702(92)90103-5. 7.4.1

J. L. Pollock. Defeasible reasoning with variable degrees of justification. *Artificial Intelligence*, 133 (1-2):233–282, 2001. 7.1, 8.4

D. Poole, R. Goebel, and R. Aleliunas. Theorist: A logical reasoning system for defaults and diagnosis. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 331–352. Springer, 1987. 7.4.3

H. Prakken. On dialogue systems with speech acts, arguments, and counterarguments. In *JELIA '00: Proceedings of the European Workshop on Logics in Artificial Intelligence*, pages 224–238, London, UK, 2000. Springer-Verlag. ISBN 3-540-41131-3. 8.4

A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991. ISBN 1-55860-165-1. 1.1.1, 5.1

A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995. 6.2

R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980. 1.3, 5.1, 7.4.3, 8.1

R. Reiter. *The frame problem in situation the calculus: a simple solution (sometimes) and a completeness result for goal regression*, pages 359–380. Academic Press Professional, Inc., San Diego, CA, USA, 1991. ISBN 0-12-450010-2. URL `http://dl.acm.org/citation.cfm?id=132218.132239`. 8.1

R. Reiter. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT Press, 2001. ISBN 9780262182188. 1.1.2, 1.3, 5.1

S. Rosenschein and L. Kaelbling. The synthesis of digital machines with provable epistemic properties. In *Proceedings of the 1986 Conference on Theoretical aspects of reasoning about knowledge*, pages 83–98, San Francisco, CA, USA, 1986. Morgan Kaufmann Publishers Inc. ISBN 0-934613-04-4. URL `http://dl.acm.org/citation.cfm?id=22628.22634`. 5.1

J. R. Searle. *Rationality in Action*. MIT Press, 2001. 7.14

G. R. Simari, A. J. García, and M. Capobianco. Actions, planning and defeasible reasoning. In J. P. Delgrande and T. Schaub, editors, *NMR*, pages 377–384, 2004. ISBN 92-990021-0-X. 1.3, 5.1

S. Soderland and D. S. Weld. Evaluating nonlinear planning. Technical Report TR-91-02-03, Department of Computer Science & Engineering, University of Washington, 1991. 1.1.2, 5.1, 8.2

K. Sycara. Multiagent systems. *AI Magazine*, 10(2):79–93, 1998. 1

Y. Tang. Integrating multiagent dialogues, planning and plan execution. In *20th International Conference on Automated Planning and Scheduling Doctoral Consortium*, Toronto, Canada, 2010. 1.4, 9.2

Y. Tang and S. Parsons. Argumentation-based dialogues for deliberation. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 552–559, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-093-0. 1.4, 8.4, 9.2

Y. Tang and S. Parsons. Using argumentation-based dialogues for distributed plan management. In *Proceedings of the AAAI Spring Symposium on Distributed Plan and Schedule Management*, Stanford, 2006. 9.2

Y. Tang and S. Parsons. A dialogue mechanism for public argumentation using conversation policies. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 445–452, Estoril, Portugal, May 12-16 2008. 9.2

Y. Tang and S. Parsons. An MDP model for planning team actions with communication. Technical report, International Technology Alliance in Network and Information Science, 2009. URL `https://www.usukitacs.com/?q=node/5268`. 9.2

Y. Tang, T. J. Norman, and S. Parsons. Agent-based dialogues to support plan execution by human teams. In *Proceedings of the Second Annual Conference of the ITA*, Imperial College, London, 2008. 9.2

Y. Tang, T. J. Norman, and S. Parsons. A model for integrating dialogue and the execution of joint plans. In *Proceedings of the Eigth International Joint Conference on Autonomous Agents and Multiagent Systems*, Budapest, Hungary, May 10-15 2009. 1.4, 9.2

Y. Tang, K. Cai, E. Sklar, P. McBurney, and S. Parsons. A system of argumentation for reasoning about trust. In *Proceedings of the 8th European Workshop on Multi-Agent Systems*, Paris, France, December 2010a. 9.2

