Representing and Reasoning about Goals and Policies of Agents

by

Jicheng Zhao

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved November 2010 by the
Graduate Supervisory Committee:

Chitta Baral, Chair
Subbarao Kambhampati
Joohyung Lee
Vladimir Lifschitz
Huan Liu

ARIZONA STATE UNIVERSITY

December 2010

ABSTRACT

Goal specification is an important aspect of designing autonomous agents. A goal does not only refer to the set of states for the agent to reach. A goal also defines restrictions on the paths the agent should follow. Temporal logics are widely used in goal specification. However, they lack the ability to represent goals in a non-deterministic domain, goals that change non-monotonically, and goals with preferences. This dissertation defines new goal specification languages by extending temporal logics to address these issues.

First considered is the goal specification in non-deterministic domains, in which an agent following a policy leads to a set of paths. A logic is proposed to distinguish paths of the agent from all paths in the domain. In addition, to address the need of comparing policies for finding the best ones, a language capable of quantifying over policies is proposed. As policy structures of agents play an important role in goal specification, languages are also defined by considering different policy structures.

Besides, after an agent is given an initial goal, the agent may change its expectations or the domain may change, thus goals that are previously specified may need to be further updated, revised, partially retracted, or even completely changed. Non-monotonic goal specification languages that can make these changes in an elaboration tolerant manner are needed. Two languages that rely on labeling subformulas and connecting multiple rules are developed to address non-monotonicity in goal specification.

Also, agents may have preferential relations among sub-goals, and the preferential relations may change as agents achieve other sub-goals. By nesting a comparison operator with other temporal operators, a language with dynamic preferences is proposed.

Various goals that cannot be expressed in other languages are expressed in the proposed languages. Finally, plans are given for some goals specified in the proposed languages.

This work is dedicated to my parents.

ACKNOWLEDGEMENTS

This work would not be possible without my mentors, colleagues, and friends.

I would like to thank Prof. Chitta Baral, my supervisor, for his help through the years and his collaboration on all the work in this dissertation. Without his help and guidance, I would know much less about research.

I would like to thank Profs. Thomas Eiter, Vladimir Lifschitz, Tran Cao Son, and Yan Zhang for their help and collaboration in the past a few yaers. The work in Chapter 6 evolved from the collaboration with Prof. Thomas Eiter. Prof. Vladimir Lifschitz gave valuable advice on representing goals in non-deterministic domains. The work on complexities of temporal logics π-CTL* and P-CTL* was enhanced by discussion with Prof. Tran Cao Son. An early version of the work on preferences in goal specification was done with Prof. Yan Zhang.

I thank Profs. Subbarao Kambhampati, Joohyung Lee, Vladimir Lifschitz, and Huan Liu for serving as committee members.

I would also like to thank many of my colleagues who have given me advice and helped me in different ways during my time as a graduate student. They are Luis Tari, Xin Zhang, William Cushing, Nam Tran, Zheshen Wang, Yunsong Meng, Marcos Alvarez Gonzalez, Shanshan Liang, Ravi Palla, Võ Hà Nguyên, Yan Qi, Jörg Hakenberg, Lei Tang, Hui Zhang, Yang Qin, Hairong Xie, Ling Zhou, Zhibin Zhou, Yin Yin, Jiayu Zhou, Juraj Dzifcak, Jianhui Chen, Nan Li, Guofeng Deng, Wenlan Jing, Tie Wang, Jiancun Fan, Ning He, and Jian Tang.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Reasoning about actions and their effects in changing the environment is an important aspect in designing autonomous agents. Systematic design of semi-autonomous agents involves specifying (i) domain description: actions that an agent can do, their impacts, environment, and etc.; (ii) control execution of an agent; and (iii) directives for an agent. There has been a large body of work on (i) [FN71, Ped87, GL98a] and a significant amount of work on (ii) [Sch87, BG00, DLPT02, BDH99]. However, there has been relatively less work on (iii), which is often referred to as goal specification. This dissertation focuses on (iii) and its relations with other components in agent design. In specifying goals of an agent, temporal logics are widely used. However, there are many interesting goals which cannot be expressed using existing temporal logics. For example, a lot of interesting goals in non-deterministic domains cannot be represented. Existing temporal logics are not able to handle elaboration tolerance in goal specification. Also, existing logics are not able to represent preferences that may change dynamically in specifying goals of an agent. This dissertation extends existing temporal logics in different directions to cover them. The following section begins with the components in designing a semi-autonomous agent.

1.1 Agent Design

In designing an agent, components considered are the environment of an agent, the ability of an agent, and the requirements for an agent.

The environment of an agent is often modeled as a transition graph. It defines states and actions in a domain, the value of each fluent in each state, and the effects of executing an action in each state. In planning community, STRIPS [FN71],

ADL [Ped89], and PDDL [McD00] are defined to model effects of actions. In reasoning about actions community, different mechanisms such as state calculus [MH69, Rei91], event calculus [KS86], action language [GL98b], fluent calculus [Thi98] are proposed to precisely represent the transition graph, especially for modeling the frame problem [MH69]. These logics are also extended to model different properties of a reasoning task or different properties of a domain.

The second aspect in automatic agent design is to define the ability of an agent. Given a transition graph, ability of an agent is often modeled as a policy program, or a policy structure of the agent. For example, in a deterministic domain, each agent might execute a sequence of actions to achieve its goal. It may also take actions by following a mapping from states to actions. In a non-deterministic domain, besides the two definitions above, a policy may also be a mapping from sequences of states to actions [HF85, AHK02, BDH99]. In a multi-agent setting, actions taken by one agent may depend on its knowledge about the domain and other agents. In a domain where agents have sensing actions, actions taken by one agent may depend on other properties of the transition graph. There are also other definitions of the policy structure, where each definition of the policy structure specifies the ability of an agent. Once the ability of an agent is specified, a plan or a policy of the agent is one instance of its policy structure.

Given an agent with a policy structured defined, the agent executes in an environment leads to a structure of states. Requirements users have for the agent are then defined as goals. A goal is not necessarily about a set of states for the agent to reach. It may also define how the agent behaves before reaching one of the desired states. Given that a trajectory is a sequence of states, each goal in a goal specification language specifies a trajectory, or a set of trajectories resulted from executing a plan or a policy in the domain. Each goal of an agent distinguishes the set of desired policies from undesired ones for the agent. This is necessary in designing

autonomous agents, as often an agent needs to be given a directive – a high level goal specification – regarding the behavior desired from it. Directives given to the agent may not easily be described. The following section elaborates on why a goal specification language is needed and why existing goal specification languages are not adequate in representing some interesting requirements for the agent.

1.2 Goal Specification with Temporal Logics

In most cases, a goal is considered as a set of states satisfying some properties. An agent satisfies a goal if it finds a path in the transition graph to one of the states. Currently, most planners in the planning community are trying to find a plan to reach one of the states.

However, besides reaching a set of states, there are other requirements for an agent. The need of specifying goals of an agent was first proposed in [McC59, MH69]. Since then, comparing to other components in autonomous agent design, there is relatively less work in the goal specification aspect of agent design. Temporal logics such as LTL [Pnu77] and CTL* [EC82, ES89, Eme90] are introduced in representing goals of an agent.

If an agent is to reach a state, there are requirement on how the state is reached. If an agent is asked to maintain some properties, there is no final states to reach. In both cases, linear temporal logic (LTL) can be used in specifying properties of a sequence of states of reaching a state or maintaining a configuration.

Also, there are requirements on other paths other than the path taken by the agent. For example, as the agent driving from a city to the other city, it may be required to have a gas station in 4 miles at any time before arriving in the destination. Note that the 4 miles driving to the gas station may not on the main path of the agent. Branching time logics such as CTL and CTL* can be used to deal with such branches in goal specification.

Many extensions of linear and branching time temporal logics are used for goal specification. For example, LTL and CTL* are extended to have metric intervals [BK98] or qualitative measures on elapsed time between the occurrences of the events [Pnu77]. Also in a timed transition system, by defining a cost function on CTL states and paths, min-max CTL [DCDS01] was proposed by allowing the quantification of CTL states and paths. Languages ATL and ATL* [AHK02] extend LTL and CTL* to game-like multi-agent systems to quantify over paths of each agent. Besides these, temporal logics are often suggested for specifying non-Markovian rewards [BBG96, BBG97, TGS$^+$06] in the decision theoretic planning community.

Most of the temporal logics mentioned above were developed in the context of program specification and model checking [HNSY92]. This dissertation shows that in representing goals of an agent, some properties of goal specifications that are not exist in model checking need to be addressed. For example, there are no goals of the kind "trying one's best" in specifying a program while a user may have a goal for an agent to try its best to reach a state.

In defining a goal specification language, the first question is that "what is a goal?". A goal is considered as a mapping from possible trajectories for the agent to choose to sets of trajectories (or sets of set of trajectories) chosen by the agent, where each trajectory is a sequence of states. A plan of an agent satisfies one goal if for all domains, the trajectory (or the set of trajectories) of the plan is one element in the set.

There are some limitations of existing goal specification languages. This dissertation addresses these limitations by extending existing goal specification languages. CTL* is extended to π-CTL* and P-CTL* to capture goals in non-deterministic domains. Languages N-LTL and ER-LTL are proposed to address non-monotonicity in goal specification. Also, language Pref-π-CTL* is proposed to address goals with

dynamic preferences.

The following sections elaborate on the importance of the proposed extensions.

Goal Specification in Non-deterministic Domains

Goal specifications in a non-deterministic domain are firstly considered. Non-deterministic domains have some properties that are not captured in existing languages. For example, in a non-deterministic domain, due to non-deterministic effects of actions, each plan leads to a set of trajectories. To capture properties of the plan, Dal Lago, Pistore, and Traverso [DLPT02] suggest that trajectories in the plan need to be distinguished from all trajectories in the domain. Instead of defining a new language as in [DLPT02], by extending CTL*, a language π-CTL* is proposed in this dissertation based on the same observation. Also, as there are multiple plans in a domain, it is necessary to compare properties of plans before choosing a particular set of plans among them. The way of comaring plans is referred to as quantifying over policies [AHK02]. This dissertation proposes language P-CTL* to capture the motivation of quantifying over policies for goal specification. These two languages are discussed in Chapter 3. Now, the transition graph in Figure 1.1 is illustrated to show the importance of policies in defining a goal specification language.

Assuming that the agent is in state s_1, the goal of the agent is to "try its best" to reach a state where p is true. In order to achieve "trying its best", the agent should be able to know all policies it has, and should be able to compare them. The goal of "trying its best" may have an interpretation that "if there is a policy in the domain with some desired properties, the policy taken by the agent should have such properties". In state s_1 of the transition graph in Figure 1.1, as there is a policy which is a mapping from states to actions to reach p, the policy that takes action a_6 in state s_1 is not trying its best in reaching p. Similarly, properties of policies can

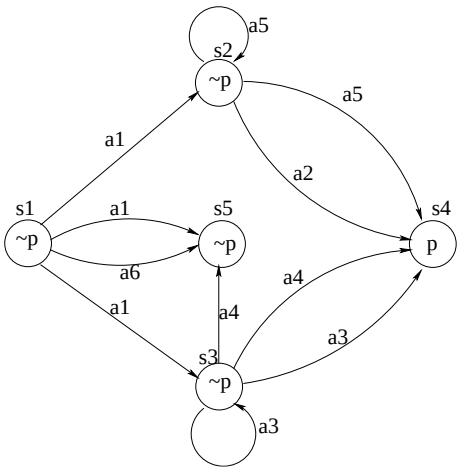

Figure 1.1: Transition diagram in a non-deterministic domain

be compared so as to find the most preferred policy. In particular, this dissertation shows that by grouping trajectories in the same policy, and by quantifying over policies, any policy in this example can be distinguished from other policies by comparing different properties policies have.

Non-monotonicity in Goal Specification

After an agent is given an initial goal, the agent may change its expectations or the domain may change. Thus the agent may modify, enhance, or discard previously specified goals. The agent may also make one change after another on its initial goal. The following example illustrates the needs of non-monotonicity in goal specification.

Example 1. *John has an agent in his office that does errands for him. John may ask the agent to bring him some coffee. But soon he realizes that the coffee machine was broken. He is not sure if the machine has been fixed or not. He then revises his directive to the agent telling it that if the coffee machine is still broken then a cup of tea would be fine. Just after that he gets a call from a colleague who says that he had called a coffee machine company and asked them to deliver a new coffee*

machine. Then John calls up the agent and tells it that if the new coffee machine is already there then it should bring him coffee. (Note that the old coffee machine may still be broken.) He also remembers that he takes sugar with his tea and that the tea machine has various temperature settings. So he tells the agent that if it is going to bring tea then it should bring him a pack of sugar and set the tea machine setting to "very hot".

To represent these goals, a non-monotonic goal specification language that enables the agent to modify its goals in an elaboration tolerant manner is required. While there have been a lot of non-monotonic logics such as logic programs [GL88], default logic [Rei87], autoepistemic logic [Moo85], only two papers [FH91, Sae87] are found on defining non-monotonic logics for goal specification but neither work addresses the elaboration tolerant issue in goal specification. It is a question whether these non-monotonic logics can be applied directly to temporal formulas. Chapter 4 of this dissertation proposes languages N-LTL and ER-LTL. Each goal in the languages is represented as a set of rules. Similar to defeasible logic [Nut87], subformulas in a rule are defeated if there are exceptions defined for the sub-formula. The idea of completion is used to capture all possible exceptions for a sub-formula. Labels are used to combine multiple rules to one temporal formula. The idea of completion was used for defining exceptions. Reiter's idea of a surface non-monotonic logic [Rei01] that gets compiled into a more tractable standard logic is used and thus avoid increase in complexity. With these techniques borrowed from other languages, each step in Example 1 can be represented in the proposed ER-LTL language. Chapter 4 of this dissertation proposes these languages and gives an approach of progressing goals after the agent has executed part of its plan.

Preferences in Goal Specification

The third direction considered in this dissertation is the goal specification with preferences. In specifying goals of an agent, users often have different preference relations among sub-goals. Users may have some preference relations under one condition but other preference relations under other conditions. The preference relation may also change dynamically as the agent proceeds with its current plan. Thus a goal specification language capable of handing dynamic preferences is needed. Son and Pontelli define a preference relation in goal specification language $\mathscr{PP}$ [SP06]. However, it only works for deterministic domain and it does not allow changes in the preference relations. The following example illustrates the needs of dynamic preferences and motivates our language Pref-π-CTL*.

Consider that a user has a goal for the agent that provides the user with plans of commuting between her home and the workplace. The user can either walk, take a bus, or hire a taxi to go from her home to the workplace. She has objectives of going to the workplace on time, spending less money on the trip, and other objectives such as keeping warm and dry. With these preferences relations among sub-goals in mind, if she gets up late, she may hire a taxi to avoid being late. If it is raining outside, she does not want to walk. The agent that make plans for this user needs to determine the preference relations among these sub-goals based on the current state of the user. For example, if it starts raining while the user walking to the workplace, the preference relation might change. In order to handle all these dynamics, the agent should be able to adjust its goal such that a sub-goal a is preferred to the other goal b under some conditions, but b is preferred to a under other conditions. Note that whether a sub-goal is preferred over other sub-goals is not determined in the initial state when the agent is deployed. The agent can only know the preference relations among sub-goals after the user has executed part of

her plan, as the execution of the plan may lead the user to a state that is different from the inital state.

In Chapter 5, language Pref-π-CTL* is proposed to represent goals with dynamic preferences. The language enables users to represent goals consist of dynamic preferences among sub-goals.

1.3 Planning

After a goal in a goal specification language is specified, it is a still a challenge problem to find planning algorithms for the goal. It is difficult, if possible, to find general planning heuristics for goals specified in these languages. Chapter 6 takes some specific goals in π-CTL* such as strong, weak, strong-cyclic plans [CPRT03] and their variations, and find plans by utilizing the approach proposed in [BEBN08] that first encodes the planning problem as a Reverse-Horn SAT problem, further translates the encoding to a Horn SAT, and then derives a polynomial time algorithm by simulating the way of solving the Horn SAT. Logic program implementations of these plans and a plan that "tries its best" to reach states satisfying some conditions are proposed.

1.4 Outline of Contributions

Different extensions of temporal logics are studied in Chapters 3 to 6. In particular, the main contributions of the dissertation are as follows:

- **Extending temporal logics to non-deterministic domains**: Chapter 3 gives a formal definition on "what is a goal". Languages π-CTL* is proposed to distinguish the set of trajectories of the agent and all trajectories in the domain. Language P-CTL* is proposed by quantifying over policies. The consideration of policies plays an important role in goal specifications in non-deterministic domains. The definition of a policy also impacts the set of goals expressed in the language, thus languages defined with different definitions

of the policy structured are proposed. An approach on proving a goal cannot be expressed in a language is proposed. The set of goals expressed in goal specification languages are then formally compared.

- **Non-monotonic extension of temporal logics**: Chapter 4 presents logics N-LTL and ER-LTL that enable users in revising goals in an elaboration tolerant manner. The way of progressing a goal is proposed to deal with the case that the agent has already executed some actions. Also, an algorithm of translating an ER-LTL goal to an equivalent LTL formula is implemented.

- **Extending temporal logics with preference**: Chapter 5 presents a logic Pref-π-CTL* with dynamic preference relations defined among sub-goals. This dynamic preference relation enables users in representing goals with different preference relations in different states.

- **Planning with goal specified in proposed languages**: Chapter 6 studies the planning problems for some goals specified in proposed languages. The approach in [BEBN08] is used for strong, weak, strong cyclic planning [CPRT03], and their variations. A logic program implementation of a plan that "tries its best" in reaching a set of states is also given.

1.5 Dissertation Organization

The rest of the dissertation is organized as follows: Chapter 2 introduces temporal logics LTL and CTL*. All logics defined in the dissertation are extended from these logics. An approach in [BEBN08] that find plans for a k-maintainability problem is also discussed. Some planning algorithms in Chapter 6 are based on a similar approach. Different extensions of temporal logics are studied in Chapters 3 to 6. In Chapter 3, languages for goal specifications in non-deterministic domains are proposed. Chapter 4 studies non-monotonic goal specification languages. In

Chapter 5, a language for representing preferences in specifying goals is proposed. In Chapter 6, planning algorithms for some goals in proposed languages are studied. The dissertation is concluded with a summary and future directions in Chapter 7.

Chapter 2

BACKGROUND

This chapter contains some background materials used in later chapters. It starts with linear temporal logic LTL and branching time temporal logic CTL*. Logics proposed in later chapters are based on these two logics. In the second part of this chapter, a planner for the *k*-maintainability [BEBN08] problem is reviewed. Some planning algorithms in Chapter 6 are proposed by following the same approach.

2.1 Background: LTL and CTL*

As languages proposed in this dissertation rely a lot on existing temporal logics, in this section existing formulations [ES89, Eme90, BK98, NS00, BKT01, PT01] of specifying goals using linear and branching time temporal logics are discussed. This section starts with goal specification using the linear temporal logic from [BK98, BKT01].

Goal Representation Using LTL

The syntax of the language is now discussed. Syntactically, LTL formulas are made up of propositions, propositional connectives $\vee$, $\wedge$, and $\neg$, and future temporal connectives $\bigcirc$, $\Box$, $\Diamond$ and U.

Definition 1. *Let $\langle p \rangle$ be an atomic proposition, $\langle f \rangle$ be an LTL formula. LTL formulas are defined as follows:*

$$\langle f \rangle ::= \langle p \rangle | \langle f \rangle \wedge \langle f \rangle \mid \langle f \rangle \vee \langle f \rangle \mid \neg \langle f \rangle \mid \bigcirc \langle f \rangle \mid \Box \langle f \rangle \mid \Diamond \langle f \rangle \mid \langle f \rangle \mathsf{U} \langle f \rangle$$

□

A *trajectory* is an infinite sequence of states. The truth of an LTL formula is defined with respect to a trajectory and a reference state.

Definition 2. *Let σ given by $s_0, s_1, \ldots, s_k, s_{k+1}, \ldots$ be a trajectory, p be an atomic proposition, s_j be a state, and f and f_is be LTL formulas.*

- $(s_j, \sigma) \models p$ *iff* p *is true in* s_j.
- $(s_j, \sigma) \models \neg f$ *iff* $(s_j, \sigma) \not\models f$.
- $(s_j, \sigma) \models f_1 \vee f_2$ *iff* $(s_j, \sigma) \models f_1$ *or* $(s_j, \sigma) \models f_2$.
- $(s_j, \sigma) \models f_1 \wedge f_2$ *iff* $(s_j, \sigma) \models f_1$ *and* $(s_j, \sigma) \models f_2$.
- $(s_j, \sigma) \models \bigcirc f$ *iff* $(s_{j+1}, \sigma) \models f$.
- $(s_j, \sigma) \models \Box f$ *iff* $(s_k, \sigma) \models f$, *for all* $k \geq j$.
- $(s_j, \sigma) \models \Diamond f$ *iff* $(s_k, \sigma) \models f$, *for some* $k \geq j$.
- $(s_j, \sigma) \models f_1 \ \mathsf{U} \ f_2$ *iff there exists* $k \geq j$ *such that* $(s_k, \sigma) \models f_2$ *and for all* i, $j \leq i < k$, $(s_i, \sigma) \models f_1$. □

The notion of trajectories consistent with an initial state and a transition function is defined so as to specify goals of an agent in LTL.

Definition 3 (Trajectory of a transition function)**.** *A trajectory $s_0, s_1, \cdots$ is consistent with an initial state s and a transition function Φ if $s_0 = s$ and for $i \geq 0$, there is an action a_i, such that $s_{i+1} \in \Phi(s_i, a_i)$.* □

Using the above definition, for any LTL formula φ, $\varphi(s, \Phi)$ is now defined as the set of trajectories $\{\sigma : \sigma$ is conistent with s and Φ and $(s, \sigma) \models \varphi\}$. A goal g can be expressed as a formula φ in language L if $\varphi(s, \Phi) = g(s, \Phi)$ for all state s and transition function Φ. Now, a policy π satisfies an LTL goal φ if the set of trajectories consistent with π is a subset of $\varphi(s, \Phi)$.

Note that this definition is slightly different from the original definition as we define a goal as a mapping from (s, Φ) to set of trajectories instead of simplyly

a set of trajectories. This definition is adopted to be consistent with the rest of the dissertation. The importance of having this definition is discussed in the next chapter.

Often [BK98, BKT01], planning with respect to LTL goals are clubbed with the assumption that there is complete information about the initial state, and the actions are deterministic. In that case there is at most one trajectory consistent with the policy. The role of LTL in specifying planning goals has been well studied and examples of that can be found in [BK98, NS00, BKT01].

Goal Representation Using Branching Time Temporal Logic

The use of a branching time temporal logic in specifying planning goals that cannot be specified using LTLs are studied in [NS00, PT01, BKT01]. The necessity of branching time operators arise for several reasons. In particular, it is needed when a user wants to specify conditions on other paths starting from the states in the agent's main path. For example, a robot going from position *A* to position *B* may be required to take a path so that from any point in the path there is a charging station within two steps. Note that these two steps do not have to be in the path of the robot. This goal cannot be expressed using LTLs and a branching time logic such as CTL* is needed. The syntax and semantics of CTL* [ES89, Eme90] is now given below.

There are two kinds of formulas in CTL*: state formulas and path formulas. Normally state formulas are properties of states while path formulas are properties of paths. The syntax of state and path formulas is as follows:

Definition 4. *Let $\langle p \rangle$ be an atomic proposition, $\langle sf \rangle$ be a state formula, and $\langle pf \rangle$ be a path formula.*

$\langle sf \rangle ::= \langle p \rangle \mid \langle sf \rangle \wedge \langle sf \rangle \mid \langle sf \rangle \vee \langle sf \rangle \mid \neg \langle sf \rangle \mid \mathsf{E}\langle pf \rangle \mid \mathsf{A}\langle pf \rangle$

$\langle pf \rangle ::= \langle sf \rangle \mid \langle pf \rangle \vee \langle pf \rangle \mid \neg \langle pf \rangle \mid \langle pf \rangle \wedge \langle pf \rangle \mid \langle pf \rangle \,\mathsf{U}\, \langle pf \rangle \mid \bigcirc \langle pf \rangle \mid \Diamond \langle pf \rangle \mid \Box \langle pf \rangle$

□

The symbols A and E are the branching time operators meaning 'for all paths' and 'there exists a path' respectively. As the quantification 'branching time' suggests, specification in the branching time logic CTL* are evaluated with respect to the branching structure of the time. The term 'path' in the meaning of A and E refers to a trajectory in the branching structure of time.

Now define the formal semantics of CTL* formulas, which are defined depending on whether they are state formulas or path formulas.

Definition 5 (Truth of state formulas)**.** *The truth of state formulas are defined with respect to a pair* (s_j, Φ)*, where* s_j *is a state and* Φ *is the transition function. In the following* p *denotes an atomic proposition,* sf_i*s are state formulas, and* pf_i*s are path formulas.*

- $(s_j, \Phi) \models p$ *iff* p *is true in* s_j.
- $(s_j, \Phi) \models \neg sf$ *iff* $(s_j, \Phi) \not\models sf$.
- $(s_j, \Phi) \models sf_1 \wedge sf_2$ *iff* $(s_j, \Phi) \models sf_1$ *and* $(s_j, \Phi) \models sf_2$.
- $(s_j, \Phi) \models sf_1 \vee sf_2$ *iff* $(s_j, \Phi) \models sf_1$ *or* $(s_j, \Phi) \models sf_2$.
- $(s_j, \Phi) \models \mathsf{E}\ pf$ *iff there exists a trajectory* σ *in* Φ *starting from* s_j *such that* $(s_j, \Phi, \sigma) \models pf$.
- $(s_j, \Phi) \models \mathsf{A}\ pf$ *iff* $(s_j, \Phi, \sigma) \models pf$ *for all trajectories* σ *in* Φ *starting from* s_j.

□

Definition 6 (Truth of path formulas)**.** *The truth of path formulas are defined with respect to a triplet* (s_j, Φ, σ) *where* Φ *is a transition function,* σ *is a trajectory* $s_0, s_1, \ldots$ *consistent with* Φ*, and* s_j *is a state in* σ.

- $(s_j, \Phi, \sigma) \models sf$ *iff* $(s_j, \Phi) \models sf$.
- $(s_j, \Phi, \sigma) \models \neg pf$ *iff* $(s_j, \Phi, \sigma) \not\models pf$.
- $(s_j, \Phi, \sigma) \models pf_1 \vee pf_2$ *iff* $(s_j, \Phi, \sigma) \models pf_1$ *or* $(s_j, \Phi, \sigma) \models pf_2$.
- $(s_j, \Phi, \sigma) \models pf_1 \wedge pf_2$ *iff* $(s_j, \Phi, \sigma) \models pf_1$ *and* $(s_j, \Phi, \sigma) \models pf_2$.
- $(s_j, \Phi, \sigma) \models \bigcirc pf$ *iff* $(s_{j+1}, \Phi, \sigma) \models pf$.
- $(s_j, \Phi, \sigma) \models \Box pf$ *iff* $(s_k, \Phi, \sigma) \models pf$, *for all* $k \geq j$.
- $(s_j, \Phi, \sigma) \models \Diamond pf$ *iff* $(s_k, \Phi, \sigma) \models pf$, *for some* $k \geq j$.
- $(s_j, \Phi, \sigma) \models pf_1 \mathsf{U}\ pf_2$ *iff there exists* $k \geq j$ *such that* $(s_k, \Phi, \sigma) \models pf_2$ *and for all i,* $j \leq i < k$, $(s_i, \Phi, \sigma) \models pf_1$. □

Using the above definition, similar to the definition in LTL, for any CTL* formula φ, $\varphi(s, \Phi)$ is defined as the set of trajectories $\{\sigma : \sigma$ is conistent with s and Φ and $(s, \Phi, \sigma) \models \varphi\}$. A policy π satisfies a CTL* goal φ if the set of trajectories consistent with π is a subset of $\varphi(s, \Phi)$.

Now the goal of getting to B such that from anywhere in the path, a state where p holds can be reached in at most two steps, can be represented in CTL* as: $(p \vee \mathsf{E}\bigcirc p \vee \mathsf{E}\bigcirc\mathsf{E}\bigcirc p)\ \mathsf{U}\ at\_B$. Additional examples of the use of branching temporal logics CTL and CTL* to specify goals are given in [BK98, NS00, PT01, BKT01].

2.2 k-maintainability

This section recalls various definitions, algorithms and results from [BEBN08]. They are used in Chapter 6 in proposing new algorithms for strong, weak, and strong cyclic plans. In this section, the definition of k-maintainability is defined first.

Definition 7 (System)**.** *A system is a quadruple* $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$, *where*

- $\mathscr{S}$ *is the set of states;*
- $\mathscr{A}$ *is the set of actions, which is the union of two disjoint sets of actions: agents actions,* $\mathscr{A}_{ag}$*, and environmental actions,* $\mathscr{A}_{env}$*.*
- $poss : \mathscr{S} \to 2^{\mathscr{A}}$ *is a function that describes which actions are possible in which states; and*
- $\Phi : \mathscr{S} \times \mathscr{A} \to 2^{\mathscr{S}}$ *is a non-deterministic transition function that specifies how the state of the world changes in response to actions.*

Assume here that possible actions always lead to some successor states, i.e., in any system, the claim that $\Phi(s,a) \neq \emptyset$ whenever $a \in poss(s)$ holds for any state s and action a. On the other hand, given Φ, if $a \in poss(s)$ whenever $\Phi(s,a) \neq \emptyset$, then $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$ is abbreviated as $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$ by default.

Definition 8 (Control, super-control policy)**.** *Given a system* $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$ *and a set* $\mathscr{A}_{ag} \subseteq \mathscr{A}$ *of agent actions,*

- *a* super-control policy for $\mathscr{D}$ w.r.t. $\mathscr{A}_{ag}$ *is a partial function* $K : \mathscr{S} \to 2^{\mathscr{A}_{ag}}$ *such that* $K(s) \subseteq poss(s)$ *and* $K(s) \neq \emptyset$ *whenever* $K(s)$ *is defined.*
- *a* control policy for $\mathscr{D}$ w.r.t. $\mathscr{A}_{ag}$ *is a super-control such that* $K(s) = 1$ *whenever* $K(s)$ *is defined.*

Definition 9. *Given a system* $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$ *and a state* s*,* $R(\mathscr{D}, s) \subseteq \mathscr{S}$ *is the smallest set of states that satisfies the following conditions: (i)* $s \in R(\mathscr{D}, s)$*, and (ii) if* $s' \in R(\mathscr{D}, s)$*, and* $a \in poss(s')$*, then* $\Phi(s', a) \subseteq R(\mathscr{D}, s)$*.* $R(\mathscr{D}, s)$ *is the smallest set of states that are reachable from* s *by following actions in poss.* □

Definition 10 (Closure)**.** *Let* $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$ *be a system and let* $S \subseteq \mathscr{S}$ *be a set of states. Then the closure of* $\mathscr{D}$ *w.r.t.* S*, denoted by* $Closure(S, \mathscr{D})$*, is defined by* $Closure(S, \mathscr{D}) = \bigcup_{s \in S} R(\mathscr{D}, s)$*.* □

Definition 11 (*Unfold*$_k(s,\mathscr{D},K)$)**.** *Let* $\mathscr{D}=\langle\mathscr{S},\mathscr{A},\Phi,poss\rangle$ *be a system, let* $s\in\mathscr{S}$*, and let K be a control for* $\mathscr{D}$*. Then* *Unfold*$_k(s,\mathscr{D},K)$ *is the set of all sequences* $\sigma=s_0,a_1,s_1,a_2,\ldots,a_l,s_l$ *where* $l\leq k$ *and* s_0=*s such that* $a_{j+1}\in K(s_j)$ *is defined for all* $j<l$, $s_{j+1}\in\Phi(s_j,a_{j+1})$*, and if* $l<k$, $K(s_l)$ *is undefined.* □

Informally, *Unfold*$_k(s,\mathscr{D},K)$ contains all maximal paths in the system that emerge by taking agent actions, starting at *s*, such that the total length of each path is at most *k*.

Now define the notion of *k*-maintainability. In it, the function $exo:\mathscr{S}\to 2^{\mathscr{A}}$ specifies which exogenous actions can occur in which states. Let $\mathscr{D}_{K,exo}$ be the system $\langle\mathscr{S},\mathscr{A},\Phi,poss_{K,exo}\rangle$, where $poss_{K,exo}(s)=K(s)\cup exo(s)$.

Definition 12. *[k-Maintainability] Given a system* $\mathscr{D}=\langle\mathscr{S},\mathscr{A},\Phi,poss\rangle$*, a set of agent actions* $\mathscr{A}_{ag}\subseteq\mathscr{A}$*, and a specification of exogenous action occurrence exo, a control* [1] *K for A w.r.t.* $\mathscr{A}_{ag}$ *k*-maintains $\mathscr{I}\subseteq\mathscr{S}$ with respect to $\mathscr{G}\subseteq\mathscr{S}$*, where* $k\geq 0$*, if it holds for each state* $s\in Closure(\mathscr{I},\mathscr{D}_{K,exo})$ *and each sequence* $\sigma=s_0,a_1,s_1,a_2,\ldots,a_l,s_l$ *in* *Unfold*$_k(s,\mathscr{D},K)$ *with* $s_0=s$ *and* $\{s_0,\ldots,s_l\}\cap\mathscr{G}\neq\emptyset$*.*

A set of states $\mathscr{I}\subseteq\mathscr{S}$ *(resp.* $\mathscr{D}$*, if* $\mathscr{I}=\mathscr{S}$*) is k*-maintainable*,* $k\geq 0$*, w.r.t. a set of states* $\mathscr{G}\subseteq\mathscr{S}$*, if there exists a control K which k-maintains* $\mathscr{I}$ *w.r.t.* $\mathscr{G}$*. Furthermore,* $\mathscr{I}$ *(resp.* $\mathscr{D}$*) is called* maintainable *w.r.t* $\mathscr{G}$*, if* $\mathscr{I}$ *(resp.* $\mathscr{D}$*) is k*-maintainable *w.r.t.* $\mathscr{G}$ *for some* $k\geq 0$*.* □

Note that as easily verified, *k*-maintainability for $k\geq|\mathscr{S}|$ and $|\mathscr{S}|$-maintainability always coincide.

The approach above is used in [BEBN08] to develop an algorithm that finds *k*-maintainable policies. The problem is referred to as *k*-Maintain. It has the following input and output:

[1] Here only $K(s)$ for $s\in Closure(S,\mathscr{D}_{K,exo})$ is of relevance. For all other *s*, $K(s)$ can be arbitrary or undefined.

Input: An input I is a system $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$, sets of states $\mathscr{G} \subseteq \mathscr{S}$ and $\mathscr{I} \subseteq \mathscr{S}$, a set $\mathscr{A}_{ag} \subseteq \mathscr{A}$, a function *exo*, and an integer $k \geq 0$.

Output: A control K such that for $\mathscr{D}$ w.r.t. $\mathscr{A}_{ag}$ k-maintains $\mathscr{G}$ w.r.t. $\mathscr{I}$, if such a control exists. Otherwise, output the answer that no such control exists.

Before describing the encoding, the following definition of an a-path is needed.

Definition 13 (a-path)**.** *In a system $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$, there exists an* a-path *of length at most $k \geq 0$ from a state s to a set of states $\mathscr{G}$, if either*

1. *$s \in \mathscr{G}$, or*

2. *$s \notin \mathscr{G}$, $k > 0$ and there is some action $a \in \mathscr{A}_{ag} \cap poss(s)$ such that for every $s' \in \Phi(s,a)$ there exists an a-path of length at most $k-1$ from s' to $\mathscr{G}$.* □

In the following encoding of an instance I of problem k-Maintain to SAT, referred to as $sat'(I)$, s_i will intuitively denote that there is an a-path from s to $\mathscr{G}$ of length at most i. The proposition s_a_i, $i > 0$, will denote that for such s there is an a-path from s to $\mathscr{G}$ of length at most i starting with action a ($\in poss(s)$). The encoding $sat'(I)$ has groups (0) – (5) of clauses as follows:

(0) For all $s \in \mathscr{S}$, and for all j, $0 \leq j < k$:

$$s_j \Rightarrow s_{j+1}$$

(1) For all $s \in \mathscr{G} \cap \mathscr{I}$: s_0

(2) For any state $s \in \mathscr{S}$ and s' such that $s' \in \Phi(a,s)$ for some action $a \in exo(s)$:

$$s_k \Rightarrow s'_k$$

(3) For every state $s \in \mathscr{S} \setminus \mathscr{G}$ and for all i, $1 \leq i \leq k$:

(3.1) $s_i \Rightarrow \bigvee_{a \in \mathscr{A}_{ag} \cap poss(s)} s\_a_i$;

(3.2) for every $a \in \mathscr{A}_{ag} \cap poss(s)$ and $s' \in \Phi(s,a)$:

$$s\_a_i \Rightarrow s'_{i-1};$$

(3.3) for every $a \in \mathscr{A}_{ag} \cap poss(s)$, if $i < k$:

$$s\_a_i \Rightarrow s\_a_{i+1}.$$

(4) For all $s \in \mathscr{I} \setminus \mathscr{G}$: s_k

(5) For all $s \in \mathscr{S} \setminus \mathscr{G}$: $\neg s_0$

Proposition 1. *[BEBN08] Let I consist of a system $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$, a set $\mathscr{A}_{ag} \subseteq \mathscr{A}$, sets of states $\mathscr{I}, \mathscr{G} \subseteq \mathscr{S}$, an exogenous function exo, and a positive integer k. For any model M of $sat'(I)$, let $C_M = \{s \in \mathscr{S} \mid M \models s_k\}$, and for any state $s \in C_M \setminus \mathscr{G}$ let $\ell_M(s)$ denote the smallest index j such that $M \models s\_a_j$ for some action $a \in \mathscr{A}_{ag} \cap poss(s)$, which is called the* a-level *of s w.r.t. M. Then,*

(i) *$\mathscr{I}$ is k-maintainable w.r.t. $\mathscr{G}$ iff $sat'(I)$ is satisfiable;*

(ii) *given any model M of $sat'(I)$, the partial function $K_M^+ : \mathscr{S} \to 2^{\mathscr{A}_{ag}}$ which is defined on $C_M \setminus \mathscr{G}$ by*

$$K_M^+(s) = \{a \mid M \models s\_a_{\ell_M(s)}\}$$

is a valid super-control; and

(iii) *any control K which refines K_M^+ for some model M of $sat'(I)$ k-maintains $\mathscr{I}$ w.r.t. $\mathscr{G}$.*

The encoding $sat'(I)$ is a reverse Horn theory. $sat'(I)$ can be rewritten to a Horn theory, $\overline{sat}'(I)$ by reversing the propositions, where the intuitive meaning of $\overline{s_i}$ and $\overline{s\_a_i}$ is the converse of the meaning of s_i and s_a_i respectively. The encoding $\overline{sat}'(I)$ is as follows:

(0) For all $s \in \mathscr{S}$ and j, $0 \leq j < k$:

$$\overline{s_{j+1}} \Rightarrow \overline{s_j}.$$

(1) For all $s \in \mathscr{G} \cap \mathscr{I}$:

$$\overline{s_0} \Rightarrow \bot.$$

(2) For any state $s \in \mathscr{S}$ and s' such that $s' \in \Phi(a,s)$ for some action $a \in exo(s)$:

$$\overline{s'_k} \Rightarrow \overline{s_k}.$$

(3) For every state $s \in \mathscr{S} \setminus \mathscr{G}$ and for all i, $1 \leq i \leq k$:

(3.1) $\left(\bigwedge_{a \in \mathscr{A}_{ag} \cap poss(s)} \overline{s\_a_i} \right) \Rightarrow \overline{s_i}$;

(3.2) for every $a \in \mathscr{A}_{ag} \cap poss(s)$ and $s' \in \Phi(s,a)$:

$$\overline{s'_{i-1}} \Rightarrow \overline{s\_a_i};$$

(3.3) for every $a \in \mathscr{A}_{ag} \cap poss(s)$, if $i < k$:

$$\overline{s\_a_{i+1}} \Rightarrow \overline{s\_a_i}.$$

(4) For all $s \in \mathscr{I} \setminus \mathscr{G}$:

$$\overline{s_k} \Rightarrow \bot.$$

(5) For all $s \in \mathscr{S} \setminus \mathscr{G}$:

$$\overline{s_0}.$$

Theorem 1. *[BEBN08] Let I consist of a system $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$, a set $\mathscr{A}_{ag} \subseteq \mathscr{A}$, sets of states $\mathscr{I}, \mathscr{G} \subseteq \mathscr{S}$, an exogenous function exo, and a positive integer k. Let, for any model M of $\overline{sat}'(I)$, $\overline{C}_M = \{s \mid M \not\models \overline{s_k}\}$, and let $\overline{\ell}_M(s) = \min\{j \mid M \not\models \overline{s\_a_j}, a \in \mathscr{A}_{ag} \cap poss(a)\}$. Then,*

(i) $\mathscr{I}$ is k-maintainable w.r.t. $\mathscr{G}$ iff the Horn SAT instance $\overline{sat}'(I)$ is satisfiable;

(ii) Given any model M of $\overline{sat}'(I)$, every control K such that $K(s)$ is defined iff $s \in \overline{C}_M \setminus \mathscr{G}$ and satisfies

$$K(s) \in \{a \in \mathscr{A}_{ag} \cap poss(s) \mid M \not\models \overline{s\_a_j}, j = \bar{\ell}_M(s)\},$$

k-maintains $\mathscr{I}$ w.r.t. $\mathscr{G}$.

From the encoding to Horn SAT above, a direct algorithm to construct a k-maintainable control, if one exists, can be distilled. The algorithm mimics the steps which a SAT solver might take in order to solve $\overline{sat}'(I)$. It uses counters $c[s]$ and $c[s\_a]$ for each state $s \in \mathscr{S}$ and possible agent action a in state s, which range over $\{-1, 0, \ldots, k\}$ and $\{0, 1, \ldots, k\}$, respectively. Intuitively, value i of counter $c[s]$ (at a particular step in the computation) represents that so far $\overline{s_0}, \ldots, \overline{s_i}$ are assigned true; in particular, $i = -1$ represents that no $\overline{s_i}$ is assigned true yet. Similarly, value i for $c[s\_a]$ (at a particular step in the computation) represents that so far $\overline{s\_a_1}, \ldots, \overline{s\_a_i}$ are assigned true (and in particular, $i = 0$ that no $\overline{s\_a_i}$ is assigned true yet).

Starting from an initialization, the algorithm updates by demand of the clauses in $\overline{sat}'(I)$ the counters (i.e., sets propositions true) using a command $upd(c,i)$: 'if $c < i$ then $c := i$' towards a fix-point. If a counter violation is detected, corresponding to violation of a clause $\overline{s_0} \to \bot$ for $s \in \mathscr{I} \cap \mathscr{G}$ in (1) or $\overline{s_k} \to \bot$ for $s \in \mathscr{I} \setminus \mathscr{G}$ in (4), no control is possible. Otherwise, a control is constructed from the counters. The algorithm is illustrated in Algorithm 2.2.

Algorithm 2.2 is easily modifiable if users simply want to output a super-control such that each of its refinements is a k-maintainable control, leaving a choice about the refinement to the user. Alternatively, such a choice based on preference information can be implemented in Step 4.

The following proposition states that the algorithm works correctly and runs in time polynomial in k and I.

[k-Control]

Input: A system $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$, a set $\mathscr{A}_{agent} \subseteq \mathscr{A}$ of agent actions, sets of states $\mathscr{I}, \mathscr{G} \subseteq \mathscr{S}$, an exogenous function *exo*, and an integer $k \geq 0$.

Output: A control K which k-maintains $\mathscr{I}$ with respect to $\mathscr{G}$, if any such control exists. Otherwise, output that no such control exists.

(Step 1) Initialization

(i) Set $\Phi_{exo} = \{\langle s,a,s' \rangle \mid s \in \mathscr{S}, a \in exo(s), s \in \Phi(s,a)\}$, $\Phi^{\overline{\mathscr{G}}}_{poss} = \{\langle s,a,s' \rangle \mid s \in \mathscr{S} \setminus \mathscr{G},\ a \in poss(s), s' \in \Phi(s,a)\}$, and for every $s \in \mathscr{S}$, $poss_{ag}(s) = \mathscr{A}_{agent} \cap poss(s)$.

(ii) For every s in $\mathscr{G}$, set $c[s] := -1$.

(iii) For every s in $\mathscr{S} \setminus \mathscr{G}$, set $c[s] := k$ if $s \in \mathscr{I}$ and $poss_{ag}(s) = \emptyset$; otherwise, set $c[s] := 0$.

(iv) For each $s \in \mathscr{S} \setminus \mathscr{G}$ and $a \in poss_{ag}(s)$, set $c[s\_a] := 0$.

(Step 2) Repeat the following steps until there is no change or $c[s]{=}k$ for some $s \in \mathscr{I} \setminus \mathscr{G}$ or $c[s] \geq 0$ for some $s \in \mathscr{I} \cap \mathscr{G}$:

(i) For any $\langle s,a,s' \rangle \in \Phi_{exo}$ with $c[s']{=}k$ do $upd(c[s],k)$.

(ii) For any $\langle s,a,s' \rangle \in \Phi^{\overline{\mathscr{G}}}_{poss}$ such that $c[s']{=}i$ and $0 \leq i < k$ do $upd(c[s\_a], i+1)$.

(iii) For any state $s \in \mathscr{S} \setminus \mathscr{G}$ such that $poss_{ag}(s) \neq \emptyset$ and $i = \min(c[s\_a] \mid a \in poss_{ag}(s))$ do $upd(c[s], i)$.

(Step 3) If $c[s]{=}k$ for some $s \in \mathscr{I} \setminus \mathscr{G}$ or $c[s] \geq 0$ for some $s \in \mathscr{I} \cap \mathscr{G}$, then output that $\mathscr{I}$ is not k-maintainable w.r.t. $\mathscr{G}$ and halt.

(Step 4) Output any control $K : \mathscr{S} \setminus \mathscr{G} \to \mathscr{A}_{agent}$ defined on all states $s \in \mathscr{S} \setminus \mathscr{G}$ with $c[s] < k$, such that $K(s) \in \{a \in poss_{ag}(s) \mid c[s\_a] = \min_{b \in poss_{ag}(s)} c[s\_b]\}$.

Proposition 2. *[BEBN08] Algorithm k-Control solves problem k-Maintain, and can be implemented to run in time $O(k\|I\|)$ for any input I.*

Since k-maintainability for $k \geq |\mathscr{S}|$ and $|\mathscr{S}|$-maintainability coincide, problem k- Maintain can be solved using k-Control in polynomial time.

With the preparation of the background knowledge, the following chapters pro-

pose the languages for different goal specification requirements and give algorithms for some of the goals. The next chapter starts with the goal specifications in non-deterministic domains.

Chapter 3

Π-CTL* AND P-CTL*: GOAL SPECIFICATION WITH TEMPORAL LOGICS IN NON-DETERMINISTIC DOMAINS

Reasoning about actions and change is an important aspect of designing autonomous agents. Closely tied to reasoning about actions and change, and also an important aspect of designing autonomous agents, is the issue of specifying desired trajectories – that satisfy the action and change principles – of the agent which is referred to as goal specification for an agent. This is necessary in designing autonomous agents, as often an agent needs to be given a directive – a high level goal specification – regarding the behavior desired from it. Temporal logics such as LTL and CTL* have been used in goal specification in deterministic domains but are not adequate for non-deterministic domains. For example, a simple goal of achieving p in a non-deterministic domain has many nuances such as having the possibility of achieving p, making sure that p is achieved, preferring guaranteed achievement of p over possible achievement, trying one's best to achieve p, and so on. These different nuances cannot be distinguished in LTL or CTL*. CTL* is extended to π-CTL* by adding two new quantifiers, exists a path following the policy and for all paths following the policy. This distinguishes paths associated with the policy from all paths in the domain. π-CTL* is further extended to P-CTL* by adding two new quantifiers, exists a policy and for all policies. Languages are also proposed for agents with different policy structures. With these extensions, many useful goals that cannot be expressed in LTL, or CTL* can be specified. The new languages also allow specification of goals that are adaptive to domains and agent's ability. With formal definitions of goals, a framework on comparing expressiveness of goal specification languages is proposed. It helps in formally proving that some goals

cannot be expressed in some goal specification languages.

3.1 Introduction and Motivation

In his pioneering paper [McC59], John McCarthy envisioned a system with commonsense which he called "Advice Taker" and said:

> The advice taker is a proposed program for solving problems by manipulating sentences in formal languages. The main difference between it and other programs or proposed programs for manipulating formal languages (the Logic Theory Machine of Newell, Simon and Shaw and the Geometry Program of Gelernter) is that in the previous programs the formal system was the subject matter but the heuristics were all embodied in the program. In this program the procedures will be described as much as possible in the language itself and, in particular, the heuristics are all so described.
>
> *T*he main advantages we expect the advice taker to have is that its behavior will be improvable merely by making statements to it, telling it about its symbolic environment and what is wanted from it. To make these statements will require little if any knowledge of the program or the previous knowledge of the advice taker. One will be able to assume that the advice taker will have available to it a fairly wide class of immediate logical consequences of anything it is told and its previous knowledge. This property is expected to have much in common with what makes us describe certain humans as having common sense. We shall therefore say that a program has common sense if it automatically deduces for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows.

In the paper, McCarthy shows how to use logic to describe properties of a world, the conditions of executing actions in that world, the effect of actions on the world, and what is desired from the world. He then shows how to use deduction with the above kinds of logical description and comes up with a plan that achieves what is specified as desired. In a later paper, McCarthy and Hayes [MH69] give a more formal and more general presentation of the above, where they introduce the Situation Calculus. Since then a large body of research has been done on the topic of reasoning about actions.

Restating the various kinds of "premises" that McCarthy's Advice Taker in [McC59] has, a *systematic design* of a (semi)-autonomous[1] agent has three main aspects:

(i) the domain description that describes actions of the agent, the description of the world including the relationship between objects in the world, the conditions when actions can or cannot be executed, and the impact of the actions on the world;

(ii) the control execution of the agent in the system, and

(iii) the directives given to the agent regarding how it should behave or what is expected from it.

In the literature a large body of work has been done on topic (i) [FN71, Ped87, GL98a], and a significant amount of work has been done on topic (ii) [Sch87, BG00, DLPT02, BDH99]. However, despite McCarthy's use of the *want* predicate in specifying what is wanted, besides this research there has been relatively less work on topic (iii). This chapter focuses on topic (iii). But since these three aspects are

[1]Here the agents considered are not fully autonomous agents who can choose their own goals or improve their control execution. This work is not working towards building agents that can make human beings their slaves.

interrelated and they relate to the main purpose of this chapter, a brief overview of all three of them is covered in the following.

Components of Agent Design

The first component is on defining the environment of the agent.

Domain Description and Transition Systems

The main research issue in describing the domain of an agent is to develop ways that allow natural and succinct, and hence often implicit, description of the transition between states of the world due to execution of action(s). But for the purpose of this chapter, it is simpler to use an explicit notion of transition systems.

A transition system is defined [GL98a] using an *action signature* which consists of three nonempty sets: a set $\mathscr{V}$ of value names, a set $\mathscr{F}$ of fluent names, and a set $\mathscr{A}$ of action names. Each fluent name represents a particular property of the system. A *state* is an interpretation of the set of fluent names. The set of states in the system is denoted as S. Let $M(f,s) \in \mathscr{V}$ be the value of fluent $f \in \mathscr{F}$ in state $s \in S$.

Definition 14. *[GL98a][Transition System] A transition system of an action signature* $\langle \mathscr{V}, \mathscr{F}, \mathscr{A} \rangle$ *consists of*

1. *a set of states S,*
2. *a function M from* $\mathscr{F} \times S$ *into* $\mathscr{V}$*, and*
3. *a transition function* Φ *from* $S \times \mathscr{A}$ *into the powerset of S.* □

The states s' such that $s' \in \Phi(s,a)$ are possible results of executing action a in state s. Action a is considered *executable* in s if $|\Phi(s,a)| > 0$. Action a is *deterministic* in s if $|\Phi(s,a)| = 1$. Action a is *non-deterministic* in s if $|\Phi(s,a)| > 1$. A domain is *non-deterministic* if it has at least one action a and one state s such

that a is non-deterministic in s. For convenience, assume that there is an action nop such that for each state $s \in S$, we have $\Phi(s, nop) = \{s\}$.

In this dissertation, assume that the world is given by a single transition system, and the transition system is known to the agent.

Control Programs and Policies

The control of an autonomous agent in a domain specifies the ability of the agent. It can be a purely deliberative type, a purely reactive type or a hybrid type. In a purely deliberative control, the agent continually follows the cycle of observe, plan-or-replan, and act. In a purely reactive control, the deliberation for or of the agent has been done beforehand and in run time it continually follows the cycle of observe, simple table-look-up and act. In one kind of hybrid control the agent may deliberate in certain states and react in others and the deliberation may itself be of various degrees. In a second kind of hybrid control the table-look-up requires evaluating formulas over the past states and actions. Focus of this chapter is mainly on reactive control. However, sometimes other definitions of the control are also considered.

As mentioned earlier a reactive control involves making observations and then looking up a table to decide on how to act. However, the structure of the table may vary from agent to agent. This structure of the table is referred as the *policy structure* and denote it as $\mathscr{P}$. Assume that an agent has a fixed policy structure. A policy π of an agent is an instantiation of its policy structure $\mathscr{P}$. A commonly used policy structure is a mapping from states to actions and a policy following that structure is a particular mapping from a specific set of states to a specific set of actions.

When an agent starts in an initial situation and follows a particular policy, the world of the agent evolves in a particular way. This evolution is formally repre-

sented as a sequence of states, also referred to as a trajectory. One can thus define when a particular sequence of states (or a trajectory) is consistent with the execution of a particular policy.

Definition 15 (Trajectories consistent with a policy)**.** *An infinite sequence of states, or a trajectory s_0, s_1, $\cdots$ is consistent with a policy π that maps from states to actions, if $s_{i+1} \in \Phi(s_i, \pi(s_i))$ for $i \geq 0$.* □

The above definition allows users to link policies with trajectories and it is useful in connecting policies with goal specifications which are often about specifying desired trajectories.

What is a Goal?

Given a transition system and a policy structure, an agent can choose a policy conforming to its policy structure to execute. The policy chosen when executed starting in a particular initial state will lead to a particular trajectory. In a domain where actions have deterministic effects this trajectory can be predetermined. But in domains where actions may have non-deterministic effects this trajectory may not be predetermined; one can at best determine a set of trajectories of how the world may progress. Under these circumstances, how does an agent decide which policy to execute?

This will depend on what the agent wants or desires. An agent may simply want to end up in one or one among a set of particular states; or it may want to have more general restrictions on how the world evolves. Such wants and desires of an agent are referred as its goal.

In classical planning, the agent's goals were to reach a final state that satisfies certain conditions. It was soon realized that in many cases the desired goal may be such that there is no final state (such as in many maintenance goals), and even if there is a final state, the desire may also include restrictions on how a final state is

reached. One example of this would be for the robot to get to a room without hitting the wall in the process of getting there. Goals were then generalized as a set of trajectories so that the agent or the robot at least follows one of them. But things get more involved when actions have non-deterministic effects. As mentioned earlier, execution of a policy under such circumstances may lead to one among a set of trajectories. In that case a goal may be to prefer some sets of possible trajectories over others; exemplified by accepting some policies over others. In other words each goal would now correspond to a set of desired set of trajectories.

The above notions of goal are all subjective or absolute: No matter what options (in terms of what actions it has at its disposal and how those actions may change the world and what the initial state may be) the agent may have, the goal is about the trajectories. However, often a goal of an agent may include aspects corresponding to choosing the "best options" among the ones that are available to the agent. To express the options that are available to the agent, the transition graph and the initial state need to be taken into account. I.e., if the transition graph, or the initial state of the agent is different, then the policies that are available for the agent to choose from could be different. Thus a goal is no longer absolute but a mapping from a set of ways about how the world may evolve to the set of ways how the evolution is desired. How the world may evolve can be expressed by an initial state s and a transition function Φ and how the evolution is desired can be expressed as (a) a set of trajectories or in some cases as (b) a set of set of trajectories.

Since in most cases a user cannot explicitly express a goal by expressing the above mentioned mapping, a succinct way of expressing goals is needed. Thus the need for a goal specification language which is the raison d'etre of this chapter.

In the literature various logics, including temporal logics, have been proposed as goal specification languages. For example, Temporal logics such as linear temporal logic LTL [Pnu77], branching time temporal logic CTL* [EC82, ES89, Eme90],

and their extensions [BK98, NS00, BKT01] have been proposed and used as goal specification languages in the autonomous agent community and planning community [BKSD95, BK98, GV99, NS00, SSD00, PT01]. In the decision theoretic planning community there are suggestions to use temporal logics in specifying non-Markovian rewards [BBG96, BBG97, TGS$^+$06].

Extension of LTL and CTL* are also studied. One direction is to extend the logic to have metric intervals [BK98] or qualitative measure on elapsed time between the occurrences of the events [Pnu77]. Following the latter, timed CTL (TCTL) [AH93], real time CTL (RTCTL) [EMSS92], and more generally qualitative logics [BEH95a, BEH95b] focus on the expressions of qualitative bounds on the occurrences of events (c.f. [ET99]).

Also in a timed transition system, Min-max CTL [DCDS01] was proposed by allowing the quantification of CTL state and path properties in terms of a cost function over real time. It uses "min" and "max" calculation in aggregating the properties of states and paths.

Another extension of LTL and CTL* is to the game-like multi-agent systems and the languages ATL and ATL* [AHK02] were invented that quantify over paths belonging to the execution of each agent. CATL [vdHJW05] further extends ATL with a ternary counterfactual commitment operator of the form $C_i(\sigma, \phi)$, with the intended reading "if it were the case that agent i committed to strategy σ, then ϕ". By using this operator in combination with the ability operators of ATL, it is possible to reason about the implications of different possible choices of agents.

In considering using temporal logics for goal specification, most of these papers – except [PT01], only consider the case when actions are deterministic. Following that direction, in [DLPT02], a question was raised regarding whether the existing temporal logics are adequate to specify many intuitive goals, especially in a non-deterministic domain, and an alternative language was proposed. In this chapter, it

is formally proved that in the case that actions have non-deterministic effects, there are goals (as motivated above) which cannot be expressed in existing temporal logics such as LTL and CTL*. However, departing from [PT01, DLPT02], extensions to these temporal logics are proposed. The proposed extensions are able to express the richer goals and nuances that one encounters in the non-deterministic domains. *To do that the formal notion that a goal is a mapping from an initial state s and a transition function* Φ *to a set of trajectories (or a set of set of trajectories) is needed. This is argued in previous paragraphs.*

To relate this notion of a goal as a mapping and the notion that a goal in a goal specification language is a formula in that language, the following notations and definitions are needed.

Consider a goal g. Let the set of trajectories that g maps an initial state s and a transition function Φ be $g(s,\Phi)$. Now consider a goal specification language $\mathscr{L}$ and a formula φ in $\mathscr{L}$. To match the notion of a goal, the semantics of $\mathscr{L}$ needs also to be defined with respect to an initial state and a transition function. In other words, given an s and a Φ, the semantics of $\mathscr{L}$ will map formulas in $\mathscr{L}$ to a set of trajectories. Intuitively, this set of trajectories "satisfy" the formula φ given s and Φ. It is denoted as $\varphi(s,\Phi)$. The notion of "satisfaction" and a corresponding entailment relation ($\models$) will be precisely defined by the semantics of the language $\mathscr{L}$. For example, in CTL* the entailment relation $\models$ is defined between triplets (s,Φ,σ) (where σ is a trajectory) and formulas of CTL*. Using that for a CTL* formula φ, the expression $\varphi(s,\Phi)$ denotes the set $\{\sigma : (s,\Phi,\sigma) \models \varphi\}$.