Y. Tang, T. J. Norman, and S. Parsons. Computing argumentation in polynomial number of bdd operations: A preliminary report. In *Seventh International Workshop on Argumentation in Multiagent Systems*, 2010b. 1.4

Y. Tang, K. Cai, E. Sklar, P. McBurney, and S. Parsons. Using argumentation to reason about trust and belief. *Journal of Logic and Computation*, 2011a. (to appear). 5.7, 9.2

Y. Tang, F. Meneguzzi, S. Parsons, and K. Sycara. Probabilistic hierarchical planning over markov decision processes. In *Proceedings of the Tenth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2011b. (extended abstract). 5.9, 6.4, 7.14, 8.2, 8.5, 9.2

Y. Tang, F. Meneguzzi, K. Sycara, and S. Parsons. Planning over MDPs through probabilistic HTNs. In *AAAI 2011 Workshop on Generalized Planning*, San Francisco, August 2011c. 5.9, 6.4, 8.2, 9.2

M. Thielscher. Introduction to the fluent calculus. *Linköping Electronic Articles in Computer and Information Science (http://www.ep.liu.se/ea/cis/1998/14/)*, 3(14), Oct. 1998. 8.1

M. Thielscher. From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111(1-2):277 – 299, 1999. ISSN 0004-3702. doi: 10.1016/S0004-3702(99)00033-8. URL `http://www.sciencedirect.com/science/article/pii/S0004370299000338`. 8.1

M. Thielscher. *Challenges for action theories*. Springer-Verlag, Berlin, Heidelberg, 2000. ISBN 3-540-67455-1. 1.1.2, 5.1, 8.1

S. E. Toulmin. *The Uses of Argument*. Cambridge University Press, 1958. ISBN 0521534836. 1.1.3, 7.1, 8.4

S. E. Toulmin, R. D. Rieke, and A. Janik. *An introduction to reasoning*. Collier Macmillan Publishers, New York London: Macmillan, 2nd ed. edition, 1984. 1.1.3, 7.1, 8.4

Q. B. Vo and N. Y. Foo. Reasoning about action: an argumentation-theoretic approach. *Journal of Artificial Intelligence Research*, 24:465–518, October 2005. ISSN 1076-9757. URL `http://dl.acm.org/citation.cfm?id=1622519.1622532`. 1.1.2, 1.1.3, 1.3, 5.1, 7.14, 8.1

D. Walton. Argumentation theory: A very short introduction. In G. Simari and I. Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 1–22. Springer US, 2009. ISBN 978-0-387-98197-0. URL `http://dx.doi.org/10.1007/978-0-387-98197-0_1`. 7.1, 8.4

D. N. Walton. *Argument Schemes for Presumptive Reasoning*. Lawrence Erlbaum Associates, Mahwah, NJ, USA, 1996. 1.1.3, 7.14

D. N. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. State University of New York Press, Albany, NY, USA, 1995. 1.1.3, 7.1, 8.4

M. Wooldridge. Agent-based software engineering. *IEE Proceedings - Software*, 144(1):26–37, 1997. 1.1.1

M. Wooldridge. *Introduction to MultiAgent Systems*. John Wiley and Sons, 2002. ISBN 047149691X. 1, 5.1

M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995. 1.1.1, 1.3, 5.1

M. Wooldridge, M.-P. Huget, M. Fisher, and S. Parsons. Model checking for multiagent systems: the mable language and its applications. *International Journal on Artificial Intelligence Tools*, 15(2):195–226, 2006. 5

B. Yang, R. E. Bryant, D. R. O'Hallaron, A. Biere, O. Coudert, G. Janssen, R. K. Ranjan, and F. Somenzi. A performance study of BDD-based model checking. In *FMCAD '98: Proceedings of the Second International Conference on Formal Methods in Computer-Aided Design*, pages 255–289, London, UK, 1998. Springer-Verlag. ISBN 3-540-65191-8. 2.2.1