Now a goal g can be expressed as a formula φ in language $\mathscr{L}$ if

$$\varphi(s,\Phi) = g(s,\Phi) \tag{3.1}$$

for all state s and transition function Φ.

In the above definitions the notion of a goal as a mapping from an initial state s,

a transition function Φ to a set of trajectories is used. They can be easily extended to the case when a goal is a mapping from an initial state s, a transition function Φ to a set of set of trajectories.

A Motivating Example

The previous section alluded to the added complexity of what goals mean in domains with non-deterministic actions. The following example illustrates the difficulty of expressing goals in such domains. Later sections will formally show the inadequacies of existing goal specification languages in expressing some of the goals mentioned in this subsection. New languages will be proposed to address the inadequacies.

In a non-deterministic domain, sometimes there does not exist a definite strategy under which one can guarantee the achievement of a particular property of the world, say p. In that case the agent may be directed to "try its best" to reach a state where p is true. This idea of "trying ones best" has many nuances and they cannot be expressed in existing temporal logics.

Example 2. *Consider a domain which has five states: s_1, s_2, s_3, s_4, and s_5. The proposition p is only true in state s_4. The other states are distinguishable based on other fluents which are not elaborated here. Suppose the only possible actions and their consequences are given in Figure 3.1, except that in each state there is always an action nop that keeps the agent in the same state.*[2] □

Consider that the agent would like to try its best[3] to get to a state where p is true. The agent and its controller are aware that some of the available actions have non-deterministic effects. Thus they are looking for mappings from states to actions

[2] Although in the examples, to save space, state space diagrams are used. These diagrams can easily be grounded on action descriptions. For an example see [DLPT02].

[3] Note that special cases of 'try your best' are the well-studied (in AI) notions of strong planing, strong cyclic planning, and weak planning [CPRT03], and *T*ryReach p of [DLPT02].

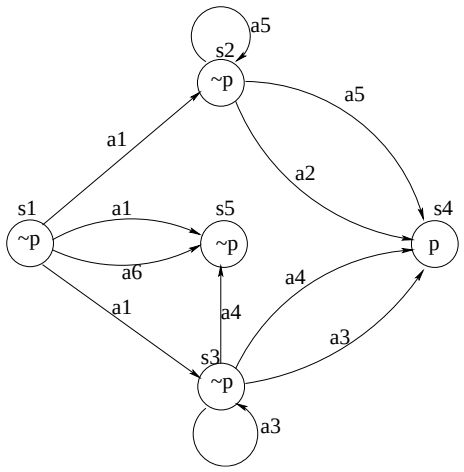

Figure 3.1: Transition diagram in a non-deterministic domain

instead of plans consisting of action sequences. Moreover, due to non-deterministic effects of actions, they are worried about how to specify their goal so that the goal is not so strict that it is not achievable and still conveys the meaning of 'trying its best'. The notion of 'trying one's best' would then have a different meaning depending on the ability of the agent, and also depending on where the agent is: In state s_1, one would prefer a_1 to a_6 because if a_6 is executed in s_1, the agent can never reach p. Similarly, in s_2, doing a_2 is better than doing a_5, since executing a_2 guarantees that p will be reached while by executing a_5 one may not reach p in the worst case. In s_3, doing a_3 is better than doing a_4 because by executing a_3 in s_3, the agent always has a hope of reaching p, but executing a_4 may lead to s_5, from which the agent can never reach p. So one interpretation of 'trying one's best' is to only accept the policy that do a_1 in s_1, a_2 in s_2 and a_3 in s_3. But one may also have a weaker goal and consider some of the other policies acceptable. To analyze this further, the following policies that are mappings from states to actions are considered. Each policy is represent by a set of pairs of states and actions of the form (s,a) which will mean that in state s, action a should be executed. It is assumed that if no action is explicitly specified for a state s then (s,nop) is implicitly present.

1. Policy $\pi_1 = \{(s_1, a_1), (s_2, a_2), (s_3, a_3)\}$
2. Policy $\pi_2 = \{(s_1, a_1), (s_2, a_2), (s_3, a_4)\}$
3. Policy $\pi_3 = \{(s_1, a_1), (s_2, a_5), (s_3, a_3)\}$
4. Policy $\pi_4 = \{(s_1, a_1), (s_2, a_5), (s_3, a_4)\}$
5. Policy $\pi_5 = \{(s_1, a_6)\}$

Consider s_1 as the initial state. Based on the preference relations of actions in each state, Figure 3.2 shows the relation between the five policies in terms of which one is preferable to the other with respect to the goal of trying one's best to get to a state where p is true. A directed edge from π_i to π_j means π_i is preferable to π_j and this preference relation is transitive. Note that given a different transition system, a different initial state, or a different goal, users might have other preferences among policies, or may even not have a preference relation.

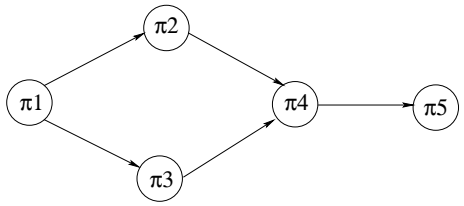

Figure 3.2: The preference relation between policies

First, try to use existing temporal logic formalisms to specify the goal of 'trying one's best to reach p.' Since the use of policies lead to multiple trajectories, a user cannot directly use the specification $\Diamond p$ from linear temporal logic with future operators (LTL) [MP92, Eme90]. Thus try to express this goal in the branching time temporal logic CTL*, where there are operators A (meaning 'for all paths') and E (meaning 'there exists a path').

Suppose the initial state of the agent is s_1. From s_1 there is a path to s_4. Thus the CTL* goal E$\Diamond p$ will be true with respect to s_1 and the transition function regardless

of which policy one chooses including π_5. Clearly, π_5 is not a policy that is trying to get to p. Thus the specification $\mathsf{E}\Diamond p$ is incorrect. Alternatively, consider the goal $\mathsf{A}\Diamond p$. This goal is too strong as, even if a user considers the initial state as s_2 from which there is a policy that guarantees p is reached, the goal $\mathsf{A}\Diamond p$ will not be true with respect to s_2 and the transition system. This is because the semantics of E and A are tied to the overall transition function. With these operators, a user cannot distinguish the set of transition relations tied to a given policy from the overall transition function in the domain. One way to overcome this is to either tie the semantics of E and A to the policy under consideration [CPRT03] or introduce new operators (say, E_{pol} and A_{pol}) that tie the paths to the policy under consideration. The second approach is chosen in this chapter, as to express certain goals it becomes necessary to have both versions (E, A, E_{pol} and A_{pol}) of the branching time operators. For example, to specify the intuition of having a policy that guarantees to reach a safe state that will never reach p from then on no matter what happens, both versions are needed in formula $\mathsf{A}_{pol}\Diamond\Box(\neg\mathsf{E}\Diamond p)$. The intuitive meaning of the operator A_{pol} is 'for all paths that are consistent with the policy under consideration' and the operator E_{pol} is 'there exists a path that is consistent with the policy under consideration.' When each policy is a mapping from states to actions, this new language is called π-CTL*. As will be described in the following sections, in π-CTL*, if a goal is represented as $\mathsf{E}_{pol}\Diamond p$, policy π_1, π_2, π_3, and π_4 satisfy the goal while policy π_5 does not satisfy the goal.

However, in π-CTL*, a goal that only accepts π_1 but rejects other policies cannot be represented. One step further, a user may want a goal specified for each subset of $\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}$. How can they be specified? Now select one such subset to explain what is needed in representing a goal that accepts π_1 and π_2 while rejects other policies. Policies π_1 and π_2 have the same action a_1 in s_1, implying that if there is a policy that can guarantee that p will be reached, the policy cho-

sen by the agent should guarantee to reach p. In general, mechanisms are needed to compare policies to indicate that "*if there is a policy* in the domain to satisfy f, the agent should take a policy to satisfy f". Language π-CTL* is extended to P-CTL* by having two new operators $\mathscr{A}\mathscr{P}$ and $\mathscr{E}\mathscr{P}$, which mean for all policies and exist a policy from a state. Latter sections in this chapter will show how various nuances of this example can be encoded in the extended language having these two new operators. For example, this goal can be specified in P-CTL* as $\mathsf{A}_{pol}\Box((\mathscr{E}\mathscr{P}\mathsf{E}_{pol}\Diamond p \Rightarrow \mathsf{E}_{pol}\Diamond p) \wedge (\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p \Rightarrow \mathsf{A}_{pol}\Diamond p))$. Intuitively, it says that in any state following the given policy, if there is a policy that makes p reachable then the policy chosen by the agent should make p reachable. Besides, if there is a policy that can always reach p no matter the non-deterministic actions, then in the policy chosen by the agent, p must be reached. Given this domain, there might be other specification in π-CTL* to distinguish π_1 and π_2 from other policies, but the formula given above is more intuitive with the quantification over policies.

One intuition to be captured with the quantification over policies is that the *expectation we have for the agent may change in the process of executing*. For example, in terms of the goal of trying the best in reaching p, initially, the agent may not be able to guarantee that p will be reached. However, in the process of executing, it may get lucky enough to reach a state from where reaching p can be guaranteed. When specifying the kind of policy that a user wants, the user may require that the agent should guarantee reaching of p from then on. It seems in [PT01, DLPT02], the authors also tried to capture the intuition of modifying plans during the execution, but their method is insufficient in doing so [BZ04].

Going further, certain goals necessitates more general notions of policies; in particular, policies that map from state sequences to actions rather than just states to actions are needed. To compare the various goal specification languages proposed in this chapter, formal methodologies are developed for comparing languages and

for defining expressiveness of languages. They are used to prove that there is a goal that can be expressed in π-CTL* but cannot be expressed in CTL*, and there is a goal that can be expressed in P-CTL* but cannot be expressed in π-CTL*. On the other hand, all goals expressed in π-CTL* can be expressed in P-CTL*. Expressiveness of a goal specification language in this context depends on the definition of the policy structure in the language. A few variations of π-CTL* and P-CTL* are defined by considering different definitions of the policy structure. These languages are also formally compared.

Contribution

In summary, the main contributions in this chapter are:

- Formally answering the question of "what is a goal";
- Extending temporal logics for goal specification in non-deterministic domains by having different branching operators in π-CTL*;
- Further extending goal specification language π-CTL* by quantifying over policies in P-CTL*;
- Pointing out that goal specification may depend on the definition of policies, and then proposing variations of π-CTL* and P-CTL* that depend on different notions of the policy;
- Proposing mechanisms and using them in formally comparing expressiveness of goal specification languages;
- Motivating on goal specification languages that are adaptive to domains.

Structure of the Chapter

The rest of this chapter is organized as follows. Section 3.2 illustrates limitations of existing logics in specifying goals in non-deterministic domains and introduces

π-CTL*. Section 3.3 shows some limitations of π-CTL*, and proposes a further extension P-CTL*. Section 3.4 demonstrates the importance of the policy structure in a language. Variations of π-CTL* and P-CTL* are then introduced. Section 3.5 formally compares expressiveness of goal specification languages. Some general issues in goal specifications such as complexity and related work are discussed in Section 3.6. This chapter is end with summary and some future work in Section 3.7.

3.2 Limitations of CTL*: Extending CTL* to π-CTL*

This section starts by showing that in a non-deterministic domain, there are goals that cannot be expressed in CTL*. CTL* is then extended to π-CTL*.

Limitations of CTL* in Non-deterministic Domains

First consider the following lemma.

Lemma 1. *Consider the transition relation Φ_1 and Φ_2 in Figure 3.3.*

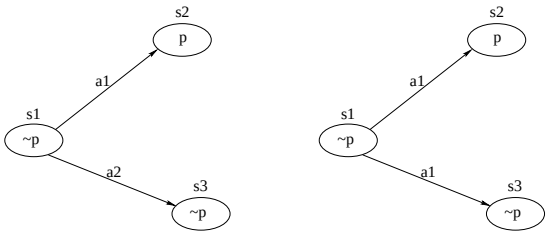

Figure 3.3: Transitions that show limitations of CTL*

1. *For any state formula φ in CTL*, $(s_1, \Phi_1) \models \varphi$ iff $(s_1, \Phi_2) \models \varphi$;*
2. *Let σ be any trajectory in Φ_1 (or Φ_2). For any path formula ψ in CTL*, $(s_1, \Phi_1, \sigma) \models \psi$ iff $(s_1, \Phi_2, \sigma) \models \psi$.*

Proof. The proof is based on the induction on depth of formulas. The notion of "depth" of formulas is defined in Appendix A.

Base Case: It is easy to see that (1) and (2) are true for CTL* formulas of depth 1.

Induction: Assume that (1) and (2) are true for CTL* formulas of depth $n < k$. That is, if $depth(\varphi) < k$ and $depth(\psi) < k$, then $(s_1, \Phi_1) \models G$ iff $(s_1, \Phi_2) \models G$ and $(s_1, \Phi_1, \sigma) \models G$ iff $(s_1, \Phi_2, \sigma) \models G$.

Let $n = k$.

Consider state formulas of depth k. It can only be the following forms: (a) $sf_1 \wedge sf_2$ (b) $sf_1 \vee sf_2$ (c) $\neg sf_1$ (d) $\mathsf{E}pf$ (e) $\mathsf{A}pf$, where sf_1, sf_2 and pf have depth less than k.

Consider (a) $sf_1 \wedge sf_2$. Since depth of sf_1 and sf_2 are less than k by induction hypothesis, $(s_1, \Phi_1) \models sf_1$ iff $(s_1, \Phi_2) \models sf_1$ and $(s_1, \Phi_1) \models sf_2$ iff $(s_1, \Phi_2) \models sf_2$. By definition, $(s_1, \Phi_1) \models sf_1 \wedge sf_2$ iff $(s_1, \Phi_1) \models sf_1$ and $(s_1, \Phi_1) \models sf_2$. Similarly, $(s_1, \Phi_2) \models sf_1 \wedge sf_2$ iff $(s_1, \Phi_2) \models sf_1$ and $(s_1, \Phi_2) \models sf_2$. Thus $(s_1, \Phi_1) \models sf_1 \wedge sf_2$ iff $(s_1, \Phi_2) \models sf_1 \wedge sf_2$.

The proofs for formulas of the forms (b) and (c) are similar.

Consider (d) $\mathsf{E}\ pf$. By definition, $(s_1, \Phi_1) \models \mathsf{E}\ pf$ iff there exists a trajectory σ in Φ_1 starting from s_1 such that $(s_1, \Phi_1, \sigma) \models pf$. it is observed that σ is a trajectory in Φ_1 iff σ is a trajectory in Φ_2. Since $depth(pf) < k$, by induction hypothesis, $(s_1, \Phi_1, \sigma) \models pf$ iff $(s_1, \Phi_2, \sigma) \models pf$. Hence, $(s_1, \Phi_1) \models \mathsf{E}pf$ iff $(s_1, \Phi_2) \models \mathsf{E}pf$.

The proof for formulas of the form (e) is similar.

Consider path formulas of depth k. It can be of the following forms: (a) $pf_1 \wedge pf_2$ (b) $pf_1 \vee pf_2$ (c) $\neg pf_1$ (d) $pf_1\ \mathsf{U}\ pf_2$ (e) $\bigcirc pf_1$ (f) $\Diamond pf_1$ (g) $\Box pf_1$, (h) sf, where pf_1 and pf_2 have depth less than k. The proof of each of these cases is similar to the proof for the corresponding state formula. □

Proposition 3. *There is a goal defined as a mapping from a state and a transition function to a set of trajectories that cannot be expressed in CTL*.*

Proof. Consider the goal g that maps (s_1, Φ_1) to the set of trajectories $s_1 s_2 s_2^*$, where $s_1 s_2 s_2^*$ is the set of trajectories with the first state being s_1 and all remaining states being s_2. g maps (s_1, Φ_2) to the set of trajectories $s_1 s_2 s_2^* \cup s_1 s_3 s_3^*$. That is $g(s_1, \Phi_1) = s_1 s_2 s_2^*$, and $g(s_1, \Phi_2) = s_1 s_2 s_2^* \cup s_1 s_2 s_2^*$.

Now show by contradiction that this goal cannot be expressed in CTL*.

Suppose goal g can be expressed in CTL* as φ. In that case $\varphi(s_1, \Phi_1) = s_1 s_2 s_2^*$ and $\varphi(s_1, \Phi_2) = s_1 s_2 s_2^* \cup s_1 s_3 s_3^*$. Consider the trajectory $s_1 s_3$ which is in the second set but not in the first. Lets refer to it as σ'. Based on the Definition of $\varphi(s, \Phi)$, $(s_1, \Phi_1, \sigma') \not\models \varphi$ while $(s_1, \Phi_2, \sigma') \models \varphi$ in CTL*.

But, according to Lemma 1, $(s, \Phi_1, \sigma') \models \varphi$ iff $(s, \Phi_2, \sigma') \models \varphi$.

There is a contradiction and hence, goal g cannot be expressed in CTL*. □

Note that the proof is based on a non-deterministic domain. In a deterministic domain the execution of a policy in an initial state leads to a unique trajectory and one can simply use LTL to specify properties of that trajectory and use the branching time operators to refer to arbitrary trajectories starting from states in the main trajectory. In case of non-deterministic domains, there are multiple possible trajectories for a policy starting from an initial state. This set of trajectories is a subset of all trajectories from the initial state. Thus one needs to distinguish trajectories that are *consistent with respect to the policy of the agent* and arbitrary trajectories. To express the former the operators E_{pol} (and A_{pol}) are introduced which means that there exists a path (and for all paths, respectively) consistent with the policy of the agent.

The syntax and semantics of this extended branching time logic π-CTL* is now formally defined, in which a policy is mapped from states to actions.

Syntax of π-CTL*

The syntax of state and path formulas in π-CTL* is as follows:

Definition 16. *Let $\langle p \rangle$ be an atomic proposition, $\langle sf \rangle$ be a state formula, and $\langle pf \rangle$ be a path formula.*

$\langle sf \rangle ::= \langle p \rangle \mid \langle sf \rangle \wedge \langle sf \rangle \mid \langle sf \rangle \vee \langle sf \rangle \mid \neg \langle sf \rangle \mid \mathsf{E} \langle pf \rangle \mid \mathsf{A} \langle pf \rangle \mid \mathsf{E}_{pol} \langle pf \rangle \mid \mathsf{A}_{pol} \langle pf \rangle$

$\langle pf \rangle ::= \langle sf \rangle \mid \langle pf \rangle \vee \langle pf \rangle \mid \neg \langle pf \rangle \mid \langle pf \rangle \wedge \langle pf \rangle \mid \langle pf \rangle \, \mathsf{U} \, \langle pf \rangle \mid \bigcirc \langle pf \rangle \mid \Diamond \langle pf \rangle \mid \Box \langle pf \rangle$

□

Semantics of π-CTL*

The semantics of π-CTL* is similar to the semantics of CTL*.

Definition 17 (Truth of state formulas in π-CTL*)**.** *The truth of state formulas is defined with respect to a triple (s_j, Φ, π) where s_j is a state, Φ is the transition function, and π is a policy that maps from states to actions.*

- $(s_j, \Phi, \pi) \models p$ *iff* p *is true in* s_j.
- $(s_j, \Phi, \pi) \models \neg sf$ *iff* $(s_j, \Phi, \pi) \not\models sf$.
- $(s_j, \Phi, \pi) \models sf_1 \wedge sf_2$ *iff* $(s_j, \Phi, \pi) \models sf_1$ *and* $(s_j, \Phi, \pi) \models sf_2$.
- $(s_j, \Phi, \pi) \models sf_1 \vee sf_2$ *iff* $(s_j, \Phi, \pi) \models sf_1$ *or* $(s_j, \Phi, \pi) \models sf_2$.
- $(s_j, \Phi, \pi) \models \mathsf{E}\ pf$ *iff there exists a path* σ *in* Φ *starting from* s_j *such that* $(s_j, \Phi, \pi, \sigma) \models pf$.
- $(s_j, \Phi, \pi) \models \mathsf{A}\ pf$ *iff* $(s_j, \Phi, \pi, \sigma) \models pf$ *for all paths* σ *in* Φ *starting from* s_j.
- $(s_j, \Phi, \pi) \models \mathsf{E}_{pol}\ pf$ *iff there exists a path* σ *in* Φ *starting from* s_j *consistent with the policy* π *such that* $(s_j, \Phi, \pi, \sigma) \models pf$.
- $(s_j, \Phi, \pi) \models \mathsf{A}_{pol}\ pf$ *iff* $(s_j, \Phi, \pi, \sigma) \models pf$ *for all paths* σ *in* Φ *starting from* s_j*, and consistent with the policy* π. □

Definition 18 (Truth of path formulas in π-CTL*)**.** *The truth of path formulas is now defined with respect to the quadruple* (s_j, Φ, π, σ)*, where* s_j *is a state,* Φ *is the transition function,* π *is a policy, and* σ *is a trajectory* $s_j, s_{j+1}, \ldots$.

- $(s_j, \Phi, \pi, \sigma) \models sf$ *iff* $(s_j, \Phi, \pi) \models sf$.
- $(s_j, \Phi, \pi, \sigma) \models \neg pf$ *iff* $(s_j, \Phi, \pi, \sigma) \not\models pf$
- $(s_j, \Phi, \pi, \sigma) \models pf_1 \wedge pf_2$ *iff* $(s_j, \Phi, \pi, \sigma) \models pf_1$ *and* $(s_j, \Phi, \pi, \sigma) \models pf_2$.
- $(s_j, \Phi, \pi, \sigma) \models pf_1 \vee pf_2$ *iff* $(s_j, \Phi, \pi, \sigma) \models pf_1$ *or* $(s_j, \Phi, \pi, \sigma) \models pf_2$.
- $(s_j, \Phi, \pi, \sigma) \models \bigcirc pf$ *iff* $(s_{j+1}, \Phi, \pi, \sigma) \models pf$.
- $(s_j, \Phi, \pi, \sigma) \models \Box pf$ *iff* $(s_k, \Phi, \pi, \sigma) \models pf$*, for all* $k \geq j$.
- $(s_j, \Phi, \pi, \sigma) \models \Diamond pf$ *iff* $(s_k, \Phi, \pi, \sigma) \models pf$*, for some* $k \geq j$.
- $(s_j, \Phi, \pi, \sigma) \models pf_1 \cup pf_2$ *iff there exists* $k \geq j$ *such that* $(s_k, \Phi, \pi, \sigma) \models pf_2$*, and for all* i*,* $j \leq i < k$*,* $(s_i, \Phi, \pi, \sigma) \models pf_1$. □

Using these definitions, for a π-CTL* formula φ, a policy π satisfies a goal φ from s_0, if $(s_0, \Phi, \pi) \models \varphi$ in π-CTL*. The set $\varphi(s, \Phi)$ denotes the set of set of trajectories

$\{\pi_\sigma : (s, \Phi, \pi) \models \varphi$ and π_σ is the set of trajectories that are consistent with policy $\pi\}$.

Goal Representation in π-CTL*

Various kinds of goals which cannot be appropriately expressed in LTL or CTL* can be expressed in π-CTL*. It is shown in the following.

π-CTL* differs from CTL*

A few goals that cannot be represented in CTL* are illustrated now. Given a policy, which is a mapping from states to actions, a user is able to check whether the policy

satisfies the goal in π-CTL*. According to the definition, a policy π satisfies a goal φ from s_0 if $(s_0, \Phi, \pi) \models \varphi$. A user needs to check properties of paths in the policy against the whole transition system. This is different from CTL*, in which there is no explicit distinction of the paths in the transition system and the paths follow the policy.

1. From the initial state, if there is a path that is possible to reach p, the agent's policy should also allow that possibility. This goal can be represented in π-CTL* as $\mathsf{E}\Diamond p \rightarrow \mathsf{E}_{pol}\Diamond p$. In a domain, given a policy, to check whether the policy satisfies a goal or not, a user also needs to refer to paths that are not consistent with the policy. This goal is one such example. In a domain, if no path can reach p by following any policy of the agent, then any policy taken by the agent would satisfy this goal. On the other hand, if there is a policy in the domain that has a chance of reaching p, the agent must take a policy that has a chance of reaching p.

2. Navigate among states that have chances of reaching p, but do not have to reach p. This goal specifies that from the initial state, each state in the policy has a path of reaching p, where those paths may not be in the policy. This can be represented in π-CTL* as: $\mathsf{A}_{pol}\Box(\mathsf{E}\Diamond p)$.

Reachability Goals Corresponding to Example 2

How various kinds of reachability goals can be specified in π-CTL* is illustrated here. The domain in Example 2 is considered.

1. $G_w^\pi = \mathsf{E}_{pol}\Diamond p$: This goal specifies that from the initial state, a state where p is true may be reached by following the given policy. This is referred to as weak planning in the literature. In Example 2, with respect to the initial state

s_1, the policies π_1, π_2, π_3, and π_4 satisfy this goal, while the policy π_5 does not.

2. $G_s^\pi = \mathsf{A}_{pol}\Diamond p$: This goal specifies that from the initial state, a state where p is true will be reached by following the given policy. This is referred to as strong planning. In Example 2, with respect to the initial state s_1, no policy satisfies this goal. But if the initial state is s_2, the policy $\{(s_2, a_2)\}$ satisfies this goal.

3. $G_{sc}^\pi = \mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)$: This goal specifies that all along the trajectory – following the given policy – there is always a possible path to a state where p is true. This is referred to as strong cyclic planning [CPRT03]. With respect to the initial state s_1, this goal is not satisfied as no policy can make this true. But if the initial state is s_2, policies $\{(s_2, a_5)\}$ and $\{(s_2, a_2)\}$ satisfy this goal.

4. $G_{aw}^\pi = \mathsf{A}_{pol}\Box(\mathsf{E}\Diamond p \rightarrow \mathsf{E}_{pol}\Diamond p)$: This goal specifies that in any state s that is reachable from the initial state by following the policy, if it is possible to reach p from s, then the agent's policy should allow that possibility. In Example 2, policies π_1, π_2, π_3, and π_4 satisfy this goal while the policy π_5 does not.

Given the initial state and a policy, if a property p can be reached in all paths consistent with the policy π or cannot be reached in any path consistent with the policy π, then by following the policy, the reachability of property p is not changed. Otherwise, due to non-deterministic effect of actions, as an agent proceeds with the execution of its policy, its situation regarding a goal may keep changing. For example, initially, a formula can be reached by the policy but the formula is not guaranteed by the policy. During the execution of actions in the policy, when the agent gets to some states, it may realize that the formula it intended to reach can no

longer be reached. When the agent gets to other states, it may also realize that one can guarantee that the formula can be reached even taking the non-deterministic property of the domain into account.

Maintainability Goals

Reachability goals are mainly considered previously. Maintainability can be considered as the opposite of reachability. For example, in the deterministic domain, given a plan, the path formula ϕ maintained iff $\neg\phi$ cannot be reached. It is also the case in the non-deterministic domain. In language π-CTL*, $\mathsf{E}_{pol}\varphi$ and $\neg\mathsf{A}_{pol}\neg\varphi$ are equivalent for any path formula φ. As a consequence, in formulating the goals about maintainability, a user can indeed translate them into the goals of checking whether a state can be reached or not, thus the various notions of reachability from the previous section have corresponding notions of maintainability.

For example, according to the relationship between reachability and maintainability, here are a few observations in the following.

If a propositional formula can be maintained in all trajectories consistent with the policy or cannot be reached in any trajectories consistent with the policy, then the maintainability of this propositional formula will not change during the execution of the policy. In other cases, for example, in the initial state, by following the policy, a formula can be maintained in some trajectories but not in all trajectories. During the execution of the policy, the agent may find out that the path formula can be maintained in all trajectories starting from the state it is in. It is also possible that the agent may find out that the path formula cannot be maintained in any of the trajectories starting from the state it is in.

Goals Composed of Multiple Sub-goals

Now consider goal specifications that are composed of two sub-goals. The composition is based on asking the following questions: Does the agent has to reach

the first goal? When does the agent give up the first goal? When the first goal is reached, does the agent still need to reach the second goal? When the first goal cannot be reached, does the agent need to reach the second goal? In the process of reaching the second goal, does the need to keep an eye on the first goal?

1. The policy must reach p, and must reach q after reaching p. The agent starts to consider q only after reaching p. The agent does not care whether q is reached or not in the process of reaching p. The π-CTL* representation of this goal is $\mathsf{A}_{pol}\Diamond(p \wedge \mathsf{A}_{pol}\Diamond q)$.

2. In a state if it is possible to reach p, try to reach p until it is impossible to do so. From the state that p can never be reached, try to reach q until it is impossible to do so. The π-CTL* representation of this goal is $\mathsf{A}_{pol}\Box((\mathsf{E}\Diamond p \rightarrow \mathsf{E}_{pol}\Diamond p) \wedge ((\neg\mathsf{E}\Diamond p \wedge \mathsf{E}\Diamond q) \rightarrow \mathsf{E}_{pol}\Diamond q))$.

3. If there is a trajectory that makes it possible to reach p, try to reach it. If you are in a state that p can never be reached, you must reach q from that state. The π-CTL* representation of the goal is $\mathsf{A}_{pol}\Box((\mathsf{E}\Diamond p \rightarrow \mathsf{E}_{pol}\Diamond p) \wedge (\neg(\mathsf{E}\Diamond p) \rightarrow \mathsf{A}_{pol}\Diamond q))$.

4. Make sure that goal $p \vee q$ is reached finally. Besides, in any state, if it is possible to reach p and the action cannot lead the agent to a state where neither p nor q can be reached, the agent tries to reach p. The π-CTL* representation of this goal is $(\mathsf{A}_{pol}\Diamond(p \vee q)) \wedge \mathsf{A}_{pol}\Box((\mathsf{E}\Diamond p \wedge \mathsf{A}_{pol} \bigcirc \mathsf{A}\Diamond(p \vee q)) \rightarrow \mathsf{E}_{pol}\Diamond p)$.

In these examples, in some cases, the agent not only wants to know whether there is a path from a state that can reach p or not, but also wants to know whether there is a policy from a state such that all paths consistent with the policy from that state can reach p or not. To better satisfy this, operators that quantifying over policies are needed.

3.3 P-CTL*: The Need for Higher Level Quantifiers

In Example 2, when considered from the starting state s_1 partitions the set of policies $\{\pi_1, \ldots, \pi_5\}$, π-CTL* can be used to express a few goals but no specifications only accepts π_1 is given. Sometimes, there is a need of comparing properties of a policy with properties of other policies in the domain. In particular, if accepting only π_1 means that only the best policy is accepted, this goal cannot be represented in π-CTL*. The following shows that this goal and other partitions of $\{\pi_1, \ldots, \pi_5\}$ can be expressed when there is an enhanced language that allows quantification over policies.

Quantifying Over Policies

An example to illustrate the need of quantifying over policies is given firstly.

Example 3. *Consider the two transition diagrams Φ_1 and Φ_2 of Figure 3.4, which may correspond to two distinct domains. The two diagrams have states s_1 and s_2, and actions a_1 and a_2. In state s_1 the fluent p is false, while p is true in state s_2. In both transition diagrams a_2 is a non-deterministic action which when executed in state s_1 may result in the transition to state s_2 or may stay in s_1, and when executed in s_2 stays in s_2. The action a_1 is only present in the transition diagram Φ_1 and if it is executed in state s_1 then it causes the transition to s_2.*

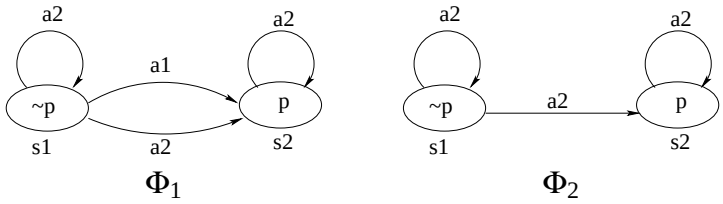

Figure 3.4: Transitions that show limitations of π-CTL*

Now suppose the agent, which is in state s_1 (where p is false), wants to try its best to get to s_2 where p is true. Aware of the fact that actions could be non-

deterministic and there may not always exist policies that can guarantee that the agent reaches p, the agent and its handlers are willing to settle for less, such as a strong cyclic policy, when no better options are available. Thus for the domain corresponding to transition diagram Φ_2, the policy $\pi = \{(s_1, a_2), (s_2, a_2)\}$ is an acceptable policy. But it is not an acceptable policy for the domain corresponding to transition diagram Φ_1, as there is a better option. In Φ_1 if one were to execute a_1 in s_1 then one is guaranteed to reach s_2 where p is true. Hence, with respect to Φ_1 only the policy $\pi' = \{(s_1, a_1), (s_2, a_2)\}$ is an acceptable policy.

The following proposition shows that the above discussed goal of 'guaranteeing to reach p if that is possible and if not then making sure that p is always reachable' cannot be expressed using π-CTL*. For that the following lemma is needed.

Lemma 2. *Consider Φ_1, Φ_2 in Figure 3.4, and $\pi = \{(s_1, a_2), (s_2, a_2)\}$.*

(i) For any state formula φ in π-CTL*, $(s_1, \Phi_1, \pi) \models \varphi$ iff $(s_1, \Phi_2, \pi) \models \varphi$.

(ii) For any path formula ψ in π-CTL* and any path σ in Φ_1 (or Φ_2), $(s_1, \Phi_1, \pi, \sigma) \models \psi$ iff $(s_1, \Phi_2, \pi, \sigma) \models \psi$.

Proof. The proof is based on the induction on the depth of formulas.

Base case: It is easy to see that (i) and (ii) are true for formulas of depth 1.

Induction: Assume that (i) and (ii) are true for formulas of depth less than n, and show that (i) and (ii) are true for formulas of depth n.

Consider state formulas of depth n. It can be of the following forms: (a) $sf_1 \wedge sf_2$ (b) $sf_1 \vee sf_2$ (c) $\neg sf_1$ (d) $\mathsf{E}pf$ (e) $\mathsf{A}pf$ (f) $\mathsf{E}_{pol}pf$ (g) $\mathsf{A}_{pol}pf$, where sf_1, sf_2 and pf have depth less than n.

Consider (d) $\mathsf{E}pf$. By definition, $(s_1, \Phi_1, \pi) \models \mathsf{E}\ pf$ iff there exists a path σ in Φ_1 starting from s_1 such that $(s_1, \Phi_1, \pi, \sigma) \models pf$. *It is observed that σ is a path starting from s_1 in Φ_1 iff σ is a path starting from s_1 in Φ_2.* Since depth of pf is less than n, by induction hypothesis, $(s_1, \Phi_1, \pi, \sigma) \models pf$ iff $(s_1, \Phi_2, \pi, \sigma) \models pf$. Hence,

$(s_1, \Phi_1, \pi, \sigma) \models \mathsf{E}pf$ iff $(s_1, \Phi_2, \pi, \sigma) \models \mathsf{E}pf$.

The proofs for formulas of other forms are similar.

Consider path formulas of depth n. It can be of the following forms: (a) $pf_1 \wedge pf_2$ (b) $pf_1 \vee pf_2$ (c) $\neg pf_1$ (d) $pf_1 \mathsf{U} pf_2$ (e) $\bigcirc pf_1$ (f) $\diamond pf_1$ (g) $\Box pf_1$, where pf_1 and pf_2 have depth less than n. The proof of each of these cases is similar to the proof for state formulas. □

Proposition 4. *There is a goal defined as a mapping from a state and a transition function to a set of set of trajectories that cannot be expressed in π-CTL*.*

Proof. This proposition is proved by defining a goal and proving it cannot be expressed in π-CTL*. Consider the goal g that maps (s_1, Φ_1) of Figure 4 to the set of set of trajectories expressed by $\{s_1 s_2 s_2^*\}$ and maps (s_1, Φ_2) of Figure 4 to the set of set of trajectories expressed by $\{s_1 s_1^* s_2 s_2^*\}$. This is denoted as $g(s_1, \Phi_1) = \{s_1 s_2 s_2^*\}$, and $g(s_1, \Phi_2) = \{s_1 s_1^* s_2 s_2^*\}$. Now show by contradiction that goal g cannot be expressed in language π-CTL*.

Otherwise, suppose g can be expressed in π-CTL* as φ. According to Formula 3.1, if goal g can be expressed as formula φ in π-CTL*, Let $g(s, \Phi) = \varphi(s, \Phi)$ in π-CTL* for all state s and transition graph Φ. Thus $\varphi(s_1, \Phi_1) = \{s_1 s_2 s_2^*\}$ and $\varphi(s_1, \Phi_2) = \{s_1 s_1^* s_2 s_2^*\}$.

Let policy π be $\{(s_1, a_2), (s_2, a_2)\}$.

Now show that $(s_1, \Phi_1, \pi) \not\models \varphi$. According to the definition, $\varphi(s, \Phi)$ denotes the set of set of trajectories

$\{\pi_\sigma : (s, \Phi, \pi) \models \varphi$ and π_σ is the set of trajectories that are consistent with policy $\pi\}$.

If $(s_1, \Phi_1, \pi) \models \varphi$ in π-CTL*, the set of trajectories consistent with π in (s_1, Φ_1) is in $\varphi(s_1, \Phi_1)$. However, the set of trajectories consistent with π in (s_1, Φ_1) is $s_1 s_1^* s_2 s_2^*$, and $s_1 s_1^* s_2 s_2^* \notin \varphi(s_1, \Phi_1)$. Thus $(s_1, \Phi_1, \pi) \not\models \varphi$.

Now show that $(s_1, \Phi_2, \pi) \models \varphi$. According to the definition, $\varphi(s, \Phi)$ denotes

the set of set of trajectories

$\{\pi_\sigma : (s,\Phi,\pi) \models \varphi$ and π_σ is the set of trajectories that are consistent with policy $\pi\}$. As $s_1 s_1^* s_2 s_2^* \in \varphi(s_1,\Phi_2)$, there is a policy π' in Φ such that $(s_1,\Phi_2,\pi') \models \varphi$ and the set of trajectories consistent with π' is $s_1 s_1^* s_2 s_2^*$. π is the only policy such that the set of trajectories consistent with it is $s_1 s_1^* s_2 s_2^*$. Thus $\pi = \pi'$ and $(s_1,\Phi_2,\pi) \models \varphi$.

Thus $(s_1,\Phi_1,\pi) \not\models \varphi$ and $(s_1,\Phi_2,\pi) \models \varphi$. According to Lemma 2, for all formulas φ in π-CTL*, $(s,\Phi_1,\pi) \models \varphi$ iff $(s,\Phi_2,\pi) \models \varphi$.

There is a contradiction. Hence the goal g cannot be expressed in π-CTL*. □

The goal defined in the proof satisfies the following requirement:

"*All along your trajectory*

<u>if</u> from any state p can be achieved for sure

<u>then</u> the policy being executed must achieve p,

<u>else</u> the policy must make p reachable from any state in the trajectory."

While the <u>then</u> and <u>else</u> part of this goal can be expressed in π-CTL*, the <u>if</u> part can be further elaborated as "*there exists a policy which guarantees that p can be achieved for sure*", and to express that, one needs to quantify over policies. Thus a new existence quantifier $\mathscr{EP}$ and its dual $\mathscr{AP}$ are introduced, meaning 'there exists a policy starting from the state' and 'for all policies starting from the state' respectively.

Syntax of P-CTL*

The syntax of π-CTL* is extended to incorporate the above mentioned two new quantifiers.

Definition 19. *Let $\langle p \rangle$ denote an atomic proposition, $\langle sf \rangle$ denote a state formula, and $\langle pf \rangle$ denote a path formula. Intuitively, state formulas are properties of states, path formulas are properties of paths. With that the syntax of state and path formulas in P-CTL* is as follows.*

$\langle sf \rangle ::= \langle p \rangle \mid \langle sf \rangle \wedge \langle sf \rangle \mid \langle sf \rangle \vee \langle sf \rangle \mid \neg \langle sf \rangle \mid \mathsf{E}\langle pf \rangle \mid \mathsf{A}\langle pf \rangle$

$\mid \mathsf{E}_{pol}\langle pf \rangle \mid \mathsf{A}_{pol}\langle pf \rangle \mid \mathscr{EP}\langle sf \rangle \mid \mathscr{AP}\langle sf \rangle$

$\langle pf \rangle ::= \langle sf \rangle \mid \langle pf \rangle \vee \langle pf \rangle \mid \neg \langle pf \rangle \mid \langle pf \rangle \wedge \langle pf \rangle \mid \langle pf \rangle \,\mathsf{U}\, \langle pf \rangle \mid \bigcirc \langle pf \rangle \mid \Diamond \langle pf \rangle \mid \Box \langle pf \rangle$

$\Box$

Note that in the above definition $\mathscr{EP}\langle sf \rangle$ is a state formula. That is because once the policy part of $\mathscr{EP}$ is instantiated, the reminder of the formula is still a property of a state. The only difference is that a policy has been instantiated and that policy needs to be followed in the reminder of the formula unless specified otherwise. The semantics of P-CTL* is defined as follows.

Semantics of P-CTL*

The semantics of P-CTL* is related to the semantics of π-CTL*.

Definition 20 (Truth of state formulas)**.** *The truth of state formulas are defined with respect to a triple (s_j, Φ, π) where s_j is a state, Φ is the transition function, and π is a policy as a mapping from states to actions.*

- $(s_j, \Phi, \pi) \models p$ *iff* p *is true in* s_j.
- $(s_j, \Phi, \pi) \models \neg sf$ *iff* $(s_j, \Phi, \pi) \not\models sf$.
- $(s_j, \Phi, \pi) \models sf_1 \wedge sf_2$ *iff* $(s_j, \Phi, \pi) \models sf_1$ *and* $(s_j, \Phi, \pi) \models sf_2$.
- $(s_j, \Phi, \pi) \models sf_1 \vee sf_2$ *iff* $(s_j, \Phi, \pi) \models sf_1$ *or* $(s_j, \Phi, \pi) \models sf_2$.
- $(s_j, \Phi, \pi) \models \mathsf{E}\ pf$ *iff there exists a path* σ *in* Φ *starting from* s_j *such that* $(s_j, \Phi, \pi, \sigma) \models pf$.
- $(s_j, \Phi, \pi) \models \mathsf{A}\ pf$ *iff* $(s_j, \Phi, \pi, \sigma) \models pf$ *for all paths* σ *in* Φ *starting from* s_j.
- $(s_j, \Phi, \pi) \models \mathsf{E}_{pol}\ pf$ *iff there exists a path* σ *in* Φ *starting from* s_j *consistent with the policy* π *such that* $(s_j, \Phi, \pi, \sigma) \models pf$.

- $(s_j, \Phi, \pi) \models \mathsf{A}_{pol}\ pf$ *iff* $(s_j, \Phi, \pi, \sigma) \models pf$ *for all paths* σ *in* Φ *starting from* s_j *consistent with the policy* π.

- $(s_j, \Phi, \pi) \models \mathscr{E}\mathscr{P}\ sf$ *iff there exists a policy* π' *being a mapping from states to actions consistent with* Φ *such that* $(s_j, \Phi, \pi') \models sf$.

- $(s_j, \Phi, \pi) \models \mathscr{A}\mathscr{P}\ sf$ *iff* $(s_j, \Phi, \pi') \models sf$ *for all policies* π' *that are mappings from states to actions consistent with* Φ. □

Definition 21 (Truth of path formulas)**.** *The truth of path formulas are now defined with respect to the quadruple* (s_j, Φ, π, σ)*, where* s_j, Φ *and* π *are as before and* σ *is an infinite sequence of states* $s_j, s_{j+1}, \ldots$*, called a path.*

- $(s_j, \Phi, \pi, \sigma) \models sf$ *iff* $(s_j, \Phi, \pi) \models sf$.

- $(s_j, \Phi, \pi, \sigma) \models \neg pf$ *iff* $(s_j, \Phi, \pi, \sigma) \not\models pf$

- $(s_j, \Phi, \pi, \sigma) \models pf_1 \wedge pf_2$ *iff* $(s_j, \Phi, \pi, \sigma) \models pf_1$ *and* $(s_j, \Phi, \pi, \sigma) \models pf_2$.

- $(s_j, \Phi, \pi, \sigma) \models pf_1 \vee pf_2$ *iff* $(s_j, \Phi, \pi, \sigma) \models pf_1$ *or* $(s_j, \Phi, \pi, \sigma) \models pf_2$.

- $(s_j, \Phi, \pi, \sigma) \models \bigcirc pf$ *iff* $(s_{j+1}, \Phi, \pi, \sigma) \models pf$.

- $(s_j, \Phi, \pi, \sigma) \models \Box pf$ *iff* $(s_k, \Phi, \pi, \sigma) \models pf$*, for all* $k \geq j$.

- $(s_j, \Phi, \pi, \sigma) \models \Diamond pf$ *iff* $(s_k, \Phi, \pi, \sigma) \models pf$*, for some* $k \geq j$.

- $(s_j, \Phi, \pi, \sigma) \models pf_1\ \mathsf{U}\ pf_2$ *iff there exists* $k \geq j$ *such that* $(s_k, \Phi, \pi, \sigma) \models pf_2$*, and for all* i*,* $j \leq i < k$*,* $(s_i, \Phi, \pi, \sigma) \models pf_1$. □

Now define when a policy π that maps from states to actions satisfies P-CTL* goal φ given an initial state s_0, and a transition function Φ.

Similar to the definitions in π-CTL*, for a P-CTL* formula φ, a policy π satisfies a goal φ from s_0, if $(s_0, \Phi, \pi) \models \varphi$ in P-CTL*. Let the set $\varphi(s, \Phi)$ denotes the

set of set of trajectories

$\{\pi_\sigma : (s, \Phi, \pi) \models \varphi$ and π_σ is the set of trajectories that are consistent with policy $\pi\}$.

For any transition function Φ, a path σ in Φ and for all policies π that map from states to actions, and a formula φ in π-CTL*, $(s_0, \Phi, \pi, \sigma) \models \varphi$ in π-CTL* iff $(s_0, \Phi, \pi, \sigma) \models \varphi$ in P-CTL*. Which implies that with each policy being a mapping from states to actions, all goals that can be expressed in π-CTL* can be expressed in P-CTL*. Considering Proposition 4, P-CTL* is a proper superset of language π-CTL* and is strictly more expressive. More on this will be discussed in Section 3.5.

Goal Representation in P-CTL*

Several goal examples that can be expressed in P-CTL* while cannot be expressed in π-CTL* or other languages such as CTL* is now illustrated.

Section 3.2 explored various goals that can be expressed in π-CTL*. Based on goal specifications G_s^π, G_w^π, and G_{sc}^π, the new quantifiers in P-CTL* is used to express conditions similar to the ones mentioned in the beginning of Section 3.3.

$C_w = \mathscr{E}\mathscr{P}\mathsf{E}_{pol}\Diamond p$: This is a state formula, which characterizes states with respect to which (i.e., if that state is considered as an initial state) there is a policy such that if one were to follow that policy then one can, but not guaranteed to, reach a state where p is true. Similarly, define $C_s = \mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p$, and $C_{sc} = \mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)$.

These three formulas are not expressible in π-CTL*, and are state formulas of P-CTL*. But, by themselves they, or a conjunction, disjunction or negation of them, are not meaningful goal formulas with respect to which one would try to develop policies (or plan) for. Nevertheless, they are very useful building blocks.

Recall the transition function in the proof of Proposition 4, given as:

All along your trajectory,

if from any state p can be achieved for sure,

then the policy being executed must achieve p,

else the policy must make p reachable from any state in the trajectory.

Now the above goal in P-CTL* can be expressed as $\mathsf{A}_{pol}\Box((\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p \Rightarrow \mathsf{A}_{pol}\Diamond p) \wedge (\neg\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p \Rightarrow \mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)))$.

The policy π in Figure 3.4 defined as $\pi(s_1) = \pi(s_2) = a_2$ achieves the above goal with respect to Φ_2, but not with respect to Φ_1, while the policy π' defined as $\pi'(s_1) = a_1$, and $\pi'(s_2) = a_2$ achieves the above goal with respect to Φ_1. The reason π does not satisfy the goal with respect to Φ_1, is that $\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p$ is true with respect to s_1 (in Φ_1), but the policy π does not satisfy $\mathsf{A}_{pol}\Diamond p$.

Goals Corresponding to Example 2

Now use P-CTL* formulas C_s, C_w and C_{sc} and π-CTL* formulas G_s^π, G_c^π, and G_{sc}^π to express various goals with respect to Example 2.

- $G_w^P = \mathsf{A}_{pol}\Box(\mathscr{E}\mathscr{P}\mathsf{E}_{pol}\Diamond p \Rightarrow \mathsf{E}_{pol}\Diamond p)$: This goal specifies that all along the trajectory following the given policy, if there is a policy that makes p reachable then the given policy makes p reachable. The policies π_1, π_2, π_3 and π_4 satisfy this goal while π_5 does not.

  As a rarity, the π-CTL* goal $G_w^\pi = \mathsf{A}_{pol}\Box(\mathsf{E}\Diamond p \Rightarrow \mathsf{E}_{pol}\Diamond p)$ also satisfies these four policies.

- $G_s^P = \mathsf{A}_{pol}\Box(\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p \Rightarrow \mathsf{A}_{pol}\Diamond p)$: This goal specifies that all along the trajectory following the given policy, if there is a policy that can always reach p no matter the non-deterministic actions, then in the policy chosen by the agent, p must be reached." The policies π_1, π_2 and π_5 satisfy this goal while π_3 and π_4 do not.

- $G_{sc}^P = \mathsf{A}_{pol}\Box(\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p) \Rightarrow \mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p))$: This goal specifies that all along the trajectory following the given policy, if there is a policy

that is a strong cyclic policy for p, then the policy chosen by the agent is a strong cyclic policy for p." The policies π_1, π_3, and π_5 satisfy this goal while policies π_2 and π_4 do not.

- $G_s^P \wedge G_c^P \wedge G_{sc}^P$: This goal specifies that all along the trajectory following the given policy, if there is a policy that guarantees that p will be reached, then the agent's policy must guarantee to reach p; else-if there is a strong cyclic policy for p, then the policy chosen by the agent must be a strong cyclic policy; and else-if there is a policy that makes p reachable then the policy makes p reachable. This can be considered as one formal specification of the goal of "trying one's best to reach p". Only π_1 among $\pi_1 - \pi_5$ satisfies this goal.

Goal	Satisfiable policies
G_w^π, G_w^P	π_1, π_2, π_3, π_4
G_{sc}^P	π_1, π_3, π_5
G_s^P	π_1, π_2, π_5
$G_w^P \wedge G_s^P$	π_1, π_2
$G_w^P \wedge G_{sc}^P$	π_1, π_3
$G_w^P \wedge G_s^P \wedge G_{sc}^P$	π_1
$G_s^P \wedge \neg G_{sc}^P$	π_2
$G_{sc}^P \wedge \neg G_s^P$	π_3
$G_w^P \wedge \neg G_{sc}^P \wedge \neg G_s^P$	π_4
$G_s^P \wedge \neg G_w^P$	π_5
G_s^π	$\emptyset$

Table 3.1: Different P-CTL* and π-CTL* goal specifications and the policies satisfying them

Based on these formulations, users may have various specifications. Some of these specifications and the subset of the policies $\pi_1 - \pi_5$ that satisfy these goals are summarized in Table 3.1. In this example, users may have an arbitrary partition of $\{\pi_1, \cdots, \pi_5\}$, while most of these partitions cannot be done in existing

languages. Language P-CTL* is more powerful in expressing the intention of comparing among policies.

Maintenance Goals and Other Goals Specified in P-CTL*

In expressing goals about maintainability, the relations for π-CTL* goals still hold here. $\mathsf{A}_{pol}\phi$ is equivalent to $\neg\mathsf{E}_{pol}\neg\phi$ for any path formula ϕ. Besides, $\mathscr{A}\mathscr{P}\phi$ is equivalent to $\neg\mathscr{E}\mathscr{P}\neg\phi$.

Some goals that involve two subgoals, p and q are specified in P-CTL*. They illustrate that the additional expressive power of P-CTL* is not just for expressing the "if-then" type of conditions discussed earlier.

- Suppose there is an agent that would like to reach q but wants to make sure that all along the path if necessary it can make a new (contingent) policy that can guarantee that p will be reached. Here, q may be the destination of the robot and p may be the property of locations that have recharging stations. This goal can be expressed in P-CTL* as $\mathsf{A}_{pol}\Box((\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p)\mathsf{U}q)$. Alternative specifications in CTL* or π-CTL* cannot capture this goal.

- Consider an agent that would like to reach either p or q, but because of non-determinism the agent is satisfied if all along its path at least one of them is reachable, but at any point if there is a policy that guarantees that p will be reached then from that point onwards the agent would like to make sure that p is reached; otherwise, if at any point if there is a policy that guarantees that q will be reached then from that point onwards the agent would like to make sure that q will be reached. This can be expressed in P-CTL* as $\mathsf{A}_{pol}\Box(\mathsf{E}_{pol}(p\vee q)\wedge(\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p\Rightarrow\mathsf{A}_{pol}\Diamond p)\wedge((\neg\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p\wedge\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond q)\Rightarrow\mathsf{A}_{pol}\Diamond q))$.

- Consider an agent whose goal is to maintain p true and if that is not possible

for sure then it must maintain q true until p becomes true. This can be expressed in P-CTL* as $\mathsf{A}_{pol}\Box((\mathscr{A}\mathscr{P}\mathsf{E}_{pol}\neg\Box p \Rightarrow \mathsf{A}_{pol}(q\mathsf{U}p)) \wedge (\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Box p \Rightarrow \mathsf{A}_{pol}\Box p))$.

The proposed language P-CTL* allows the specification of such goals. P-CTL* has the ability of letting the agent to compare and analyze policies and "adjust" accordingly. Hence, it is useful for the agent to plan in a non-deterministic or dynamic domains in which current states are unpredictable.

Although P-CTL* is a rich goal specification language, it still has limitations. These limitations are partly due to the policy defined in the language. The next section formally elaborates on having a different policy structure in defining a language.

3.4 P_σ-CTL*: Need for Different Notions of the Policy Structure

In languages π-CTL* and P-CTL*, the policy structure is defined as a mapping from states to actions. Now illustrate that the definition on policy structure in a language has a great impact on the set of goals expressed in the language.

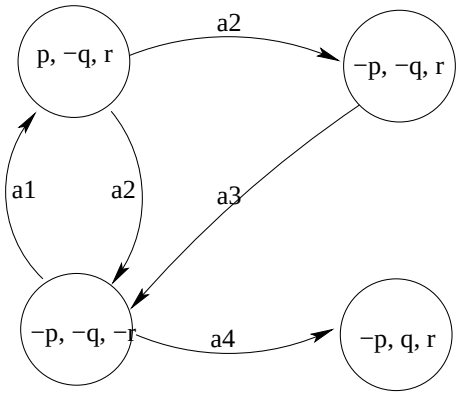

Figure 3.5: A transition with different policy structures

For example, in Figure 3.5, suppose the goal g has a property that it maps transition graph Φ and initial state s_1 to the sets of set of trajectories $\{s_1s_2s_3s_1s_4s_4^* \cup$

$s_1 s_2 s_1 s_4 s_4^*\}$. This goal states that the agent needs to reach (a state where) p (is true) first and then reach q.

This goal cannot be expressed by a formula in P-CTL* due to the policy defined in the language. By analyzing the transition graph, given that the agent is initially in state s_1, the agent can take the following strategy that corresponds to the set of trajectories $s_1 s_2 s_3 s_1 s_4 s_4^* \cup s_1 s_2 s_1 s_4 s_4^*$. The agent can initially execute action a_1 in state s_1, and then taking action a_2 in the resulting state s_2. If a_2 happens to take the agent to state s_1, the agent then takes action a_4. If the execution of a_2 in state s_2 takes the agent to state s_3, the agent should then execute a_3 followed by a_4 to reach state s_4. However the strategy described above is not a mapping from states to actions. It takes different actions at s_1 the two times it is there. It is not an action sequence either as action a_2 has non-deterministic effects and no common actions can be executed in the resulted states. In fact, given that the initial state is s_1, there is no action sequences or mappings from states to actions to satisfy the requirement of reaching p and then reaching q in the domain. In order to capture the strategy described above, a policy as a mapping from state sequences to actions is defined.

Definition 22 (Policy as a mapping from state sequences to actions)**.** *A* policy π *is a mapping from each sequence of finite number of states* T *to an action* A*. A policy is* valid *if for each trajectory* $\sigma \in T$ *of the form* $s_0, s_1, \cdots$*, for* $i \geq 0$*, it is true that* $s_{i+1} \in \Phi(s_i, \pi(s_0, s_1, \cdots, s_i))$. □

Having policies as mappings from state sequences to actions is not new. There are some similar definitions in the literature. It is often called a strategy, an algorithm, or a protocol [HF85, AHK02]. In [BDH99], a policy is defined as a mapping from state-action sequences to actions. Each state-action sequence is a sequence of states and actions from the initial state to the current state. In [DLPT02], a *context* is attached to each state, which encodes the properties of historical states. As users

usually do not care about actions taken in the past in a goal specification language, a policy is now defined as a mapping from state sequences to actions instead of a mapping from state and action sequences to actions.

A trajectory consistents with a policy that is a mapping from state sequences to actions is now defined.

Definition 23 (Trajectories consistent with a policy)**.** *A trajectory* $\sigma = s_0, s_1, \cdots$ *is consistent with a policy* π *that maps state sequences to actions if* $s_{i+1} \in \Phi(s_i, \pi(s_0, s_1, \cdots, s_i))$ *for* $i \geq 0$. □

This notion of policies is related to the notion of policies as mappings from states to actions. If a policy that is a mapping from trajectories to actions has a property that trajectories with the same last states are mapped to the same action, it can be simplified to a mapping from states to actions. For each policy that maps states to actions, there is a policy that maps histories to actions such that a trajectory is consistent with one policy iff it is consistent with the other one.

With each policy being a mapping from state sequences to actions, goal specification languages π_σ-CTL* and P_σ-CTL* are defined in a similar approach as π-CTL* and P-CTL*. Relations of these languages are discussed in the next section.

Now consider the goal of reaching p and then reaching q in transition graph in Figure 3.5. The goal is represented in π_σ-CTL* as $\mathsf{A}_{pol}\Diamond(p \wedge \Diamond q)$. A policy $\{(s_1, a_1), (s_1s_2, a_2), (s_1s_2s_3, a_3), (s_1s_2s_1, a_4), (s_1s_2s_3s_1, a_4)\}$ in P_σ-CTL* satisfies this goal. This goal cannot be represented in language π-CTL* or P-CTL*.

Definitions of policy structures are not limited to the ones defined above. A policy may also defined as a mapping from states to sets of actions, or from pairs of LTL/CTL*/etc. formulas to actions, etc. A goal specification language may also defined such that the policy of the agent can be the combination of two other policy

definitions. The policy structure represents the architecture of the agent, it denotes the ability of the agent. For two agents with different abilities, the same instruction given to them may lead to different outcomes.

3.5 Expressiveness of a Goal Specification Language

In previous sections, different goal specification languages are proposed in representing goals, where each goal g is a mapping $g(s,\Phi)$ from a transition graph Φ and an initial state s to a set of trajectories (or a set of set of trajectories). A set of formulas are defined in each goal specification language. Goals are then represented by these formulas with the definition that a goal g is represented by a formula φ in the language if $g(s,\Phi) = \varphi(s,\Phi)$ for any transition graph Φ and state s. In each language, the definition of $\varphi(s,\Phi)$ implies the relations of goals and formulas. With this definition, this chapter showed that there is a goal in P-CTL* which cannot be expressed in π-CTL*, and there is a goal in P_σ-CTL* which cannot be expressed in P-CTL*.

In addition to this, a relation between formulas and policies is defined based on the entailment relation in each language. For example, in π-CTL*, policy π satisfies a goal φ from state s_0 in Φ if $(s_0,\Phi,\pi) \models \varphi$.

Comparing formulas and policies in different languages, two languages are different for different reasons. The set of formulas in π-CTL* is a proper subset of set of formulas in P-CTL*, and in both languages, each policy is a mapping from states to actions thus these two languages share the same set of policies. This relation is denoted as *syntax-advanced.*

On the other hand, the set of formulas in P-CTL* and the set of formulas in P_σ-CTL* are the same, while policies defined in P_σ-CTL* are mappings from trajectories to actions, and policies defined in P-CTL* are mappings from states to actions. As policies in P-CTL* is a proper subset of polices in P_σ-CTL*. this is

denoted as *policy-advanced*. These relations between goal specification languages are formally defined as follows.

Definition 24 (syntax-advanced)**.** *Given two languages L_1 and L_2, L_1 is* syntax-advanced *than L_2 if*

1. *the set of policies in both languages are the same for any transition system Φ and state s,*
2. *the set of goal formulas in L_2 is a proper subset of goal formulas in L_1, and*
3. *for a formula φ in L_2, an initial state s, and a transition system Φ, a policy π satisfies φ in L_2 iff it satisfies φ in L_1.* □

Definition 25 (policy-advanced)**.** *Given two languages L_1 and L_2, L_1 is* policy-advanced *than L_2 if*

1. *the set of policies in L_2 is a proper subset of policies in L_1 for any transition system Φ and state s_0,*
2. *the set of goal formulas in both languages are the same, and*
3. *for a policy π in L_2, an initial state s, and a transition system Φ, policy π satisfies a formula φ in L_2 iff it satisfies φ in L_1.* □

Based on the semantics of each language, P-CTL* is syntax-advanced than π-CTL*. To prove it, it is easy to check Item 1) and 2) in Definition 24. Item 3) is proved by showing that for a formula φ in π-CTL*, the set of policies satisfying φ in π-CTL* is the same as the set of policies satisfying φ in P-CTL*. That is, $(s,\Phi,\pi) \models \varphi$ in π-CTL* iff $(s,\Phi,\pi) \models \varphi$ in P-CTL*. This can be implied by the semantics of π-CTL* and P-CTL*. Similarly, P_σ-CTL* is syntax-advanced than π_σ-CTL*.

Similarly, π_σ-CTL* is policy-advanced than π-CTL*. To prove it, Item 1) and Item 2) in Definition 25 are easy to check. Item 3) states that a policy π in π-CTL* satisfies a formula φ in π-CTL* iff it satisfies the same formula in P-CTL*. This can be implied by the semantics of π-CTL* and π_σ-CTL*.

However, P_σ-CTL* is not policy-advanced than P-CTL*. In checking the Item 3) in Definition 25, for the same formula, semantics of $\mathscr{E}\mathscr{P}$ and $\mathscr{A}\mathscr{P}$ in two languages are different. They states the comparison of all policies in the language, while these two languages have different sets of policies.

Note that in languages π-CTL*, P-CTL*, π_σ-CTL*, and P_σ-CTL*, $\varphi(s,\Phi)$ are all defined as a set of set of trajectories
$\{\pi_\sigma : (s,\Phi,\pi) \models \varphi$ and π_σ is the set of trajectories that are consistent with policy $\pi\}$. Thus these languages can be compared based on the set of goals expressed in each of them.

Proposition 5. *Given a goal that is a mapping from states and transition graphs to sets of set of trajectories,*

- *A goal expressed in π-CTL* can be expressed in P-CTL*;*
- *A goal expressed in π_σ-CTL* can be expressed in P_σ-CTL*.*

Proof. Let $\varphi(s,\Phi)^{\pi-CTL*}$ be $\varphi(s,\Phi)$ in language π-CTL*. Let $\varphi(s,\Phi)^{P-CTL*}$ be $\varphi(s,\Phi)$ in language P-CTL*. Suppose a goal g can be expressed in π-CTL* as φ. $g(s,\Phi)$ is defined as $\varphi(s,\Phi)^{\pi-CTL*}$ for any state s and transition graph Φ. Now prove that $\varphi(s,\Phi)^{\pi-CTL*} = \varphi(s,\Phi)^{P-CTL*}$. As P-CTL* is syntax-advanced than π-CTL*, the set of policies satisfying φ in these two languages are the same, and policies defined in these two languages are the same. As $\varphi(s,\Phi)$ is defined as
$\{\pi_\sigma : (s,\Phi,\pi) \models \varphi$ and π_σ is the set of trajectories that are consistent with policy $\pi\}$,

$\varphi(s,\Phi)^{\pi-CTL*} = \varphi(s,\Phi)^{P-CTL*}$. Thus a goal expressed in π-CTL* can be expressed in P-CTL*.

It is similar to prove that a goal expressed in π_σ-CTL* can be expressed in P_σ-CTL*. □

However, there is no such relations between π-CTL* and π_σ-CTL* or between P-CTL* and P_σ-CTL*. For example, considering a goal that maps s_1 and Φ_1 to a set of set of trajectories $\{s_1 s_1^* s_2 s_2^*, s_1 s_2 s_2^*\}$ in transition graph Φ_1 of Figure 3.4. This goal can be be represented in π-CTL* as $\mathsf{A}_{pol}\Diamond p$. However, this goal cannot be represented as $\mathsf{A}_{pol}\Diamond p$ in π_σ-CTL*. The formula $\varphi = \mathsf{A}_{pol}\Diamond p$ maps s_1 and Φ_1 to a set of set of trajectories that consists of a lot more elements. For example, a set of trajectories $s_1 s_1 s_2 s_2^* \cup s_1 s_2 s_2^*$ is one element in $\varphi(s_1,\Phi_1)$. A policy in π_σ-CTL* that try action a_2 twice before taking action a_1 in state s_1 is a policy satisfying this goal in π_σ-CTL*.

There are other approaches of defining expressiveness of a goal specification language. In Appendix B, another approach of defining expressiveness of a goal specification language is proposed. It has different properties as the framework proposed above.

3.6 Discussion and Related Work

In this section, a few issues related to goal specifications are discussed. This section starts with the importance of the policy structure in a goal specification language.

Goal Specification with Different Policy Structures

The goal "try your best to reach p" has properties of comparing policies of the agent. Thus the agent need to be aware of the set of policies available to her before she can choose the best one. This goal cannot be captured by just comparing sets of trajectories in the transition graph, as some sets of trajectories may not be available to the agent even though they are the best trajectories.

As there may be different definition of policies in a transition system. Depending on different initial states and different transition system the agent is in, a goal is a mapping from the "possible ways the world could evolve *for the agent*" to "some desired ways". Each "possible ways the world could evolve for the agent" stands for a possible state structure that are available to the agent, which is usually ties to the policy structure of the agent. This implies that in some cases, users may not only need to consider how the world may evolve, but also need to consider how the world may evolve *for the agent*. This is interesting as different agents may have different policy structures. Thus for the agent to choose the "best options" among the ones that are available, the agent need to know all the options that are available to her. How the world may evolve for the agent can be expressed by a triple of an initial state s, a transition function Φ, and a policy structure $\mathscr{P}$ of the agent. In previous section, when one specifies a goal in the languages π-CTL*, π_σ-CTL*, P-CTL* and P_σ-CTL*, a particular policy structure is assumed implicitly or explicitly. However, sometimes, users may have a requirement in mind while users are not aware of the particular ability (i.e., policy structure) of the agent. Thus the following definition on goals might be needed.

Definition 26. *A goal g is a mapping from triples of initial state s, transition function Φ, and policy structure $\mathscr{P}$ to sets of trajectories (or sets of set of trajectories), which is denoted as $g(s,\Phi,\mathscr{P})$.*

Let g be a formula in goal specification language L, s be the initial state, Φ be the transition function. Now a goal can be expressed as a formula φ in language L if

$$\varphi(s,\Phi,\mathscr{P}) = g(s,\Phi,\mathscr{P}).$$

With this definition, when a goal is given to the agent, users do not need to be aware of the policy of the agent. Each agent can interpret the requirement based on

its own policy structure and then choose the corresponding policies to execute.

Limitations of Goal Specification with Temporal Logics

A few limitations of current goal specification with temporal logic approach are listed now. All temporal logic approach in goal specification share these limitations.

Firstly, there is no elegant way of comparing states explicitly. Current goal specification languages only specify properties of the policy taken by the agent in terms of relations of fluents. There is not explicit representation and comparison of states, which is necessary in some cases. For example, users may want to make sure that the agent stays in the first state where fluent p is reached, or users may want to prevent the agent from visiting the same state twice. Representing such a requirement in an elegant way is a challenge problem.

Secondly, current goal specification languages are not good at handling paths consist of a finite number of states. For example, users want the agent to find a policy such that the agent cannot reach p, but must stay in the first state from where there is a policy that guarantees to reach p (e.g., a state s.t. $\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p$). In a non-deterministic domain, users can try to encode the goal in P-CTL* as $(\mathsf{A}_{pol}\Diamond\Box(\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p)) \wedge \mathsf{A}_{pol}\Box\neg p$ but the formula does not exactly capture the intention of staying in that state. Given a finite state sequence, it can be extend to an infinite state sequence by appending the last state, which is the result of the action "*nop*". On the other hand, given an infinite state sequence, it is not easy to get a properly defined state sequence of a finite number of states.

Complexity Issues

So far in this chapter the issue of complexity of planning and plan checking with respect to goals in the various proposed languages has not been explored. The complexity results not only depend on the goal language but also on how the transition diagram is encoded. This section points to some of the earlier papers on complex-

ity with respect to temporal logics [WD05, JL03, LMO06, BKT01] and presents one sample result where the transition diagram is encoded using an action language. In particular, the language considered is STRIPS+, an extension of STRIPS representation of the transition system in [FG00] that allows actions to have non-deterministic effects. In short, actions in STRIPS+ composed of preconditions in *pre*, deterministic effects in d_eff, and non-deterministic effects in i_eff. The input and output of the problem is now defined:

Given an action signature $\langle \mathscr{V}, \mathscr{F}, \mathscr{A} \rangle$, a state s_0 in S denoted as the initial state, the transition function Φ defined by the action language in [FG00], and a temporal formula g in goal specification language L, the *Plan Existence Checking* problem is about deciding whether there is a policy π such that $(s_0, \Phi, \pi) \models_L g$.

Proposition 6. *Deciding whether there is a policy in a non-deterministic domain that satisfies a π-CTL* formula is EXPTIME-hard.*

Proof. To prove that the problem is EXPTIME-hard, EXPTIME-complete problem G_4 [SC79] is reduced to a plan existence checking problem.

In a G_4 problem, a $13DNF$ formula f and two sets of variables are given as input. There are two players in the game. Each play has one set of variables. Each player can choose one variable belong to him and flip it. Two players take turns with passing allowed in flipping variables. The output of G_4 problem is true if the first player has a policy to guarantee winning the game. The output is false if no such policy exists for the first player.

The translation from a G_4 problem to a plan existence checking problem is defined as follows:

For each variable r in the G_4 problem, there is a fluent r in the non-deterministic planning problem. Let the set of r fluents corresponding to player A be R_A. The set of r fluents corresponding to player B be R_B. Besides, there is an extra fluent A_{turn}

for the planning problem. It indicates whether it is player A's turn to execute the next action.

In the planning problem, there are two sets of different actions. Conceptually, one set of actions corresponds to actions of player A, while the other set player B. One action belonging to player A can be executed only if A_{turn} is true in the state. On the other hand, one action belonging to player B can be executed only if A_{turn} is not true in the state. The set of actions executable by player A are deterministic actions. The number of actions executable by player A doubles the size of R_A. For each fluent r in R_A, there are two actions $flip_{rt}$ and $flip_{rf}$ such that they will make the fluent r true if it is false, and make it false if it is true.

$$\begin{aligned} flip_{rt} \quad & pre: A_{turn}, \neg r \\ & d\_eff: \neg A_{turn}, r \\ flip_{rf} \quad & pre: A_{turn}, r \\ & d\_eff: \neg A_{turn}, \neg r \end{aligned}$$

Besides, there is a dummy action, which corresponds to the pass through of player A:

$$\begin{aligned} dummy_A \quad & pre: A_{turn} \\ & d\_eff: \neg A_{turn} \end{aligned}$$

Only one action can be executed when A_{turn} is not true, it changes at most one fluent from R_B.

$$\begin{aligned} action_B \quad & pre: \neg A_{turn} \\ & d\_eff: A_{turn} \\ & i\_eff: \{r_{b1}\}, \{\neg r_{b1}\}, \cdots, \{r_{br}\}, \{\neg r_{br}\} \end{aligned}$$

Where $r_{b1}, \cdots, r_{br}$ are all fluents in R_B. Note that $action_B$ may not chance value of any fluent, which corresponds to the pass through of player B. When $action\_B$ changes value of fluents, it can only change value of at most one fluent at a time. Length of the $action\_B$ is polynomial to the number of fluents in R_B. It is a polynomial time translation. Note that any A's action will make A_{turn} false and the action $action_B$ will make A_{turn} true. By this, player A and B take turns in flipping fluents.

Given the formula in G_4 problem being f, the π-CTL* goal to be checked is $\mathsf{A}_{pol}(\neg(A_{turn} \wedge f)\mathsf{U}(\neg A_{turn} \wedge f))$. The claim is that there is a policy for player A iff there is a policy, i.e., a mapping from states to actions, to satisfy the goal in the transformed planning domain.

Firstly, it is easy to see that it is a polynomial time reduction. Now prove the correspondence of these problems:

If there is a policy for player A to win the game, then player A has a policy that while execute the action he choose, no matter what player B execute, will guarantee to reach a state where f is true while in the process of reaching that state, there is no state that person B makes f true. Note that the policy taken by player A must be a mapping from states to actions.

On the other hand, if there is a policy satisfying the π-CTL* goal, there is a policy for player A. □

From the proof, it is known that the program can be encoded with a P-CTL* formula. Thus deciding whether there is a policy in a non-deterministic domain that satisfies a P-CTL* formula is EXPTIME-hard.

Now define the policy checking problem to check whether a goal is satisfied by a policy. Given an action signature $\langle \mathscr{V}, \mathscr{F}, \mathscr{A} \rangle$, a state s_0 in S denoted as the initial state, the transition function Φ defined by the action language in [FG00], a policy π that is a mapping from states to actions (or a policy that is an action

sequence, or a mapping from state sequences to actions), and a temporal formula g in goal specification language L, the em Policy Checking problem is about deciding whether or not $(s_0, \Phi, \pi) \models g$ in language L.

Proposition 7. *The policy checking problems for π-CTL* is PSPACE-complete.*

Proof. It is known that the model checking problem for CTL* is PSPACE-complete [EL85, Sch03] in deterministic domain. The model checking for LTL is PSPACE-hard [SC85]. Similarly, the model checking for π-CTL* is PSPACE-hard as well.

The following shows that the policy checking problem for π-CTL* is in PSPACE. An algorithm is constructed based on the algorithm for CTL*. Two transition functions are defined. They proceed simultaneously. One of the transition function is Φ, and the other one is Φ_π, the transition function corresponding to the policy π of the agent. Take a similar approach as in Section 4 of [AHK02], for a formula φ, all its sub-formulas are considered. Label each state in Φ and Φ_π with all sub-formulas of φ that are satisfied in the state. A sub-formula is constructed recursively. If the sub-formula is constructed by preceding A_{pol} or E_{pol}, check and update on the transition function Φ_π. If the sub-formula is constructed by preceding A or E, check and update on the transition function Φ. All the remaining are the same as CTL* in both transition function. As the checking for CTL* is in P-SPACE, the policy checking problems for π-CTL* is in P-SPACE. □

Related Works

In the history of computer science, there has been a lot of research in specifying purpose of programs, and proving correctness of programs with respect to given specification, using temporal logic. Temporal logics were used for specification and verification of concurrent systems and some of the work are presented in the books such as [CM88, MP92] and surveyed in [Eme90]. Some work has also been done on automatically and semi-automatically synthesizing [MW84, CE81, PR89, ES84]

parts of a concurrent programs. Most of the work is on extending logics to have metric intervals [BK98], qualitative measure on elapsed time between the occurrences of the events [Pnu77, AH93], or having a timed transition system [DCDS01], or in a game-like multi-agent system [AHK02, vdHJW05]. There are some work on making use of temporal connectives for specifying the control programs of semi-autonomous systems. The early research on this include the state based temporal logic in [McD82b], the interval based logic in [All84], and the interval based generalization of state based temporal logic in [Sho87]. How to use temporal logic to specify goals of agents is not well studied in those directions.

Different from those work, in using temporal logics in goal specification, this chapter points out that as richer and richer goal specification languages are developed, languages that are intimately associated with the policy structure of the agent are needed. For agents with different policy structures, or abilities, the same intuition might have different interpretations. For example, if a user asks the agent to try its best to reach p, one agent may not even execute a single action as long as he can convince the user that he is not able to reach p. In that case, the user still consider the agent as tried his best. The other agent in the same state might have to reach p as he has a different policy structure and is capable of reaching p.

Language π-CTL* captures the intuition of grouping the set of trajectories associated with the policy under consideration. Some constructs from [DLPT02] capture the same intuition. However their language is somewhat orthogonal to temporal logics and as a result using their language one cannot build up on the existing expressiveness of temporal logics such as LTL and CTL*. On the other logics proposed in this chapter build up on existing temporal logics. In [BZ04] a more detailed and critical analysis of their language is given. The paper also points point out additional limitations of the work in [DLPT02]. Similar to π-CTL*, in [dLPdB08], logic α-CTL* was proposed by branching on actions instead of the set of trajecto-

ries in a policy.

Quantification over policies was proposed in the context of games in the language ATL and ATL* [AHK02]. An extension of that called CATL [vdHJW05] has also been proposed. They are similar to P-CTL*. However, their focus in on games. If a user considers the domain to be non-deterministic domain, there is no easy way of single out each path. If a single deterministic domain is considered, it is not obvious that one can have a 1-1 correspondence between those formalisms and ours as the transition considered in P-CTL* is non-deterministic. For example, one may have a translation from this formalism (one person, but with non-deterministic transitions) to their formalism (two person games with deterministic transitions) to take care of the non-deterministic effects of actions. For example, the action a_1 and a_6 in state s_1 in Figure 3.1 is translated to actions in Figure 3.6.

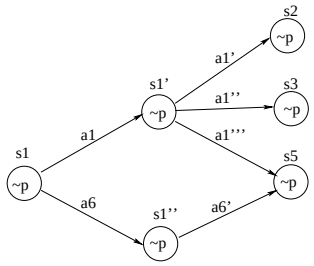

Figure 3.6: Differences of ATL and P-CTL* in specifying goals

Note that states $s_1^{'}$ and $s_1^{''}$ are new states and $a_1^{'}$, $a_1^{''}$, $a_1^{'''}$ and $a_6^{'}$ are new deterministic actions. a_1 and a_6 are actions of the first agent and $a_1^{'}$, $a_1^{''}$, $a_1^{'''}$ and $a_6^{'}$ are actions belong to the second agent. A similar translation for other states and actions can be done. With this translation, their formalism can be used to take care of non-deterministic actions. However, their formalism cannot be used to represent all goals in π-CTL* and P-CTL*. In P-CTL*, $\mathsf{E}f$ and $\mathscr{E}\mathscr{P}\mathsf{E}_{pol}f$ correspond to different set of paths. The latter only consider the paths that follows a policy while the first formula consists of all possible paths in the domain. Similarly, $\mathsf{A}f$ and

$\mathscr{A}\mathscr{P}\mathsf{A}_{pol}\,f$ have different meaning. In general, P-CTL* allows non-deterministic domains, it distinguishes between the definitions of all paths and the set of paths due to the agent actions while this distinction is not captured in ATL, ATL* or CATL. However, it might be true that their formalism can be used to represent goals in *P*-CTL*. Definition of the policy plays an important role in these languages while the impact of policy definitions is not considered in ATL and ATL*.

3.7 Summary

Systematic design of semi-autonomous agents involves specifying (i) the domain description: the actions the agent can do, its impact, the environment, etc.; (ii) the control execution of the agent; and (iii) directives for the agent. While there has been a lot of research on (i) and (ii), there has been relatively less work on (iii). This chapter made amends and explored the expressive power of existing temporal logic based goal specification languages. This chapter showed that in presence of actions with non-deterministic effects many interesting goals cannot be expressed using existing temporal logics such as LTL and CTL*. This chapter gave a formal proof of this, and showed showed that by introducing additional branching time operators A_{pol} and E_{pol} where the path is tied to the policy being executed users can express the goals that were thought inexpressible using temporal logic in [DLPT02]. A new language π-CTL* was proposed. This chapter then illustrated the necessity of having new quantifiers which are called "exists policy" and "for all policies" and developed the language P-CTL* which builds up on π-CTL* and has the above mentioned new quantifiers. The chapter further extended the goal specification languages π-CTL* and P-CTL* to π_σ-CTL* and P_σ-CTL* by adopting a different and more expressive policy structure. It turns out that new languages with such new policy definition exhibit different properties than π-CTL* and P-CTL*. Such a result reveals the importance of agent structure in goal specification languages. In

particular, in goal specification languages, to better suit the agent, one should take the agent architecture into account. This chapter showed how many of the goals that cannot be specified in earlier languages can be specified in the newly proposed languages.

An interesting aspect of this work is that it illustrates the difference between program specification and goal specification. Temporal logics were developed in the context of program specification, where the program statements are deterministic and there are no goals of the kind "trying one's best". In cognitive robotics, actions have non-deterministic effects and sometimes one keeps trying until one succeeds, and similar attempts to try one's best. The proposed language P-CTL* allows the specification of such goals. P-CTL* has the ability of letting the agent to compare and analyze policies and "adjust" its current goal accordingly.

An orthogonal expressiveness issue is related to the policy structure. This chapter focus on the policy structures as a mapping from states to actions, and as a mapping from histories to actions. A framework of formally comparing expressiveness of goal specification languages is proposed. The relations between π-CTL*, P-CTL*, π_σ-CTL*, and P_σ-CTL* are examined. The approach for comparing these languages can be easily extended to compare richer languages, such as the ones with policies that map LTL and CTL* formulas to actions.

In terms of future work, the connection of goal specification and planning is an interesting topic. This chapter illustrates that in more expressive goal specification languages, some strong requirement of the goal reduces the search space. One conjecture is that as the goal specification languages gets more and more expressive and specific, some categories of the planning problems with respect to such goals might become easier to solve.

Chapter 4

N-LTL AND ER-LTL: NON-MONOTONIC TEMPORAL LOGICS THAT FACILITATE ELABORATION-TOLERANT REVISION OF GOALS

In many domains such as in a human-robot interaction domain like a rescue and recovery situation, as the situation unveils physically or in the user's mind as time goes by, goals once specified may need to be further updated, revised, partially retracted, or even completely changed. Retract the earlier specification and give a completely new specification is undesirable as it costs precious time in terms of communication and formulation for the new specification, and may not even be appropriate, as the agent may have started acting based on the earlier specification. Ideas from the knowledge representation community are extrapolated, where non-monotonic knowledge representation languages are proposed for elaboration tolerant knowledge representation, and propose the development of non-monotonic temporal logics N-LTL and ER-LTL that rely on labeling sub-formulas and connecting multiple rules. The chapter also proposes the approach of progressing an ER-LTL program to take care of the case that the agent has started acting based on earlier specifications.

4.1 Introduction

This chapter summarizes and elaborates on the papers [BZ07] and [BZ08]. It starts with why it is important to have non-monotonic goals, and why non-monotonic requirements in goal specification languages are different from that in classical logics.

The previous chapter illustrates that an important component of autonomous agent design is goal specification. Often goals of agents are not just about or not necessarily about reaching one of a particular set of states, but also about satis-

fying certain conditions imposed on the trajectory. Besides, reactive agents with maintenance goals may not have a particular set of final states to reach. Also, agents acting in non-deterministic domains may lead to multiple trajectories instead of one. Temporal logics such as linear temporal logic LTL, branching time temporal logics CTL*, π-CTL*, and their extensions [BK98, NS00, BKT01] are invented. Thus the use of temporal logics and temporal connectives to specify goals has been suggested in the autonomous agent community and planning community [BKSD95, BK98, GV99, NS00, PT01]. In the decision theoretic planning community suggestions have been made to use temporal logics in specifying non-Markovian rewards [BBG96, BBG97, TGS$^+$06]. The previous chapter studies temporal logic extensions π-CTL* and P-CTL* to better capture properties of goals in non-deterministic domains.

However, in many domains such as in a human-robot interaction domain like a rescue and recovery situation, goals once specified may need to be further updated, revised, partially retracted, or even completely changed. This could be because at the time of initially specifying the goal, the user did not have complete information about the situation, or he was in haste and hence he did not completely think through the whole situation, and as the situation unveiled physically or in the user's mind, he had to change his specification. In other cases, it is not necessary to consider all possible cases in giving the initial goal. The following example illustrates these points.

Example 4. *John has an agent in his office that does errands for him. John may ask the agent to bring him some coffee. But soon he realizes that the coffee machine was broken. He is not sure if the machine has been fixed or not. He then revises his directive to the agent telling it that if the coffee machine is still broken then a cup of tea would be fine. Just after that he gets a call from a colleague who says that*

he had called a coffee machine company and asked them to deliver a new coffee machine. Then John calls up the agent and tells it that if the new coffee machine is already there then it should bring him coffee. (Note that the old coffee machine may still be broken.) He also remembers that he takes sugar with his tea and that the tea machine has various temperature settings. So he tells the agent that if it is going to bring tea then it should bring him a pack of sugar and set the tea machine setting to "very hot".

One may wonder why does not John in the above example give a well thought out directive at the start without making further changes after that. As mentioned earlier, some of it is because he lacked certain information, such as a new coffee machine having been ordered; in another case he had forgotten about the coffee machine being broken, and since he takes tea less often, he had also initially forgotten about the extra sugar.

In specifying goals of agents, often it is needed to specify goals non-monotonically. For example, initially, an agent may be given a goal of having p true through the trajectory while reaching s. Later, the agent may decide to weaken its goal so that in certain exceptional cases p does not have to be true. It is quite common that goals need to changed non-monotonically. In rescue and recovery situations with robots being directed by humans, there is often so much chaos together with the gradual trickling of information and misinformation that the human supervisors may have to revise their directives to the robots quite often.

Another motivation for having a non-monotonic goal specification language that allows easy updating through adding is that users may not want to give the agent a directive that is too specific, too complicated, and that takes into account all possible exceptions, from the very beginning. Besides users may not even know all the exceptions initially. A good non-monotonic goal specification language should

allow users to specify a simple goal initially and should allow users to refine it by adding new exceptions. All in elaboration tolerance manner.

To deal with the problem that the goals are unclear initially or need to be changed later on, one approach would be for the agent to replace its original goal by a revised goal, coming up with a completely new revised goal, or obtaining the revised goal by doing surgery on the original goal specification. However, that may cost precious time in terms of communication and formulation of the new specification, and may not be even appropriate, as the agent may already have started acting based on the earlier specification. Besides, this violates the principle of elaboration tolerance. These are limitations of existing temporal logics. What is needed is a goal specification language that allows users to update the goal specification by simply adding new statements to the original specification. Such a goal specification language would be non-monotonic.

This raises the question of choosing a goal specification language that can be revised or elaborated easily. As McCarthy says in [McC98], a natural language would be more appropriate. However, there is still a need of a formal language, sometimes as an intermediary between a natural language and the machine language and other times as a goal specification language. Considering the necessity and usefulness of temporal logics in specifying trajectories in standard planning and in specifying non-Markovian rewards in decision theoretic planning, to remain upward compatible with existing work in these directions, this work stays with the temporal connectives in temporal logics. The question then is: What kind of temporal logic will allow users easy revision of specifications?

In other aspects of knowledge representation, the use of non-monotonic logics for elaboration tolerant representation [McC98] is often advocated and for reasons similar to the example above: Intelligent entities need to reason and make decisions with incomplete information and in presence of additional information they should

be able to retract their earlier conclusions. Thus a non-monotonic temporal logic could be a good candidate for the purpose.

Looking back at the literature, although there have been many proposals for non-monotonic logics [McD82a], so far only two [FH91, Sae87] non-monotonic versions of temporal logics are found. The first extends auto-epistemic logic with temporal operators and does not explore issues such as elaboration tolerant representation of exceptions and weak exceptions. The second has semantics issues that are mentioned in the first.

This chapter proposes non-monotonic versions of temporal logics. The focus is on the overall aim of having non-monotonic goal languages. So rather than follows the path of non-monotonic modal logics and auto-epistemic logic this chapter focuses on specific aspects of knowledge representation that need non-monotonicity and borrows some specific techniques that allow such non-monotonicity.

One of the important use of non-monotonicity is the ability to express normative statements such as "normally q's have the property p." This resonates well with the need of non-monotonic goal languages as users may need to specify that "normally a state should satisfy the property p". Accompanying normative statements users have various kinds of exceptions. For example, consider the age old normative statement "birds normally fly". One kind of exception to such a statement is that "penguins are birds that do not fly". It is called a strong exception. Another kind of exception, referred to as weak exceptions, is that "injured birds are weak exceptions to the normative statement about birds flying;" as for wounded birds users do not know whether they fly or not. There is a need of similar exceptions with respect to goal specifications. A normative goal specification may specify that "normally a state should satisfy the property p". Strong exceptions may be states that satisfy some other conditions, while weak exceptions may be these conditions do not need to be satisfied.

To accommodate the above, proposed language N-LTL introduces two special notations[1]

- $[r]\phi$
- $[[r]]\phi$.

The intuitive meaning of the first one is that normally ϕ holds in a state and the label r lists the weak exceptions. The intuitive meaning of the second one is that normally ϕ holds in a state and the label r lists the strong exceptions. The role of r here is similar to the role of labeling defaults and normative statements when representing them in logic programming. There, often the label is used as a parameter with respect to the *ab* predicates.

This formulation is related to what we human beings communicate among ourselves. Users used to state something and then further refer to it using words such as "that" or "the". Users are referring to a sub-formula with a label in a similar way. Users use these labels also because users want to keep the temporal relations of sub-formulas that are specified in earlier formulas.

Since the non-monotonicity in goal languages is not due to having incomplete knowledge about the states, but rather due to the specifier not quite precisely knowing what she wants, N-LTL does not use operators such as the negation as failure operator '*not*' from logic programming. Here the issue is different from inferring or assuming negation by default.

On the other hand N-LTL borrows the idea behind program completion in logic programming to specify and interpret the conditions listed corresponding to the label r. Thus there may be a set of conditions written as

$$\langle r : \psi_1 \rangle \quad \ldots \quad \langle r : \psi_k \rangle$$

[1]Unlike traditional exceptions and weak exceptions, N-LTL wants the specifier to have predecided control over whether a particular goal fragment could have a weak exception or an exception, but not both.

that specify the exception or weak exception conditions with respect to r. Given the above, the overall condition associated with r becomes $\psi_1 \vee \ldots \vee \psi_k$. One is allowed to add additional conditions. For example, if $\langle r : \psi_{k+1} \rangle$ is added to the above set then the overall condition associated with r becomes $\psi_1 \vee \ldots \vee \psi_k \vee \psi_{k+1}$.

It is illustrated with respect to the following example.

Example 5. *Suppose initially the agent wants to maintain p true while reaching for s. The agent knows beforehand that the aim to maintain p is not strict; it is just that the agent does not know yet, under what conditions truth of p may not be necessary. After a while, the agent realizes that when q is true there is no need to have p true.*

The initial goal can be written in the language as $\langle g : (\Box[r]p) \wedge \Diamond s \rangle$. It says that the agent should maintain p while reaching s. If the exception r happens in some states, the agent may not need to maintain p in those states. The weak exception r is then specified as $\langle r : q \rangle$.

To informally illustrate how non-monotonicity is manifested in the above example, when a language is monotonic is defined.

Definition 27. *A logic L together with a query language Q and entailment relation $\models$ is monotonic if for all T, T' in L and t in Q, $T \models t$ implies $T \cup T' \models t$.* □

With respect to Example 5 let T be $\{\langle g : (\Box[r]p) \wedge \Diamond s \rangle\}$, $T' = \{\langle r : q \rangle\}$. Intuitively, $T \models \Box p$ and $T \cup T'$ is equivalent to $\Box(p \vee q) \wedge \Diamond s$ in LTL thus $T \cup T' \not\models \Box p$. Hence the proposed language N-LTL is non-monotonic.

The language N-LTL has many limitations such as it only allows strong exceptions and weak exceptions but does not allow arbitrary revising or retracting existing sub-formulas. Besides, when there is an exception in N-LTL, it must be predefined whether it is a weak exception or a strong exception. These limitations restrict the ability of N-LTL to specify goals in an evolving scenario.

This chapter continues on developing an appropriate non-monotonic temporal goal specification language that allows elaboration tolerant revision of goal specifications. The language ER-LTL is developed, which is also based on LTL [MP92]. Each ER-LTL program is composed of a set of rules of the form

$$\langle h : [r](f_1 \rightsquigarrow f_2)\rangle \tag{4.1}$$

The symbol h is referred to as the head of the rule and Rule 4.1 states that, normally, if formula f_1 is true, then the formula f_2 should be true, with exceptions given by rules with r in their heads and this rule is an exception to a formula labeled by h. ER-LTL also takes a similar approach as N-LTL and use Reiter's idea of a surface non-monotonic logic [Rei01] that gets compiled into a more tractable standard logic and thus avoid increase in complexity; The idea of completion is used when rules about exceptions are given for the same precondition. With simple rules as Rule 4.1, users are able to express various ways to revise goals. This includes specification of exceptions to exceptions, strengthening and weakening of preconditions, and revision and replacement of consequents.

This chapter is organized as follows: Section 4.2 proposes the syntax and semantics of a new language N-LTL. Section 4.3 proposes the syntax and semantics of language ER-LTL, and illustrate with examples on how ER-LTL can be used in specifying goals and revising them in an elaboration tolerant manner. As users may revise the goal of an agent after the agent has already executed part of his plan, the approach of progressing an ER-LTL program is discussed in Section 4.4. Applying the approach to other monotonic goal specification languages is briefly discussed in Section 4.5. Section 4.7 compares languages N-LTL and ER-LTL, discusses related works, and discusses properties of applying the techniques in ER-LTL to propositional logic. The chapter is concluded with a summary and future work.

4.2 N-LTL: A Non-monotonic Extension of LTL

This section extends LTL to capture non-monotonic requirements in specifying a goal. The new language is called N-LTL which stands for *non-monotonic LTL*. The syntax and semantics of N-LTL is first defined.

Syntax

While designing the language two questions need to be addressed:

- If syntactically the goal is one temporal formula, how can users revise it to have new goals by just adding to the original formula?
- How to refer to one part of a specification in another part of the specification?

For the first N-LTL borrows ideas from Reiter's approach to situation calculus [Rei01] where he compiles his specification to classical logic. While the classical logic part is monotonic, reasoning with respect to the specification language is non-monotonic and the non-monotonicity is achieved through the compilation process. For the second N-LTL borrows ideas from logic programming. Similar to a logic program consisting of a set of rules, each N-LTL program is a set of rules and rule labels are used such as r in Example 5 to link these rules into one temporal formula.

Definition 28. *Let $\{g\}$, R, and P be three disjoint sets of atoms. Let $\langle r\rangle$ be an atom in R, $\langle p\rangle$ be an atom in P, $e \in \{g\} \cup R$. $\langle f\rangle$ is a formula defined below:*

$$\begin{aligned}\langle f\rangle ::= \quad & \langle p\rangle | \langle f\rangle \wedge \langle f\rangle \mid \langle f\rangle \vee \langle f\rangle \mid \neg\langle f\rangle \mid \bigcirc \langle f\rangle \mid \Box\langle f\rangle \mid \\ & \Diamond\langle f\rangle \mid \langle f\rangle \mathsf{U} \langle f\rangle \mid [\langle r\rangle](\langle f\rangle) \mid [[\langle r\rangle]](\langle f\rangle)\end{aligned}$$

An N-LTL program is a set of rules $\langle e : f\rangle$, Where e is the head*, and f is the* body *of rule $\langle e : f\rangle$.* □

In an N-LTL program, g is a special symbol that stands for the final goal formula. R is the set of labels to be used to define corresponding exceptions and weak

exceptions. A formula f defines the conditions of atoms in R or the conditions of g. Intuitively, $[\langle r\rangle](\langle f\rangle)$ means that normally f is true, with the weak exceptions denoted through r. $[[\langle r\rangle]](\langle f\rangle)$ means that normally f is true, with the strong exceptions denoted through r. The weak and strong exception conditions corresponding to r are defined through other rules.

Definition 29 (Atom Dependency, Loop-free)**.** *Let $\langle e_1 : f_1\rangle$ be a rule in an N-LTL program. If $e_2 \in R$ occurs in the body of $\langle e_1 : f_1\rangle$, then e_2* depends *on e_1. The dependency relation is transitive. An N-LTL program is* loop-free *if no atom in R depends on itself in the program.* □

Semantics of N-LTL Programs

As mentioned earlier, the semantics of N-LTL programs is defined by following the approach taken in Reiter's situation calculus [Rei01]: N-LTL programs are compiled to LTL theories.

[Translate N-LTL program to LTL formula] A loop-free N-LTL program T is translated to an LTL formula $Tr(T)$ as follows:

1. Let $\langle e : f_1\rangle, \langle e : f_2\rangle, \cdots, \langle e : f_n\rangle$ be all the rules in T with e in the head, $e \in \{g\} \cup R$. A formula $f_1 \vee f_2 \vee \cdots \vee f_n$ is constructed, and it is called $E(e)$. Do this for any atom e if the set of rules with e in the head is not empty.
2. If atom a_1 depends on atom a_2, and $E(a_1)$ is defined, replace any occurrence of $[a_1](f)$ in $E(a_2)$ with $f \vee E(a_1)$. The revised formula is still called $E(a_2)$.
3. If atom a_1 depends on atom a_2, and $E(a_1)$ is defined, replace any occurrence of $[[a_1]](f)$ in $E(a_2)$ with $E(a_1)$. The revised formula is still called $E(a_2)$.
4. Do Step 2 and Step 3 recursively until no atoms e depending on g while $E(e)$ is not empty occurs in $E(g)$.
5. Finally, in $E(g)$, replace all remaining $[r](f)$ and $[[r]](f)$ with f. The revised goal formula is $Tr(T)$. □

This algorithm is illustrated with an example.

Example 6. *Consider N-LTL program T as follows:*

$$\langle g : (\Diamond[r_1](p)) \wedge [r_3](q)\rangle$$

$$\langle r_1 : [[r_2]](v)\rangle$$

$$\langle r_1 : \Box t\rangle$$

$$\langle r_2 : s\rangle$$

According to the definition, initially $E(g) = \Diamond[r_1](p) \wedge [r_3](q)$, $E(r_1) = [[r_2]](v) \vee \Box t$, and $E(r_2) = s$. By replacing the formulas according to the dependence relations, $E(g) = \Diamond(p \vee [[r_2]](v) \vee \Box t) \wedge [r_3](q) = \Diamond(p \vee s \vee \Box t) \wedge [r_3](q)$. There is no rules with r_3 as the head. Thus $E(g) = \Diamond(p \vee s \vee \Box t) \wedge q$. Further, $Tr(T) = \Diamond(p \vee s \vee \Box t) \wedge q$.

The program in Example 6 is loop-free. Loop-free N-LTL programs have the following property:

Proposition 8. *$Tr(T)$ is a well defined LTL formula for loop-free N-LTL program T.*

Proof. Define a formula as close-to-good if after replacing all sub-formula $[a_1](f)$ and $[[a_1]](f)$ to f in the formula, the formula is an LTL formula. According to the definition of N-LTL formulas in Definition 28, the body of each rule in an N-LTL program is a close-to-good formula.

Now check on the translation of the program in Algorithm 4.2. $E(e)$ for each e in the head in Step 1 is a close-to-good formula. In Step 2 and Step 3, after the replacement, each $E(a_2)$ is a close-to-good formula. Finally, Step 4 in Algorithm 4.2 removes all $[a_1](f)$ and $[[a_1]](f)$ to f. The resulted program is an LTL formula. □

Given this property, when a plan satisfies a goal specified in N-LTL can be defined.

Definition 30. *Let T be a loop-free N-LTL program. Given a state s, and a trajectory $\sigma = s_0, s_1, \cdots, s_k, \cdots$, $(s,\sigma) \models T$ in N-LTL if $(s,\sigma) \models Tr(T)$ with respect to LTL.* □

Definition 31 (Plans with respect to N-LTL goals)**.** *Let T be a loop-free N-LTL program. A sequence of actions $a_1, \cdots, a_n$ is a* plan *from the initial state s for the N-LTL goal T, if σ is the trajectory corresponding to s and $a_1, \cdots, a_n$, and $(s,\sigma) \models T$ in N-LTL.* □

Temporal logics such as LTL have a different property when an LTL goal formula is considered to be a set of temporal formulas. If P is a plan with respect to an LTL goal $T \cup T'$ then P is a plan with respect to T. In LTL, adding more formulas reduces (or at best leaves it unchanged) the set of plans satisfying it while in N-LTL, this is not the case.

Properties and N-LTL in Goal Specification

Now, the notion that N-LTL is non-monotonic is defined. LTL is monotonic since $T \models t$ implies $T \cup T' \models t$ where T and T' are two sets of LTL temporal formulas and t is an LTL temporal formula. The following entailment in N-LTL is considered.

Definition 32 (Entailment)**.** *$T \models T'$ if $Tr(T) \models Tr(T')$, where T and T' are two loop-free N-LTL programs.* □

Proposition 9. *The entailment in Definition 32 is non-monotonic.*

Proof. To prove that the entailment is non-monotonic, now find a program T, T_2, and T' such that $T \models T'$ but $T \cup T_2 \not\models T'$, where T, T' and $T \cup T_2$ are all loop-free N-LTL programs.

For example, let T and T' be $\langle g : [r_1](\Box p)\rangle$. It is clear that $T \models T'$ as $Tr(T) = Tr(T') = \Box p$. Let T_2 be $\langle r_1 : [r_2](\Box q)\rangle$. $Tr(T \cup T_2) = \Box q$. It is not the case that $Tr(T \cup T_2) \models Tr(T')$ in LTL. Thus the entailment is non-monotonic. □

Consider one example to illustrate the way of using N-LTL.

Example 7. *One professor asks his robot to make a photocopy of one document and fetch a cup of coffee. However, before the robot goes out the office, the professor finds out that the coffee is sold out. No plan can satisfy the goal given to the robot. The professor would now like to weaken the goal. Following are three possibilities:*

1. *He would be happy with a cup of tea instead;*
2. *He just needs the copy of the document and is willing to forget about the coffee;*
3. *The robot may come back to his office with the document copied and go about looking for the coffee later.*

If the professor was using N-LTL and from past experience knows that he may have to revise his goal, especially with respect to coffee, he can express the initial goal as:

$$\langle g : \Diamond((\lbrack r\rbrack coffee) \wedge copy \wedge \Diamond office)\rangle$$

It is equivalent to satisfying $\Diamond(coffee \wedge copy \wedge \Diamond office)$ *in LTL.*

Later on, according to the new conditions and new alternatives, the professor may revise his original goal by adding one of the following three rules:

1. *Adding the new rule* $\langle r : tea\rangle$ *which makes the overall goal equivalent to* $\Diamond((coffee \vee tea) \wedge copy \wedge \Diamond office)$ *in LTL;*
2. *Let* $\top$ *denote true and* $\bot$ *denote false. Adding the new rule* $\langle r : \top\rangle$ *which makes the overall goal equivalent to* $\Diamond((coffee \vee \top) \wedge copy \wedge \Diamond office)$ *in LTL, which is equivalent to* $\Diamond(copy \wedge \Diamond office)$ *in LTL;*

3. *Adding the new rule* $\langle r : \Diamond(coffee \wedge \Diamond office)\rangle$ *which makes the overall goal equivalent to* $\Diamond(coffee \wedge copy \wedge \Diamond office) \vee \Diamond(copy \wedge \Diamond office \wedge \Diamond(coffee \wedge \Diamond office))$ *in LTL. Now it allows the agent to get the document copied first before waiting for the coffee.*

If users want to get the tea instead of coffee, users should know initially that the sub-task of getting coffee can be replaced by a different goal. Now, the initial formulation is

$$\langle g : \Diamond(([[r]]coffee) \wedge copy \wedge \Diamond office)\rangle$$

When users add a new goal $\langle r : tea\rangle$*. The goal afterwards is equivalent to* $\Diamond(tea \wedge copy \wedge \Diamond office)$ *in LTL.*

Note that "copy" must be satisfied in this domain as the professor does not want to weaken this condition. While the robot is on its way of getting things done, some other exceptions may happen. With N-LTL, users may further refine part of the goal.

Note that if a rule in the formula has a sub-string $[[r_1]](f)$, then removing rules $\langle r_1 : \bot\rangle$ may affect the semantics of the formula. If there is no sub-string $[[r_1]](f)$ in any part of the formula, then users can remove rules $\langle r_1 : \bot\rangle$ without affecting the semantics of the formula. Here, f stands for any temporal formula.

4.3 ER-LTL

N-LTL has some limitations. Firstly, it needs to be pre-specified whether an exception is a strong exception or a weak exception. Secondly, the way of dealing with rules is limited. In some cases, it may end up having a lot of changes to the initial program. The non-monotonic temporal logic ER-LTL that is based on LTL is presented; ER stands for "*E*xceptions and *R*evisions". It takes care of some limitations

of N-LTL. Besides, it is more powerful in expressing exceptions and revisions. The syntax and semantics of the language is defined firstly.

Syntax

Definition 33 (ER-LTL program)**.** *Let G, R, and P be three disjoint sets of* atoms. *Let g be the only atom in G. Let* $\langle r \rangle$ *be an atom in R,* $\langle p \rangle$ *be an atom in P. An* ER-LTL formula $\langle f \rangle$ *is defined recursively as:*

$$\begin{aligned} \langle f \rangle \quad ::= \quad & \langle p \rangle \mid (\langle f \rangle \wedge \langle f \rangle) \mid (\langle f \rangle \vee \langle f \rangle) \mid \neg \langle f \rangle \mid \bigcirc \langle f \rangle \mid \\ & \Box \langle f \rangle \mid \Diamond \langle f \rangle \mid (\langle f \rangle \mathsf{U} \langle f \rangle) \mid [\langle r \rangle](\langle f \rangle \rightsquigarrow \langle f \rangle) \end{aligned}$$

An ER-LTL rule *is of the form* $\langle h : [r](f_1 \rightsquigarrow f_2) \rangle$*, where* $h \in G \cup R$*,* $r \in R$*, and* f_1 *and* f_2 *are two ER-LTL formulas; h is referred to as the* head*, and* $[r](f_1 \rightsquigarrow f_2)$ *as the* body *of the rule. An* ER-LTL program *is a finite set of ER-LTL rules.* □

The symbols $\top$ and $\bot$ are abbreviations for propositional formulas that evaluate to *true* and *false* respectively. For example, for atom $q \in P$, $q \vee \neg q$ is abbreviated as $\top$, and $q \wedge \neg q$ is abbreviated as $\bot$.

The same as in N-LTL, rules in ER-LTL with head g express the initial goal which may later be refined.

In comparison to LTL, $[r](f_1 \rightsquigarrow f_2)$ is the only new constructor in ER-LTL. f_1 is referred to as the precondition and f_2 as the consequent of this formula. It states that normally if the precondition f_1 is true, then the consequent f_2 needs to be satisfied, with the exceptions specified via r. The conditions denoting the exceptions labeled by r are defined using other rules. When those exceptions are presented in the program, other goals instead of f_2 need to be satisfied. If the sub-formula is preceded with a head atom $h \in R \cup G$ as in $\langle h : [r](f_1 \rightsquigarrow f_2) \rangle$, it further states that this sub-formula is an exception to formulas labeled by h.

Similar to N-LTL, several auxiliary definitions that will be used in defining the semantics of ER-LTL are defined.

Definition 34 (Atom dependency)**.** *Let T be an ER-LTL program. Let h_1, h_2 be atoms in $R \cup G$. Atom h_1* depends *on h_2 in T if there is a rule in T such that h_2 occurs in the body of the rule while h_1 is the head of the rule. The dependency relation is transitive.* □

Example 8. *Consider the following rules:*

$$\langle r_1 : [r_2](p \rightsquigarrow ((\Box q) \rightsquigarrow r))\rangle \tag{4.2}$$

$$\langle r_1 : [r_2]((\Diamond p \vee [r_3](\Box q \rightsquigarrow (p \mathsf{U} q))) \rightsquigarrow (\Diamond q))\rangle \tag{4.3}$$

Rule 4.2 is not a syntactically valid ER-LTL rule. $((\Box q) \rightsquigarrow r)$ in it is not a valid ER-LTL formula. It should be preceded by a label in R. Rule 4.3 is a valid ER-LTL rule. With respect to a program consisting of Rule 4.3, r_1 depends on r_2 and r_3.

Definition 35 (Loop-free, Leaf)**.** *An ER-LTL program is* loop-free *if in the program, no atom in R depends on itself. An atom is called a* leaf *in the program if it does not depend on any atom in R.* □

Semantics

Now define a translation from ER-LTL to LTL so as to relate the semantics of ER-LTL to the semantics of LTL. A similar technique as in N-LTL is used to capture temporal relations among different rules to combine them to be one temporal formula. Atom r_1 depends on r_2 states that r_2 should be fully expanded before r_1.

[Translate ER-LTL program to LTL formula] A finite loop-free ER-LTL program T is translated to an LTL formula $Tr(T)$ as follows:

1. For each sub-formula in T of the form $[r_t](l_{t0} \rightsquigarrow f_{t0})$ where for all rules with r_t in the head:

$$\langle r_t : [r_{t1}](l_{t1} \rightsquigarrow f_{t1})\rangle$$
$$\ldots$$
$$\langle r_t : [r_{tk}](l_{tk} \rightsquigarrow f_{tk})\rangle$$

   r_{ti} $(1 \leq i \leq k)$ are leaf atoms. l_{ti}, f_{ti} $(0 \leq i \leq k)$ are LTL formulas.

   a) If $[r_t](l_{t0} \rightsquigarrow f_{t0})$ is not preceded with ":", the formula $[r_t](l_{t0} \rightsquigarrow f_{t0})$ is replaced with $(l_{t0} \wedge \neg l_{t1} \wedge \cdots \wedge \neg l_{tk} \Rightarrow f_{t0}) \wedge (l_{t0} \wedge l_{t1} \Rightarrow f_{t1}) \wedge \cdots \wedge (l_{t0} \wedge l_{tk} \Rightarrow f_{tk})$. The program is still called T;

   b) If $[r_t](l_{t0} \rightsquigarrow f_{t0})$ is preceded with ":" and it is in a rule of the form $\langle r_v : [r_t](l_{t0} \rightsquigarrow f_{t0})\rangle$ where $r_v \in G \cup R$, the rule is replaced with:

$$\langle r_v : [r_t](l_{t0} \wedge \neg l_{t1} \wedge \cdots \wedge \neg l_{tk} \rightsquigarrow f_{t0})\rangle$$
$$\langle r_v : [r_t](l_{t0} \wedge l_{t1} \rightsquigarrow f_{t1})\rangle$$
$$\ldots$$
$$\langle r_v : [r_t](l_{t0} \wedge l_{tk} \rightsquigarrow f_{tk})\rangle$$

   The resulting program is still called T.

2. Repeat Step 1 until it can no longer be applied further.

3. Suppose $\langle g : [r_i](l_i \rightsquigarrow f_i)\rangle$ $(0 \leq i \leq n)$ are all rules with the head g. $Tr(T)$ is defined as $\wedge_{i=0}^{n}(l_i \Rightarrow f_i)$. □

Example 9. *An ER-LTL program T is given as follows*[2]*:*

$$\langle g : [r_1](\mathit{bird} \rightsquigarrow \mathit{fly})\rangle$$

$$\langle r_1 : [r_2](\mathit{penguin} \rightsquigarrow \neg \mathit{fly})\rangle$$

$$\langle r_1 : [r_3](\mathit{wounded} \rightsquigarrow \top)\rangle$$

$$\langle r_2 : [r_4](\mathit{flyingPenguin} \rightsquigarrow \mathit{fly})\rangle$$

[2]Here and in a later example the flying bird example that has been used a lot in the non-monotonic reasoning literature is used. This is only for quick illustration purposes, and not to suggest that ER-LTL is an alternative to traditional non-monotonic languages. There has been significant progress in the research on non-monotonic reasoning. ER-LTL is not one alternative of these logics. The claim is only with respect to non-monotonic temporal logics, which have not been explored much.

After the first processing of step 1 of Algorithm 4.3, the output is the set of rules:

$$\langle g : [r_1](\mathit{bird} \rightsquigarrow \mathit{fly})\rangle$$

$$\langle r_1 : [r_2](\mathit{penguin} \wedge \neg \mathit{flyingPenguin} \rightsquigarrow \neg \mathit{fly})\rangle$$

$$\langle r_1 : [r_3](\mathit{wounded} \rightsquigarrow \top)\rangle$$

$$\langle r_1 : [r_2](\mathit{penguin} \wedge \mathit{flyingPenguin} \rightsquigarrow \mathit{fly})\rangle$$

After the second processing of step 1, the set of rules is:

$$\langle g : [r_1](\mathit{bird} \wedge \neg \mathit{penguin} \wedge \neg \mathit{wounded} \rightsquigarrow \mathit{fly})\rangle$$

$$\langle g : [r_1](\mathit{bird} \wedge \mathit{penguin} \wedge \neg \mathit{flyingPenguin} \rightsquigarrow \neg \mathit{fly})\rangle$$

$$\langle g : [r_1](\mathit{bird} \wedge \mathit{wounded} \rightsquigarrow \top)\rangle$$

$$\langle g : [r_1](\mathit{bird} \wedge \mathit{penguin} \wedge \mathit{flyingPenguin} \rightsquigarrow \mathit{fly})\rangle$$

Finally, based on step 3, $Tr(T)$ is the output. It can be simplified to: $(\mathit{bird} \wedge \neg \mathit{penguin} \wedge \neg \mathit{wounded} \Rightarrow \mathit{fly}) \wedge (\mathit{bird} \wedge \mathit{penguin} \wedge \neg \mathit{flyingPenguin} \Rightarrow \neg \mathit{fly}) \wedge (\mathit{bird} \wedge \mathit{penguin} \wedge \mathit{flyingPenguin} \Rightarrow \mathit{fly})$.

ER-LTL in Goal Specification

Loop-free ER-LTL programs have the following property.

Proposition 10. *Given a loop-free ER-LTL program T, $Tr(T)$ is an LTL formula.*

Proof. Define a formula as close-to-good if after replacing all sub-formula $[r](l \rightsquigarrow f)$ with $(l \Rightarrow f)$, the resulted is an LTL formula. According to the definition of ER-LTL formulas in Definition 33, the body of each rule in an ER-LTL program is a close-to-good formula.

Now check on the translation of the program in Definition 4.3. After each processing in Step 1 in the definition, each formula in the program is a close-to-good. Thus Step 3 makes the final program an LTL formula. □

Given this property, when a plan satisfies an ER-LTL program can be defined.

Definition 36. *Let T be a loop-free ER-LTL program, $\sigma = s_0, s_1, \cdots, s_k, \cdots$ be a trajectory, and i be an index of σ. $(i,\sigma) \models T$ in ER-LTL if $(i,\sigma) \models Tr(T)$ in LTL.* □

An ER-LTL program T is equivalent to an LTL formula T' if $Tr(T)$ and T' are equivalent in LTL. For any LTL formula G, there is an equivalent ER-LTL program.

When planning with an ER-LTL goal T, find plans for LTL formula $Tr(T)$. When T is updated to $T \cup T'$, users need to find plans for LTL formula $Tr(T \cup T')$.

Definition 37 (Entailment)**.** *Given that T_1 and T_2 are loop-free ER-LTL programs, $T_1 \models T_2$ if $Tr(T_1) \models Tr(T_2)$ in LTL.* □

Proposition 11. *The entailment in Definition 37 is non-monotonic.*

Proof. Consider the following two ER-LTL rules:

$$\langle g : [r_1](\top \rightsquigarrow \Box p)\rangle \tag{4.4}$$

$$\langle r_1 : [r_2](\Diamond q \rightsquigarrow \Diamond q)\rangle \tag{4.5}$$

Let T_1 be a program consisting of Rule 4.4, and T_2 be a program consists of Rule 4.4 and Rule 4.5. $Tr(T_1) = \Box p$ while $Tr(T_2) = \Diamond q \vee \Box p$. It is easy to see that $T_1 \models T_1$ while $T_2 = T_1 \cup \{Rule\ 4.5\}$ and $T_2 \not\models T_1$. Thus the entailment relation defined in Definition 37 in ER-LTL is non-monotonic. □

This implies that a plan possibly satisfy an ER-LTL program $T_1 \cup T_2$ but not T_1. It should be noted that the opposite is also true.

Exceptions and Revisions in ER-LTL

Now illustrate the application of ER-LTL in modeling exceptions and revisions. This sub-section starts with the modeling of exceptions that happen mainly because the user has incomplete information about the domain, the domain has been

changed after the initial goal is given, or the user does not have a clear specification for the agent initially.

Exceptions

First consider modeling weak exceptions and strong exceptions in goal specification.

Weak Exception and Strong Exception As discussed earlier, strong exceptions are to refute the default conclusion when exceptions happen; Weak exceptions are to render the default inapplicable. In terms of goal specification, suppose $f_1 \wedge f_2$ is the initial goal users have, after having the weak exception on f_1, users do not know whether sub-goal f_1 should be true or not, users thus can remove the sub-formula f_1 from the existing specification. On the other hand, if users have a strong exception on f_1, users should conclude that f_1 is no longer true, and cannot be true. Thus, users need to have $\neg f_1$ as a part of the revised goal specification. Consider the following example, again, for simplicity, given with respect to the birds flying scenario.

Example 10. *Birds normally fly. Penguins are birds that do not fly. Users do not know whether wounded birds fly or not.*

The initial statement can be written as

$$\langle g : [r_1](bird \rightsquigarrow fly)\rangle \tag{4.6}$$

It is equivalent to the LTL formula bird $\Rightarrow$ *fly. If append rule*

$$\langle r_1 : [r_2](penguin \rightsquigarrow \neg fly)\rangle \tag{4.7}$$

to it, the program is equivalent to the LTL formula $((bird \wedge \neg penguin) \Rightarrow fly) \wedge ((bird \wedge penguin) \Rightarrow \neg fly)$. *If append rule*

$$\langle r_1 : [r_3](wounded \rightsquigarrow \top)\rangle \tag{4.8}$$

about wounded birds to Rule 4.6, output is a program that is equivalent to the LTL formula $((bird \wedge \neg wounded) \Rightarrow fly)$.

This example shows that when users need a strong exception, users can specify the negation of the initial consequents explicitly as in Rule 4.7. When users need a weak exception, users can simply say as in Rule 4.8 that under the exception, no consequents are needed.

Exception to Exception The way of dealing with exceptions to exception in ER-LTL is illustrated by the following example.

Example 11. *Birds normally fly. Penguins are birds that do not fly. However, a flying penguin is a penguin that can fly.*

The initial statement is written as Rule 4.6. Later, Rule 4.7 and a rule

$$\langle r_2 : [r_4](flyingPenguin \rightsquigarrow fly)\rangle \tag{4.9}$$

are appended. Rule 4.9 is an exception to the exception stated in Rule 4.7. The program consisting of the three rules is equivalent to the LTL formula $(bird \wedge (\neg penguin \vee flyingPenguin) \Rightarrow fly) \wedge (bird \wedge penguin \wedge \neg flyingPenguin \Rightarrow \neg fly)$.

Revision: Change User Intentions

Various revisions of the goal are allowed if it is represented in ER-LTL. ER-LTL splits the requirements to preconditions and consequents such that users may have goals as "if some conditions are satisfied, the agent should satisfy some goals". A few approaches of revising preconditions and consequents are listed now. They help to revise any part of the initial ER-LTL goal. In the following, a simple example is considered where the initial ER-LTL program is:

$$\langle g : [r_1](f_1 \rightsquigarrow f_2)\rangle \tag{4.10}$$

where f_1 and f_2 are two LTL formulas.

Changing Consequents Ways to change consequents in the goal are considered first.

Example 12. *Given Rule 4.10, if the ER-LTL rule*

$$\langle r_1 : [r_2](f_1 \rightsquigarrow f_3)\rangle,$$

is appended where f_3 *is an LTL formula, the revised program is equivalent to the LTL formula* $((f_1 \wedge \neg f_1) \Rightarrow f_2) \wedge ((f_1 \wedge f_1) \Rightarrow f_3)$*, or* $f_1 \Rightarrow f_3$*. Now the consequent has changed from* f_2 *to* f_3*.*

The consequent can be changed to be stronger or weaker than the initial specification. It can also be revised to one that is different from the initial specification. Similar revisions can be made for preconditions as well.

Changing Preconditions Now list a few examples illustrating how to change preconditions in a goal specification.

Example 13 (Making Preconditions Stronger)**.** *Suppose users want to refine the goal given as Rule 4.10 by having a new precondition* f_3 *together with* f_1*. The program can be refined by appending the rule:*

$$\langle r_1 : [r_2](\neg f_3 \rightsquigarrow \top)\rangle. \tag{4.11}$$

The new formula states that if $\neg f_3$ *is satisfied, then the goal is satisfied naturally. The refined program is equivalent to the LTL formula* $(f_1 \wedge f_3) \Rightarrow f_2$*.*

Example 14 (Making Preconditions Weaker)**.** *Suppose users want to refine the goal given as Rule 4.10 so that under a new condition* f_3*, consequent* f_2 *also need to be satisfied. This refinement will weaken the precondition* f_1*. The program can be refined by appending the rule:*

$$\langle g : [r_1](f_3 \rightsquigarrow f_2)\rangle.$$

The new program is equivalent to the LTL formula $(f_1 \vee f_3) \Rightarrow f_2$.

Example 15 (Changing Preconditions)**.** *Suppose users want to refine the goal given as Rule 4.10 so as to change the precondition* f_1 *to* f_3*. This can be done by appending the following rules*

$$\langle r_1 : [r_2](f_1 \rightsquigarrow \top)\rangle$$

$$\langle g : [r_3](f_3 \rightsquigarrow f_2)\rangle$$

to the program consisting of Rule 4.10. The new program is equivalent to the LTL formula $((f_1 \wedge \neg f_1) \Rightarrow f_2) \wedge (f_1 \wedge f_1 \Rightarrow \top) \wedge (f_3 \Rightarrow f_2)$*, which can be simplified as* $f_3 \Rightarrow f_2$.

Revision after Revision Now consider an example that needs further revision after the first revision.

Example 16. *In Example 12, the revised program is equivalent to the LTL formula* $f_1 \Rightarrow f_3$*. If users want to further revise the consequent to* f_4*, and make the program equivalent to* $f_1 \Rightarrow f_4$*, the rule* $\langle r_2 : [r_3](f_1 \rightsquigarrow f_4)\rangle$ *can be added to the existing program.*

Nested Revision Nested revisions are also common when users introduce a new goal to the domain while not clear about the preconditions and consequents of the new goal. Rules that specify that the preconditions and consequents will be given later. We illustrate this by the following example.

Example 17. *Suppose the initial ER-LTL program is*

$$\langle g : [r_1](\top \rightsquigarrow f_1)\rangle.$$

Suppose now we know in addition to f_1 *some thing more needs to be done; but we do not yet know what. We can append the following rule to accommodate that*

possibility:

$$\langle r_1 : [r_2](\top \rightsquigarrow f_1 \wedge [r_3](\top \rightsquigarrow \top))\rangle,$$

It will allow users to add additional requirements later.

Allow Sub-formula to be Modified

The agent given the initial goal may not know which sub-goal or which part of the formula might be revised further. As seen in ER-LTL, in order to make revisions or define exceptions for a sub-formula, the sub-formula need to be declared as modifiable firstly. Thus a way of declaring a sub-formula to be modifiable is needed. The following example illustrates this:

Example 18. *The initial goal is given as*

$$\langle g : [r_1](\top \rightsquigarrow f_1 \wedge f_2)\rangle$$

As the user may aware that f_2 might be further modified without affecting f_1. The following rule can be added to capture such a motivation.

$$\langle r_1 : [r_2](\top \rightsquigarrow f_2 \wedge [r_3](\top \rightsquigarrow f_2))\rangle$$

The modified and the original program are equivalent to the same LTL formula $f_1 \wedge f_2$.

Thus ER-LTL enables users in revising goals of an agent in an elaboration tolerant manner. The following sub-section elaborates on how the evolution of John's requirement introduced in the Introduction section can be represented.

Representing John's Requirements in ER-LTL

Now show the way of applying ER-LTL in representing the problem in Example 4.

Example 19. *John can specify his initial goal in ER-LTL as:*

$$\langle g : [r_0](\top \rightsquigarrow \Diamond(\mathit{coffee} \wedge \Diamond \mathit{back}))\rangle. \tag{4.12}$$

It is equivalent to the LTL formula $\Diamond(coffee \wedge \Diamond back)$. *It states that the agent needs to get a cup of coffee and then come back.*

After realizing that the coffee machine might be broken, John can refine his goal by adding the following two rules:

$$\langle r_0 : [r_1](\top \rightsquigarrow \Diamond([r_2](\top \rightsquigarrow coffee) \wedge \Diamond back))\rangle \tag{4.13}$$

$$\langle r_2 : [r_3](broken \rightsquigarrow tea)\rangle \tag{4.14}$$

Rule 4.13 now allows the sub-formula about coffee in the initial goal to be further refined. The overall specification is now equivalent to the LTL formula $\Diamond((\neg broken \Rightarrow coffee) \wedge (broken \Rightarrow tea) \wedge \Diamond back)$. *Notice that John did not have to retract his previous goal and give a new goal; neither did he have to change the earlier specification; he just had to add to his previous specification and the semantics of the language takes care of the needed change. This is an example of "elaboration tolerance" of a language.*

Later, after knowing from a colleague that a new coffee machine might be installed, John can give the agent a new command by adding one more rule to the existing goal:

$$\langle r_3 : [r_4](newMachine \rightsquigarrow coffee)\rangle$$

The overall goal is now equivalent to the LTL formula $\Diamond((broken \wedge \neg newMachine \Rightarrow tea) \wedge (\neg(broken \wedge \neg newMachine) \Rightarrow coffee) \wedge \Diamond back)$.

Finally, John can give the agent a new command by adding the following rule.

$$\langle r_3 : [r_5](\neg newMachine \rightsquigarrow (hot \wedge \bigcirc(tea \wedge sugar)))\rangle$$

The overall goal is now equivalent to the LTL formula: $\Diamond((broken \wedge \neg newMachine \Rightarrow (hot \wedge \bigcirc(tea \wedge sugar)) \wedge (\neg(broken \wedge \neg newMachine) \Rightarrow coffee)) \wedge \Diamond back)$.

Note that in this example, the way of expanding and revising the goal in an elaboration tolerance manner is introduced by introducing a different consequent,

weakening the requirements, introducing exceptions to exceptions, and introducing nested exceptions.

4.4 Progressing ER-LTL

When represent a goal in ER-LTL, it is interesting to study how the goal can be simplified, and how the goal can be progressed based on earlier states in the trajectory. In order to do so, how a new rule added to a program affect the models of existing program, and under what condition two programs are "strong-equivalent" are presented below. These definitions are helpful when there is a need to simplify a progressed ER-LTL program.

Strengthening and Weakening in ER-LTL

This section considers how the new rules added to a program affect the existing ER-LTL program.

With the introduction of preconditions and consequents, ER-LTL branches on preconditions. Given a loop-free ER-LTL program, adding a new rule with head g correspond to adding a new branch. Adding a new rule with head $r \in R$ correspond to adding a new branch, or revising existing branches. Branches are added and removed based on the new rules added. For example, given an ER-LTL rule of the form:

$$\langle r_1 : [r_2](f_1 \rightsquigarrow f_2)\rangle$$

Another rule with head r_2 of the form

$$\langle r_2 : [r_3](f_3 \rightsquigarrow f_4)\rangle$$

introduces a new branch if $f_3 \not\models f_1$. Otherwise, the existing branch on f_2 is removed and the new branch on f_4 is added.

With different rules added, a goal can be made easier or more difficult to satisfy:

Definition 38. *Given two ER-LTL programs T_1 and T_2, if $T_1 \cup T_2 \models T_1$, T_2 is called a strengthener of T_1; if $T_1 \models T_1 \cup T_2$, T_2 is called a weakener of T_1.*

After union with a weakener of a program, the new ER-LTL program is satisfied by more or equal number of policies. After union with a strengthener of a program, the new ER-LTL program is satisfied by fewer or equal number of policies.

Note that if there is a similar definition as Definition 38 for LTL, any LTL formula is a strengthener of any other LTL formula. Adding rules to a monotonic temporal logic always strengthening the logic. This also explains that LTL is monotonic while ER-LTL is non-monotonic.

However, in non-monotonic logic ER-LTL, some programs are always strengthener to other programs.

Proposition 12. *A rule of the form*

$$\langle g : [r](f_1 \rightsquigarrow f_2)\rangle \tag{4.15}$$

is a strengthener of any loop-free ER-LTL program, where $g \in G$, $r \in R$, and f_1 and f_2 are well defined ER-LTL formulas.

Proof. Let Π be a loop-free ER-LTL program and its corresponding LTL formula is $Tr(\Pi)$. Let a rule of the form 4.15 be t. According to the translation in Algorithm 4.3, before Step 3, the corresponding program of $\Pi \cup \{t\}$ will have one more rule than the program of Π. The rule is of the form $\langle g : [r](f_3 \rightsquigarrow f_4)\rangle$ where f_3 and f_4 are LTL formulas that correspond to f_1 and f_2 accordingly. Finally, $Tr(\Pi \cup \{t\})$ is of the form $Tr(\Pi) \wedge (f_3 \Rightarrow f_4)$. As $Tr(\Pi) \wedge (f_3 \Rightarrow f_4) \models Tr(\Pi)$, Rule 4.15 is a strengthener of loop-free ER-LTL program Π. □

Strong Equivalence in ER-LTL

Now check the programs that have the same consequents in weakening or strengthening other programs. The notion of strong equivalence is defined similar to that

in logic program [LPV01] that helps to simplify the program without affecting the rest of the program.

Definition 39 (strong equivalence)**.** *Two ER-LTL programs T_1 and T_2 are strongly equivalent if $T_2 \cup T$ and $T_1 \cup T$ are loop-free programs, and $T_1 \cup T \models T_2 \cup T$ and $T_2 \cup T \models T_1 \cup T$ for any ER-LTL program T.* □

It is easy to know that if two ER-LTL programs are strongly equivalent, then a policy in a domain satisfies the goal denoted by one program if and only if it satisfies the other one. Now list some ER-LTL programs that are strongly equivalent:

Proposition 13. *Let $r \in R$, $h \in R \cup G$, $g \in G$, and f_1, f_2, and f_3 be arbitrary ER-LTL formulas.*

$$\Pi_1: \quad \langle h : [r](f_1 \rightsquigarrow f_2)\rangle$$
$$\langle h : [r](f_3 \rightsquigarrow f_2)\rangle$$

and

$$\Pi_2: \quad \langle h : [r]((f_1 \vee f_3) \rightsquigarrow f_2)\rangle$$

are strongly equivalent.

Proof. Firstly, it is safe to assume that f_1, f_2, and f_3 are LTL formulas. According to the semantics of ER-LTL, occurrences of the same ER-LTL formula will be translated to the same LTL formula.

Given a program Π, now prove that $\Pi \cup \Pi_1$ and $\Pi \cup \Pi_2$ are equivalent. For each occurrence of r in program Π, suppose the set of rules with r in head are:

$$\langle r : [r_1](f_{i1} \rightsquigarrow f_{j1})\rangle$$
$$\ldots$$
$$\langle r : [r_n](f_{in} \rightsquigarrow f_{jn})\rangle$$

where $r_1, \cdots, r_n$ do not occur in the head of other rules. According to the semantics of ER-LTL, in $\Pi \cup \Pi_1$, the set of rules will be replaced by

$$\langle h : [r]((f_1 \wedge \neg f_{i1} \wedge \cdots \wedge \neg f_{in}) \rightsquigarrow f_2)\rangle$$
$$\langle h : [r]((f_1 \wedge f_{i1}) \rightsquigarrow f_{j1})\rangle$$
$$\cdots$$
$$\langle h : [r]((f_1 \wedge f_{in}) \rightsquigarrow f_{jn})\rangle$$
$$\langle h : [r]((f_3 \wedge \neg f_{i1} \wedge \cdots \wedge \neg f_{in}) \rightsquigarrow f_2)\rangle$$
$$\langle h : [r]((f_3 \wedge f_{i1}) \rightsquigarrow f_{j1})\rangle$$
$$\cdots$$
$$\langle h : [r]((f_3 \wedge f_{in}) \rightsquigarrow f_{jn})\rangle$$

In $\Pi \cup \Pi_2$, the set of rules will be replaced by

$$\langle h : [r](((f_1 \vee f_3) \wedge \neg f_{i1} \wedge \cdots \wedge \neg f_{in}) \rightsquigarrow f_2)\rangle$$
$$\langle h : [r](((f_1 \vee f_3) \wedge f_{i1}) \rightsquigarrow f_{j1})\rangle$$
$$\cdots$$
$$\langle h : [r](((f_1 \vee f_3) \wedge f_{in}) \rightsquigarrow f_{jn})\rangle$$

For each occurrence of h in the body of a rule that is preceded by ":" of the form

$$\langle t : [h](f_{h1} \rightsquigarrow f_{h2})\rangle,$$

the corresponding rule in program $\Pi \cup \Pi_1$ is replaced by

$$\langle t : [h]((f_{h1} \wedge \neg(f_1 \wedge \neg f_{i1} \wedge \cdots \wedge \neg f_{in}) \wedge \neg(f_1 \wedge f_{i1}) \wedge \cdots \wedge \neg(f_1 \wedge f_{in}) \wedge$$

$$\neg(f_3 \wedge \neg f_{i1} \wedge \cdots \wedge \neg f_{in}) \wedge \neg(f_3 \wedge f_{i1}) \wedge \cdots \wedge \neg(f_3 \wedge f_{in})) \rightsquigarrow f_{h2})\rangle$$

$$\langle t : [h]((f_{h1} \wedge f_1 \wedge \neg f_{i1} \wedge \cdots \wedge \neg f_{in}) \rightsquigarrow f_2)\rangle$$

$$\langle t : [h]((f_{h1} \wedge f_1 \wedge f_{i1}) \rightsquigarrow f_{j1})\rangle$$

$$\cdots$$

$$\langle t : [h]((f_{h1} \wedge f_1 \wedge f_{in}) \rightsquigarrow f_{jn})\rangle$$

$$\langle t : [h]((f_{h1} \wedge f_3 \wedge \neg f_{i1} \wedge \cdots \wedge \neg f_{in}) \rightsquigarrow f_2)\rangle$$

$$\langle t : [h]((f_{h1} \wedge f_3 \wedge f_{i1}) \rightsquigarrow f_{j1})\rangle$$

$$\cdots$$

$$\langle t : [h]((f_{h1} \wedge f_3 \wedge f_{in}) \rightsquigarrow f_{jn})\rangle$$

the corresponding rule in program $\Pi \cup \Pi_2$ is replaced by

$$\langle t : [h]((f_{h1} \wedge \neg((f_1 \vee f_3) \wedge \neg f_{i1} \wedge \cdots \wedge \neg f_{in})$$

$$\wedge\neg((f_1 \vee f_3) \wedge f_{i1}) \wedge \cdots \wedge \neg((f_1 \vee f_3) \wedge f_{in})) \rightsquigarrow f_{h2})\rangle$$

$$\langle t : [h]((f_{h1} \wedge (f_1 \vee f_3) \wedge \neg f_{i1} \wedge \cdots \wedge \neg f_{in}) \rightsquigarrow f_2)\rangle$$

$$\langle t : [h]((f_{h1} \wedge (f_1 \vee f_3) \wedge f_{i1}) \rightsquigarrow f_{j1})\rangle$$

$$\cdots$$

$$\langle t : [h]((f_{h1} \wedge (f_1 \vee f_3) \wedge f_{in}) \rightsquigarrow f_{jn})\rangle$$

Both above two rules will be simplified as

$$\langle t : [h]((f_{h1} \wedge \neg(f_1 \vee f_3)) \rightsquigarrow f_{h2})\rangle \tag{4.16}$$

It is easy to see that the other rules are equivalent to the same LTL formulas.

For each occurrence of h in the body of a rule that is not preceded by ":" of the form

$$\langle t : [r]([h](h_1 \rightsquigarrow h_2)), \tag{4.17}$$

the corresponding rule in $\Pi \cup \Pi_1$ is translated as

$$((f_1 \wedge \neg f_{i1} \wedge \cdots \wedge \neg f_{in}) \Rightarrow f_2) \wedge ((f_1 \wedge f_{i1}) \Rightarrow f_{j1}) \wedge \cdots \wedge ((f_1 \wedge f_{in}) \Rightarrow f_{jn}) \wedge$$
$$((f_3 \wedge \neg f_{i1} \wedge \cdots \wedge \neg f_{in}) \Rightarrow f_2) \wedge ((f_3 \wedge f_{i1}) \Rightarrow f_{j1}) \wedge \cdots \wedge ((f_3 \wedge f_{in}) \Rightarrow f_{jn})$$

the corresponding rule in $\Pi \cup \Pi_2$ is translated as

$$(((f_1 \vee f_3) \wedge \neg f_{i1} \wedge \cdots \wedge \neg f_{in}) \Rightarrow f_2) \wedge (((f_1 \vee f_3) \wedge f_{i1}) \Rightarrow f_{j1}) \wedge \cdots \wedge (((f_1 \vee f_3) \wedge f_{in}) \Rightarrow f_{jn})$$

it is easy to see that these two LTL formulas are equivalent.

Thus the $\Pi \cup \Pi_1$ and $\Pi \cup \Pi_2$ are equivalent. Π_1 and Π_2 are strongly equivalent.

□

Proposition 14. *Let $r \in R$, $h \in R \cup G$, $g \in G$, and f_1, f_2, and f_3 be arbitrary ER-LTL formulas.*

$$\Pi_3 : \ \langle h : [r](f_1 \rightsquigarrow f_2) \rangle$$

and

$$\Pi_4 : \ \langle h : [r](f_1 \rightsquigarrow f_1 \wedge f_2) \rangle$$

are strongly equivalent.

Proof. Given any program Π, now prove that $\Pi \cup \Pi_3$ and $\Pi \cup \Pi_4$ are equivalent.

The rule in Π_3 and Π_4 only differ in the right part of the $\rightsquigarrow$. They share the same formula in the left part of symbol $\rightsquigarrow$, meaning that all occurrence of $Tr(f_1 \rightsquigarrow f_2)$ in $Tr(\Pi \cup \Pi_3)$ are replaced with $Tr(f_1 \rightsquigarrow (f_1 \wedge f_2))$, and the output is $Tr(\Pi \cup \Pi_4)$. Based on properties of LTL, $f_1 \Rightarrow f_2$ and $f_2 \Rightarrow (f_1 \wedge f_2)$ are equivalent. Thus Π_3 and Π_4 are strongly equivalent. □

Similarly, the following proposition is true. It helps to simply a program.

Proposition 15. *Let $r \in R$, $h \in R \cup G$, $g \in G$, and f_1, f_2, and f_3 be arbitrary ER-LTL formulas*

$$\Pi_5 : \ \langle g : [r](f_1 \rightsquigarrow \top)\rangle$$

and

$$\Pi_6 : \ \langle r_1 : [r_2](\bot \rightsquigarrow f_1)\rangle$$

and $\emptyset$ are strongly equivalent.

These definitions on strongly equivalent will be more interesting if they can be used to simplify a program. The following sub-section discusses its application in progressing an ER-LTL program.

Progressing ER-LTL

In [BK98], authors proposed the progressing for MITL. The approach can be easily adopted in progressing LTL formulas with a different way of dealing with the symbol U. Now consider the way of progressing an LTL program T after observing a sequence of states $s_0, \cdots, s_i$. It is illustrated in Algorithm 4.4.

Now extend the work on progressing to non-monotonic logics. An ER-LTL program T is progressed after observing a states s. It is illustrated in Algorithm 4.4.

Definition 40. *$Progress(T, s)$ for an ER-LTL program T, and a state s is defined as*

$$(T/g) \cup \bigcup_{l \in T} Progress(l, s) \tag{4.18}$$

where T/g is the set of rules in T with the head not in $\{g\}$.

A program T can be simplified by removing rules whose heads do not occur in the body of any rule.

Note that in the progressing of the formulas, a set of rules are introduced that progress one step on state s for each rule in the existing program. Also note that the

[Progressing LTL] **Inputs**: A state s_i, with formula label f.
Output: a new formula $Progress(f, s_i)$ representing the temporal formula for the successor state.
Algorithm:

1. If $f = p$, and the current state with p true, then $Progress(f, s) = \top$;
2. If $f = \neg f_1$, then $Progress(f, s) = \neg Progress(f_1, s)$;
3. If $f = f_1 \vee f_2$, then $Progress(f, s) = Progress(f_1, s) \vee Progress(f_2, s)$;
4. If $f = \bigcirc f_1$, then $Progress(f, s) = f_1$;
5. If $f = f_1 \mathsf{U} f_2$, then $Progress(f, s) = Progress(f_2, s) \vee (Progress(f_1, s) \wedge (f_1 \mathsf{U} f_2))$.

As usual $\neg(\neg f_1 \vee \neg f_2)$ is denoted by $f_1 \wedge f_2$, $\top \mathsf{U} f$ is denoted by $\Diamond f$, and $\neg\Diamond\neg f$ is denoted by $\Box f$.

[Progressing ER-LTL] For a rule l of the form $\langle h : f \rangle$ in an ER-LTL program T. $Progress(l, s)$ is defined as $\{\langle h_s : Progress(f, s)\rangle\}$ if $h \in R$, or $\{\langle h : Progress(f, s)\rangle\}$ if $h \in \{g\}$, where $Progress(f, s)$ for a formula f and a state s is as follows:

1. If $f = p$, and the current state with p true, then $Progress(f, s) = \top$;
2. If $f = \neg f_1$, then $Progress(f, s) = \neg Progress(f_1, s)$;
3. If $f = f_1 \vee f_2$, then $Progress(f, s) = Progress(f_1, s) \vee Progress(f_2, s)$;
4. If $f = \bigcirc f_1$, then $Progress(f, s) = f_1$;
5. If $f = f_1 \mathsf{U} f_2$, then $Progress(f, s) = Progress(f_2, s) \vee (Progress(f_1, s) \wedge (f_1 \mathsf{U} f_2))$;
6. If $f = [r](f_1 \rightsquigarrow f_2)$, then
   $Progress(f, s) = [r_s](Progress(f_1, s) \rightsquigarrow Progress(f_2, s))$.

---

new rules introduced with head r_s for $r \in R$. This helps to link this rule r_s to other rules.

Definition 41 (*Progress*$(T, (s_0; s_1; \cdots; s_i))$)**.** *Progress*$(T, (s_0; s_1; \cdots; s_i))$ *is defined as*

$Progress(\ldots Progress(Progress(T, s_0), s_1), \cdots, s_i)$.

The progressed program can be simplified by strong equivalences properties of programs. This is illustrated in the following example.

Example 20. *Suppose the agent has the goal:*

$$\langle g : [r_1]((\Diamond p) \rightsquigarrow (\bigcirc[r_3](\top \rightsquigarrow \Diamond q)))\rangle$$

$$\langle r_1 : [r_2]((\bigcirc p) \rightsquigarrow (\Box r))\rangle$$

$$\langle r_3 : [r_4](r \rightsquigarrow \top)\rangle$$

The goal is equivalent to LTL formula

$$((\Diamond p \wedge \neg \bigcirc p) \Rightarrow (\bigcirc(r \vee \Diamond q))) \wedge (\bigcirc p \Rightarrow \Box r) \tag{4.19}$$

Suppose the agent is initially in a state $\{p, q, \neg r\}$ *and has executed an action and get to a state* $\{\neg p, q, r\}$*. Now progress the goal by one step on state* $\{p, q, \neg r\}$ *to:*

$$\langle r_1 : [r_2]((\bigcirc p) \rightsquigarrow (\Box r))\rangle \tag{4.20}$$

$$\langle r_3 : [r_4](r \rightsquigarrow \top)\rangle \tag{4.21}$$

$$\langle g : [r_{1s}](\top \rightsquigarrow ([r_3](\top \rightsquigarrow \Diamond q)))\rangle \tag{4.22}$$

$$\langle r_{1s} : [r_{2s}](p \rightsquigarrow \bot)\rangle \tag{4.23}$$

$$\langle r_{3s} : [r_{4s}](\bot \rightsquigarrow \top)\rangle \tag{4.24}$$

As in Formula 4.18, Rule 4.20 and Rule 4.21 are in (T/g)*, which are copied from the previous program. The last 3 rules are in* $\cup_{l \in T} Progress(l, s)$*.*

This program can be simplified as

$$\langle r_3 : [r_4](r \rightsquigarrow \top)\rangle$$

$$\langle g : [r_{1s}](\top \rightsquigarrow ([r_3](\top \rightsquigarrow \Diamond q)))\rangle$$

$$\langle r_{1s} : [r_{2s}](p \rightsquigarrow \bot)\rangle$$

which is equivalent to LTL formula

$$(r \vee \Diamond q) \wedge \neg p \tag{4.25}$$

It is equivalent to the LTL formula after progressing Formula 4.19 on state $\{p, q, \neg r\}$.

In reality, it is possible that this goal after progressing is refined by appending new rules. For now, after executing another action, the agent gets to a state $\{\neg p, \neg q, \neg r\}$. *Further progress the goal:*

$$\langle r_3 : [r_4](r \rightsquigarrow \top)\rangle$$
$$\langle r_{1s} : [r_{2s}](p \rightsquigarrow \bot)\rangle$$
$$\langle r_{3t} : [r_{4t}](\bot \rightsquigarrow \top)\rangle$$
$$\langle g : [r_{1st}](\top \rightsquigarrow ([r_{3t}](\top \rightsquigarrow \Diamond q)))\rangle$$
$$\langle r_{1st} : [r_{2st}](\bot \rightsquigarrow \bot)\rangle$$

It can be simplified as

$$\langle g : [r_{1st}](\top \rightsquigarrow ([r_{3t}](\top \rightsquigarrow \Diamond q)))\rangle$$

It is equivalent to LTL formula $\Diamond q$, *and is equivalent to the formula after progressing Formula 4.25.*

Note that the initial program can be more complicated than the one given in this example. For example, if the initial program has a rule of the form $\langle g : [r_1](p \rightsquigarrow (\Diamond[r_3](\top \rightsquigarrow \Diamond q)))\rangle$, it would be complicated to evaluate the sub-formula $\Diamond[r_3](\top \rightsquigarrow \Diamond q)$ when progressing on a state.

Due to the recursive nature in the definition of translating an ER-LTL program to a LTL program, and due to the only difference in the progressing steps, it is easy to prove the following proposition:

Proposition 16. *Let ErProgress be the progressing approach defined for language ER-LTL. Let LtlProgress be the progressing approach defined for language LTL. It is true that*

$$Tr(ErProgress(T,\sigma)) = LtlProgress(Tr(T),\sigma)$$

This implies that the progressing steps are well-defined. Now users can specify a goal in ER-LTL and give it to agents. Once users want to change the goal of an agent, they can do so by appending new rules and send these rules to the agent. The agent receiving the new instructions can change its goal non-monotonically. In the case that the agent has executed some actions. The agent can progress its ER-LTL goals based on earlier states in the trajectory before translating the ER-LTL goals to LTL for further executions.

4.5 Non-monotonic Extension of CTL*, π-CTL*, and P-CTL*

The approach of defining N-LTL can be applied to other temporal logics. Different from LTL, in CTL* [ES89, Eme90], π-CTL* [BZ04], and P-CTL* [BZ06], formulas are categorized as state formulas and path formulas. As a consequence, the corresponding non-monotonic languages should be defined with respect to path formulas and state formulas respectively. Now define the way of extending the definition of non-monotonic temporal logics for CTL*, called N-CTL*. Language N-CTL* is based on CTL*.

Definition 42. *Let $\{g\}$, R_{pf}, R_{sf} and P be four disjoint sets of atoms. Let $\langle r_{pf}\rangle$ be an atom in R_{pf}, $\langle r_{sf}\rangle$ be an atom in R_{sf}, $\langle p\rangle$ be an atom in P, $e \in \{g\} \cup R_{sf}$. Let $\langle sf\rangle$ denote a state formula, and $\langle pf\rangle$ denote a path formula.*

$$\langle sf\rangle ::= \langle p\rangle \mid \langle sf\rangle \wedge \langle sf\rangle \mid \langle sf\rangle \vee \langle sf\rangle \mid \neg\langle sf\rangle \mid \mathsf{E}\langle pf\rangle \mid \mathsf{A}\langle pf\rangle \mid$$
$$[\langle r_{sf}\rangle]\langle sf\rangle \mid [[\langle r_{sf}\rangle]]\langle sf\rangle$$
$$\langle pf\rangle ::= \langle sf\rangle \mid \langle pf\rangle \vee \langle pf\rangle \mid \neg\langle pf\rangle \mid \langle pf\rangle \wedge \langle pf\rangle \mid \bigcirc\langle pf\rangle \mid$$
$$\langle pf\rangle \,\mathsf{U}\, \langle pf\rangle \mid \Diamond\langle pf\rangle \mid \Box\langle pf\rangle \mid [\langle r_{pf}\rangle]\langle pf\rangle \mid [[\langle r_{pf}\rangle]]\langle pf\rangle$$

An N-CTL* formula is a set of rules $\langle e : sf \rangle$, or $\langle r_{pf} : pf \rangle$. Each of e or r_{pf} is the head *of the rule, and each of sf or pf is the* body *of the rule $\langle e : sf \rangle$ or $\langle r_{pf} : pf \rangle$.* □

The semantics of N-CTL* is defined in a way similar to the definition of the semantics of N-LTL. Now illustrate the usefulness of N-CTL* through two examples.

Example 21. *The initial goal is to require that p be true until q is reached. However, it is realized that in some domain, this goal is too strong as no plan can always have p until reaching q. If in some states in the main path, all possible trajectories will have p true in the future, then it may considered as an exception and there is no need to have p. The initial goal can be represented as $\langle g : [r](p)\mathsf{U}q \rangle$ in N-CTL*. This goal is equivalent to $p\mathsf{U}q$ in CTL*. It can be further refined by adding one rule about the exception r as $\langle r : \mathsf{A}\Diamond p \rangle$. The revised goal is equivalent to $(p \vee \mathsf{A}\Diamond p)\mathsf{U}q$, which is equivalent to $(\mathsf{A}\Diamond p)\mathsf{U}q$.*

Example 22. *Initially, the goal is to make sure that in most trajectories starting from the initial state, $\Diamond p$ is true. However, later on, users may require that once $\Diamond q$ is satisfied by some trajectories, those trajectories are considered as exceptional ones and users do not require them to satisfy $\Diamond p$. The initial goal is represented as: $\langle g : \mathsf{A}[r](\Diamond p) \rangle$. Later, the goal can be weakened by adding one more rule about the exceptions: $\langle r : (\Diamond q) \rangle$.*

The initial goal is equivalent to the CTL* formula $\mathsf{A}\Diamond p$ and the revised goal is equivalent to $\mathsf{A}(\Diamond p \vee \Diamond q)$.

It is noted that the approach in N-LTL and ER-LTL can be applied to other temporal logics such as CTL*, π-CTL* and P-CTL*.

4.6 A Program Translating an ER-LTL goal to a LTL goal

To help translating a goal represented in ER-LTL to a LTL formula so as to be used by other existing systems that accept LTL formulas, a program of translating an ER-LTL program to LTL is given. The program first call a Lexical analyzer generator Lex and a Yacc compatible compiler Bison [bis] to generate parses. Afterwards, an implementation of the Algorithm 4.3 translates the parsed program to an LTL formula. The program is available at http://www.public.asu.edu/∼jzhao6/erltl.tar.

A few programs and their corresponding outputs are listed below:

```
1. Input:
{g0 :[r1](a \arrow b) }
{r1 : [r2](c \arrow d) }.
Output:
g :  [] (\top \arrow ((a  \and \not (c)) \arrow (b)) \ and
(((c) \and (a)) \arrow (d)) )

2. Input:
{g0 :[r1](a \arrow b) }
{r1 : [r2](c \arrow d) }
{r1 : [r3](e \arrow \diamond p) }.
Output:
g :  [] (\top \arrow ((a \and \not (c) \and \not (e))
\arrow (b)) \and (((c) \and (a)) \arrow (d)) \and
(((e) \and (a)) \arrow ( \diamond p)) )

3. Input:
{g0: [r1]( a \arrow ([r2]((\diamond c) \arrow (\Box d))))}
{r2: [r3]( top \arrow (\not e))}.
```

```
Output:
g :  [] (\top \arrow ((a) \arrow ((( \diamond c \and
 \not (top)) \arrow ( \Box d)) \and (((top) \and
 ( \diamond c)) \arrow ( - e)))) )
```

4.7 Discussion

Languages ER-LTL and N-LTL are compared now to check whether we can translate program in one language to an equivalent program in the other language.

Comparing ER-LTL and N-LTL

ER-LTL and N-LTL different in the way of using completion: N-LTL does completion on the formulas disjunctively, and ER-LTL does completion on preconditions of the exceptions conjunctively.

In ER-LTL, the weak exceptions in N-LTL are discarded. Both weak exception and strong exception will be taken care of by the new symbol $\rightsquigarrow$. The way of distinguishing preconditions and consequents with the symbol $\rightsquigarrow$ in ER-LTL is more intuitive. Arbitrary revision of the goals is allowed. For example, in the extreme case, All existing requirements can be eliminated by having a set of rules of the form $\langle [r_i] : (\top \rightsquigarrow \top) \rangle$ where r_i occurs in the ER-LTL program, and then add the new requirements. In N-LTL, not all existing requirements can be eliminated as in ER-LTL.

These two languages are also related. If an N-LTL program has only strong exceptions and having at most one rule for each $r \in R$, it can be translated rule by rule to an ER-LTL program. The N-LTL program and the translated ER-LTL program are equivalent to the same LTL program. The translation is as follows: Given the rule $\langle h : f_1 \rangle$, it is written as

$$\langle h : [r_{\_}](\top \rightsquigarrow f_1) \rangle$$

where there is no rule in the program with head $r_{\_}$. Further, for any sub-formula in f_1 of the form $[[r]](f_2)$, it is replaced as $[r_1](\top \rightsquigarrow f_2)$.

Example 23. *Given an N-LTL program as follows.*

$$\langle g : [[r_1]](f_1)\rangle$$

$$\langle r_1 : [[r_2]](f_2) \wedge f_3\rangle$$

It can be translated to an ER-LTL as

$$\langle g : [r_{\_}](\top \rightsquigarrow [r_1](\top \rightsquigarrow f_1))\rangle$$

$$\langle r_1 : [r_{\_}](\top \rightsquigarrow [r_2](\top \rightsquigarrow f_2) \wedge f_3)\rangle$$

They are equivalent to the same LTL program.

Related Works

There are a few work that are related to the non-monotonic logics proposed in this chapter.

Paper [FH91] has somewhat similar aim as ours. It extends auto-epistemic logic with temporal operators. It is very different from this work, and does not discuss the issues such as exceptions, weak exceptions, elaboration tolerance that are discussed in N-LTL and ER-LTL.

N-LTL and ER-LTL are related to traditional non-monotonic logics including logic program and default logic, especially when the underline formula in the default logic are considered as LTL formulas. The occurrences of the symbols are considered as the triggering of exceptions. The idea of completion are used when rules are defined for exceptions. However, they are different in various aspects. For example, semantics of default logic depends on models of the program, and it does not allow revision of sub-formulas in a formula. Semantics of the objective language such as entailment relations are involved in defining semantics of the default logic. On the other hand, semantics of ER-LTL relies on the translation of the

logic to LTL before the temporal operands are examined. ER-LTL also has a great advantage in terms of complexity.

Recently, a paper [PSBZ10] on applying default logic to temporal logics was published. Different from default logic, the logic replace each propositional formula with a temporal formula. One limitation of the work is that calculating a model of the logic is of high complexity. Also, it is not natural to apply the logic to real applications, especially when a part of the previously specified goal need to be changed.

This work is also similar to defeasible logic [Nut87] such that rules are treated as preconditions and effects.

Applying the Techniques in ER-LTL to Propositional Logic

The constructs in ER-LTL can be used to define a defeasible logic where LTL is replaced by simple propositional logic. Lets call this logic as ER-POP. It has some interesting properties.

ER-POP shares some common properties with defeasible logic [] and its extension plausible logic [Bil98]. Labels are used in both languages to denote rules. Besides, an ordering on rules are defined in each language and the ordering is acyclic.

However, ER-POP differs defeasible logic in a few aspects. Firstly, ER-POP allows nested rules. In defeasible logic, each program is a few sets of rules together with an ordering on rules. Nested rules are not allowed. Secondly, the ordering in these two languages have different semantics. The tree structure ordering in ER-POP denotes the ordering of revising sub-formulas. On the other hand, the ordering in defeasible logic denotes whether a rule can be fired or not. The ordering in defeasible logic is a linear order. A rule cannot be fired if its conclusion contradict with that is implied by rules with higher priority. Finally, the models of ER-POP rely on a translation to propositional logic. The models of defeasible logic rely on

a gradually growing set of literals by firing rules.

4.8 Summary

In many domains, goals specified might be further revised or partially retracted due to incomplete information users have about the domain initially. Non-monotonic temporal logics can be used for specifying goals which can then be revised in an elaboration tolerant manner. This chapter discussed two non-monotonic extensions of LTL. The idea of completion and exception from logic programming and the idea of a surface non-monotonic logic that can be translated to a monotonic logic, from Reiter, are borrowed. The approach of extending LTL can be used to extend other monotonic temporal logics such as CTL and CTL*.

The chapter motivated the need for such non-monotonic temporal logics from the point of view of needing ways to express goals that can be changed in an elaboration tolerant manner. Several properties of such logics are presented and their application in modeling revisions is illustrated.

This chapter also discussed progressing of an ER-LTL program. This is important as agent received new requirements may already executed some actions to satisfy earlier goals. Thus the agent need to progress the previous requirements and the new requirements based on the trajectory of the agent.

In terms of future work, the approach used in syntactic formula revision is not restricted to temporal formulas. Its implications vis-a-vis existing non-monotonic logics, belief revision mechanisms, and formalizing natural language discourses [3] needs to be explored.

[3] Sentences in natural language have references such as “that”, “those”, “the” may hint about the replacement. However, it is a challenge problem for the replacement.

Chapter 5

PREF-Π-CTL*: GOAL SPECIFICATION WITH DYNAMIC PREFERENCE IN NON-DETERMINISTIC DOMAINS

In this chapter, a goal specification language Pref-π-CTL* for representing goals with dynamic preferences in a non-deterministic domain is proposed. Pref-π-CTL* extends π-CTL* by introduction a new binary operator $\lhd$ to denote preferences among temporal formulas. This language is more intuitive and is simpler in representing nested preferences and dynamic preferences as the new operator $\lhd$ is treated the same way as other temporal operators. Some of these goals cannot be captured in other temporal logics with preferences. Further, a program is given for finding one particular planning problem in Pref-π-CTL* that defines preferences relations among weak, strong, and strong cyclic plans in a non-deterministic domain.

5.1 Introduction

An important aspect of designing autonomous agent is to specify what users want for the agent. This is called goal specification. Different goal specification languages were proposed. Among them, temporal logics such as line temporal logic LTL [Pnu77], branching time temporal logic CTL* [EC82, ES89, Eme90], and their extensions [BK98, NS00, BKT01] have been proposed and used as goal specification languages in the autonomous agent community and planning community. CTL* is also extended to non-deterministic domains by quantifying over plans [AHK02, BZ04, BZ06].

Each goal specification language defines a set of goal formulas, and specifies a set of plans satisfying each goal formula. In the case that multiple plans satisfy a goal, it is interesting to find out more preferred plans among them. This is often done by defining a preference relation among goals, or among plans. One approach

of defining the preference relation among plans is to consider some goals as soft constraints [BCGR99]. As in PDDL3 [GL05], all plans satisfying hard constraints are considered acceptable plans, while plans also satisfying some soft constraints are more preferred among them. Preference relation can also be interpreted differently. In $\mathscr{PP}$ [SP06], a preference relation among plans is defined for each goal formula. The preference relation is defined to get the most preferred plans. As a consequence, goal g_1 prefers to g_2 meaning that if plans satisfying g_1 exist in the domain, choose such a plan. Otherwise, choose other plans satisfying g_2.

This chapter uses the notion of preference as in $\mathscr{PP}$ but consider non-deterministic domains. A plan in non-deterministic domain is also called a policy. This chapter argues that given that each goal is a mapping from transition graphs and initial states to sets of trajectories (or sets of set of trajectories), and given that agents can quantify over policies, preference relations among goals as in $\mathscr{PP}$ can be captured without explicit comparison of temporal formulas.

For instance, in language P-CTL*, each goal is a mapping from transition graphs and initial states to sets of set of trajectories. As a consequence, goals defined are adaptive to domains, meaning that for a given goal in the language, the same set of trajectories (or policy) is acceptable in one transition graph while not acceptable in the other transition graph, if there are more preferred set of trajectories (or policy) in the domain. This implies that among the set of policies satisfying the goal, users usually prefer some policies over the others. For example, $(\mathscr{E}\mathscr{P}g_1 \Rightarrow g_1) \wedge ((\neg\mathscr{E}\mathscr{P}g_1 \wedge \mathscr{E}\mathscr{P}g_2) \Rightarrow g_2)$ is a goal in P-CTL* states that users prefer policies satisfying g_1 to policies satisfying g_2. In a transition graph, if there is a policy satisfying g_1, the agent needs to take a policy satisfying g_1. Otherwise, a policy satisfying g_2 is acceptable. Even though the preference relation can be captured in P-CTL*. The goal stated above is not as intuitive as the formula $g_1 \lhd g_2$, which explicitly defines the preference relations among temporal formulas, or policies

satisfying the temporal formulas.

Now consider another example from [BZ06] given as in Figure 3.1 to illustrate that P-CTL* is capable of expressing preferences among sub-goals while a language with explicit preference relation among formulas is more intuitive in capturing the goals.

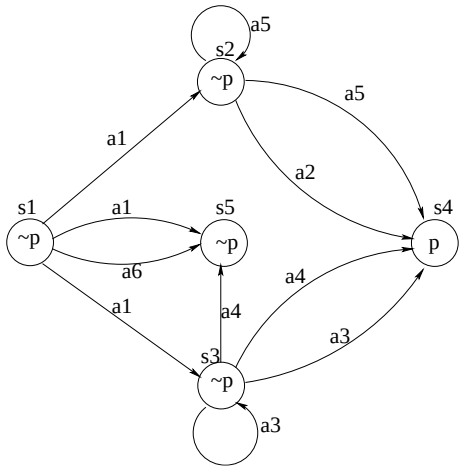

Figure 5.1: Transition diagram in a non-deterministic domain

In the example, a goal that is adaptive to domains and changes its expectation in different states is defined as follows: In any state of the domain, the agent is trying to find a strong plan, and then a strong cyclic plan if no strong plan can be found, and a weak plan if no strong cyclic plan can be found. Strong plan, strong cyclic plan, and weak plan can be expressed in π-CTL* as $\mathsf{A}_{pol}\Diamond p$, $\mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)$, and $\mathsf{E}_{pol}\Diamond p$ respectively. This goal above is expressed in P-CTL* [BZ06] as $\mathsf{A}_{pol}\Box((\mathscr{E}\mathscr{P}\mathsf{E}_{pol}\Diamond p \Rightarrow \mathsf{E}_{pol}\Diamond p) \wedge (\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p) \Rightarrow \mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)) \wedge (\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p \Rightarrow \mathsf{A}_{pol}\Diamond p))$. Note that in this example, any strong plan is also a strong cyclic plan, which in turn is a weak plan. Without this property among sub-goals, the formula above will be more complicated. This goal states that the agent has a few options in any state of the planning problem and a preferences relation is defined among these options. A

more intuitive representation of the goal would be:

$$\mathsf{A}_{pol}\Box((strong \lhd strong\_cyclic) \lhd weak)$$

or

$$\mathsf{A}_{pol}\Box((\mathsf{A}_{pol}\Diamond p \lhd \mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)) \lhd \mathsf{E}_{pol}\Diamond p). \tag{5.1}$$

It states that in any state of the plan, users always prefer to have a strong plan, and users prefer to have a strong cyclic plan if it is not possible to have a strong plan.

This goal is useful. For example, suppose that the strong, weak, and strong cyclic plans be the different operation plans a Doctor has for an operation. Due to non-deterministic outcome of the actions, the Doctor needs to make sure to try his best to save the patient's life. Thus the Doctor need to choose the best actions in any state of the plan.

This chapter proposes language Pref-π-CTL* that is based on π-CTL* but with a preference relation $\lhd$ between state formulas. It shows that the goal above can be represented in Pref-π-CTL* as Formula 5.1.

Language Pref-π-CTL* has some good properties comparing to other goal specification languages with preferences. It is noted that the goal above cannot be expressed in $\mathscr{PP}$ [SP06] even after extending $\mathscr{PP}$ to non-deterministic domains. In $\mathscr{PP}$, to capture preference relations, logics are defined by attaching rules about preference relations to underline goal specification languages. It does not allow temporal operators to wrap around general preferences. More importantly, it is not easy, if it is possible to express preference relations that are dynamic w.r.t. given conditions in $\mathscr{PP}$. One such example is that users prefer g_1 over g_2 under one condition but prefer g_2 over g_1 under other conditions. New language Pref-π-CTL* allows nested preferences and allows dynamic preference relations as users do not need to distinguish between basic desire formulas and general preferred formulas as in [SP06].

The rest of this chapter is organized as follows. Section 5.2 defines syntax and semantics of language Pref-π-CTL*. Section 5.3 studies properties of Pref-π-CTL*. Section 5.4 compares Pref-π-CTL* with related languages. This chapter is end with summary and future work.

5.2 Pref-π-CTL*: Extending CTL* with Preferences

Now syntax and semantics of the goal specification language with preferences are defined. The logic extends π-CTL* by adding a preference relation to temporal formulas. The logic is called Pref-π-CTL*.

Similar to π-CTL*, formulas in Pref-π-CTL* are either state formulas or path formulas.

Definition 43. *Let $\langle p \rangle$ be an atomic proposition, $\langle sf \rangle$ be a state formula, and $\langle pf \rangle$ be a path formula.*

$\langle sf \rangle ::= \langle p \rangle \mid \langle sf \rangle \wedge \langle sf \rangle \mid \langle sf \rangle \vee \langle sf \rangle \mid \neg \langle sf \rangle \mid \mathsf{E}\langle pf \rangle \mid \mathsf{A}\langle pf \rangle \mid \mathsf{E}_{pol}\langle pf \rangle \mid \mathsf{A}_{pol}\langle pf \rangle \mid \langle sf \rangle \lhd \langle sf \rangle$

$\langle pf \rangle ::= \langle sf \rangle \mid \langle pf \rangle \vee \langle pf \rangle \mid \neg \langle pf \rangle \mid \langle pf \rangle \wedge \langle pf \rangle \mid \langle pf \rangle \,\mathsf{U}\, \langle pf \rangle \mid \bigcirc \langle pf \rangle \mid \Diamond \langle pf \rangle \mid \Box \langle pf \rangle$

$\Box$

The operator $\lhd$ is allowed to occur recursively on state formulas. For example, $(\mathsf{A}_{pol}\Box q) \lhd \mathsf{A}_{pol}\Diamond(p \lhd q)$ states that users prefer $\mathsf{A}_{pol}\Box q$ to $\mathsf{A}_{pol}\Diamond(p \lhd q)$, and in $\mathsf{A}_{pol}\Diamond(p \lhd q)$, users prefer to reach a state where p is true to a state where q is true. A goal satisfying "a state is preferred over another state" is now defined.

Definition 44 (Truth of state formulas in Pref-π-CTL*)**.** *Truth of a state formula is defined with respect to a triple (s_j, Φ, π) where s_j is a state, Φ is the transition function, and π is a policy that is a mapping from states to actions for all states in the transition graph.*

- $(s_j, \Phi, \pi) \models p$, $\neg sf$, $sf_1 \wedge sf_2$, $sf_1 \vee sf_2$, $\mathsf{E}\ pf, \mathsf{A}\ pf$, $\mathsf{E}_{pol}\ pf$, $\mathsf{A}_{pol}\ pf$ *are defined similarly as in* π*-CTL**.

- $(s_j, \Phi, \pi) \models sf_1 \lhd sf_2$ *iff*

  - $(s_j, \Phi, \pi) \models sf_1$, *and* $(s_j, \Phi, \pi) \models sf_2$, *or*
  - $(s_j, \Phi, \pi) \models sf_1$, *and no other policies such that* $(s_j, \Phi, \pi_2) \models sf_1$ *and* $(s_j, \Phi, \pi_2) \models sf_2$, *or*
  - $(s_j, \Phi, \pi) \models sf_2$, *and no other policies* π_2 *such that* $(s_j, \Phi, \pi_2) \models sf_1$, *or*
  - *No policies* π_2 *such that* $(s_j, \Phi, \pi_2) \models sf_1$ *or* $(s_j, \Phi, \pi_2) \models sf_2$. □

Truth of path formulas is defined similarly as in π-CTL*.

Definition 45 (Truth of path formulas in Pref-π-CTL*)**.** *The truth of path formulas is now defined with respect to the quadruple* (s_j, Φ, π, σ), *where* s_j *is a state,* Φ *is the transition function,* π *is a policy, and* σ *is a trajectory* $s_j, s_{j+1}, \ldots$.

$(s_j, \Phi, \pi, \sigma) \models sf$, $\neg pf$, $pf_1 \wedge pf_2$, $pf_1 \vee pf_2$, $\bigcirc pf$, $\Box pf$, $\Diamond pf$, $pf_1 \mathsf{U} pf_2$ *are defined similarly as in* π*-CTL***;*

Based on the semantics of Pref-π-CTL*, $(s_j, \Phi, \pi) \models sf_1 \lhd sf_2$ iff

$$(s_j, \Phi, \pi) \models (sf_1 \wedge sf_2) \vee ((\neg \mathscr{E}\mathscr{P}(sf_1 \wedge sf_2)) \wedge sf_1)$$
$$\vee ((\neg \mathscr{E}\mathscr{P} sf_1) \wedge sf_2) \vee (\neg \mathscr{E}\mathscr{P} sf_1 \wedge \neg \mathscr{E}\mathscr{P} sf_2).$$

in P-CTL*. This implies that each formula in Pref-π-CTL* can be translated to a formula in P-CTL*.

Now define when a policy satisfies a goal g given an initial state s_0, and a transition function Φ in Pref-π-CTL*.

Definition 46 (Policy satisfies a Pref-π-CTL* goal)**.** *Given an initial state* s_0, *a state mapping policy* π, *a transition function* Φ, *and a Pref-CTL** *goal* φ, π *is a policy for* φ *from* s_0, *iff* $(s_0, \Phi, \pi) \models \varphi$ *in Pref-*π*-CTL**. □

Note that in a transition system, if there is no policy satisfying either φ_1 or φ_2, any policy in the transition system satisfies $\varphi_1 \lhd \varphi_2$. In the case users prefer φ_1 over φ_2 while want a least one of them is satisfied, users can represent the goal as $(\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \lhd \varphi_2)$. The following section lists a few properties of a Pref-π-CTL* program before illustrates its applications.

5.3 Properties of Pref-π-CTL*

This section shows a few properties of Pref-π-CTL* that helps in simplifying a program.

Proposition 17. *Let sf_1, sf_2, and sf_3 be state formulas in Pref-π-CTL*. It is true that $(s,\Phi,\pi) \models (sf_1 \wedge sf_3) \lhd (sf_2 \wedge sf_3)$ if $(s,\Phi,\pi) \models (sf_1 \lhd sf_2) \wedge sf_3$.*

Proof. Given that $(s,\Phi,\pi) \models (sf_1 \lhd sf_2) \wedge sf_3$, it is known that $(s,\Phi,\pi) \models sf_3$ and one of the following is true:

1. $(s,\Phi,\pi) \models sf_1 \wedge sf_2$,
2. $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_2) \wedge sf_1$,
3. $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge sf_2$,
4. $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_2$.

To prove that $(s,\Phi,\pi) \models (sf_1 \wedge sf_3) \lhd (sf_2 \wedge sf_3)$, it is sufficient to prove that any of the following is true:

- $(s,\Phi,\pi) \models sf_1 \wedge sf_2 \wedge sf_3$,
- $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_2 \wedge sf_3) \wedge (sf_1 \wedge sf_3)$,
- $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_3) \wedge (sf_2 \wedge sf_3)$,
- $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_3) \wedge \neg\mathscr{E}\mathscr{P}(sf_2 \wedge sf_3)$.

In case of Item 1, it is easy to know that $(s,\Phi,\pi) \models sf_1 \wedge sf_2 \wedge sf_3$. In case of Item 2, as there is no policy satisfying sf_1 and sf_2, there is no policy satisfying $sf_1 \wedge sf_2 \wedge sf_3$ as well. $(s,\Phi,\pi) \models sf_1$ is true and $(s,\Phi,\pi) \models sf_3$, thus $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_2 \wedge sf_3) \wedge (sf_1 \wedge sf_3)$. In case of Item 3, as there is no policy satisfying sf_1, there is no policy satisfying sf_1 and sf_3. It is known that $(s,\Phi,\pi) \models sf_2$ and $(s,\Phi,\pi) \models sf_3$, thus $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_3) \wedge (sf_2 \wedge sf_3)$. In case of Item 4, as there is no policy satisfying sf_1, there is no policy satisfying $sf_1 \wedge sf_3$. As there is no policy satisfying sf_2, there is no policy satisfying $sf_2 \wedge sf_3$. Thus $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_3) \wedge \neg\mathscr{E}\mathscr{P}(sf_2 \wedge sf_3)$. □

Proposition 18. *Let sf_1, sf_2, and sf_3 be state formulas in Pref-π-CTL*. It is true that*

- $(s,\Phi,\pi) \models (sf_1 \lhd sf_2) \vee (sf_1 \lhd sf_3)$ *if* $(s,\Phi,\pi) \models sf_1 \lhd (sf_2 \vee sf_3)$,
- $(s,\Phi,\pi) \models (sf_1 \lhd sf_3) \vee (sf_2 \lhd sf_3)$ *if* $(s,\Phi,\pi) \models (sf_1 \vee sf_2) \lhd sf_3$,
- $(s,\Phi,\pi) \models (sf_1 \lhd sf_2) \wedge (sf_1 \lhd sf_3)$ *only if* $(s,\Phi,\pi) \models sf_1 \lhd (sf_2 \wedge sf_3)$,
- $(s,\Phi,\pi) \models (sf_1 \lhd sf_3) \wedge (sf_2 \lhd sf_3)$ *only if* $(s,\Phi,\pi) \models (sf_1 \wedge sf_2) \lhd sf_3$.

Proof. First prove that $(s,\Phi,\pi) \models (sf_1 \lhd sf_2) \vee (sf_1 \lhd sf_3)$ if $(s,\Phi,\pi) \models sf_1 \lhd (sf_2 \vee sf_3)$.

Given that $(s,\Phi,\pi) \models sf_1 \lhd (sf_2 \vee sf_3)$, one of the following is true:

1. $(s,\Phi,\pi) \models sf_1 \wedge (sf_2 \vee sf_3)$,
2. $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge (sf_2 \vee sf_3)) \wedge sf_1$,
3. $(s,\Phi,\pi) \models (\neg\mathscr{E}\mathscr{P}sf_1) \wedge (sf_2 \vee sf_3)$,
4. $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}(sf_2 \vee sf_3)$.

To prove that $(s,\Phi,\pi) \models (sf_1 \lhd sf_2) \vee (sf_1 \lhd sf_3)$, it is sufficient to show that one of the following is true:

- $(s,\Phi,\pi) \models sf_1 \wedge sf_2$,
- $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_2) \wedge sf_1$,
- $(s,\Phi,\pi) \models (\neg\mathscr{E}\mathscr{P}sf_1) \wedge sf_2$,
- $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_2$,
- $(s,\Phi,\pi) \models sf_1 \wedge sf_3$,
- $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_3) \wedge sf_1$,
- $(s,\Phi,\pi) \models (\neg\mathscr{E}\mathscr{P}sf_1) \wedge sf_3$,
- $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_3$.

It is easy to check the case in Item 1 and Item 3. Consider the case in Item 2. Given that $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge (sf_2 \vee sf_3)) \wedge sf_1$, it is true that $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge (sf_2 \vee sf_3)) \wedge sf_1$. Thus $(s,\Phi,\pi) \models sf_1 \wedge \neg\mathscr{E}\mathscr{P}((sf_1 \wedge sf_2) \vee (sf_1 \wedge sf_3))$. Thus $(s,\Phi,\pi) \models sf_1 \wedge \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_2)$. Similarly, given Item 4, it is known that $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_2$.

Similarly, it can be proved that $(s,\Phi,\pi) \models (sf_1 \lhd sf_3) \vee (sf_2 \lhd sf_3)$ if $(s,\Phi,\pi) \models (sf_1 \vee sf_2) \lhd sf_3$.

Now prove that given $(s,\Phi,\pi) \models (sf_1 \lhd sf_2) \wedge (sf_1 \lhd sf_3)$, $(s,\Phi,\pi) \models sf_1 \lhd (sf_2 \wedge sf_3)$ is true.

Given that $(s,\Phi,\pi) \models (sf_1 \lhd sf_2) \wedge (sf_1 \lhd sf_3)$, it is known:

1. $(s,\Phi,\pi) \models sf_1 \wedge sf_2$,
2. $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_2) \wedge sf_1$,

3. $(s,\Phi,\pi) \models (\neg\mathscr{E}\mathscr{P}sf_1) \wedge sf_2$,

4. $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_2$.

and

1. $(s,\Phi,\pi) \models sf_1 \wedge sf_3$,

2. $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_3) \wedge sf_1$,

3. $(s,\Phi,\pi) \models (\neg\mathscr{E}\mathscr{P}sf_1) \wedge sf_3$,

4. $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_3$.

To prove $(s,\Phi,\pi) \models sf_1 \lhd (sf_2 \wedge sf_3)$, it is needed to show one of the following is true:

1. $(s,\Phi,\pi) \models sf_1 \wedge (sf_2 \wedge sf_3)$,

2. $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge (sf_2 \wedge sf_3)) \wedge sf_1$,

3. $(s,\Phi,\pi) \models (\neg\mathscr{E}\mathscr{P}sf_1) \wedge (sf_2 \wedge sf_3)$,

4. $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}(sf_2 \wedge sf_3)$.

Now check the 16 combinations in $(sf_1 \lhd sf_2) \wedge (sf_1 \lhd sf_3)$ and show that in each of them, one of the 4 cases above is satisfied.

Given that $(s,\Phi,\pi) \models sf_1 \wedge sf_2$, if $(s,\Phi,\pi) \models sf_1 \wedge sf_3$, $(s,\Phi,\pi) \models sf_1 \wedge (sf_2 \wedge sf_3)$. If $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_3) \wedge sf_1$, $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge (sf_2 \wedge sf_3)) \wedge sf_1$.

Given that $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_2) \wedge sf_1$, in all cases, it is known have $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge (sf_2 \wedge sf_3)) \wedge sf_1$.

Given that $(s,\Phi,\pi) \models (\neg\mathscr{E}\mathscr{P}sf_1) \wedge sf_2$, if $(s,\Phi,\pi) \models sf_1 \wedge sf_3$, $(s,\Phi,\pi) \models sf_1 \wedge (sf_2 \wedge sf_3)$. If $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_3) \wedge sf_1$, $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}(sf_1 \wedge$

$(sf_2 \wedge sf_3)) \wedge sf_1$. If $(s,\Phi,\pi) \models (\neg\mathscr{E}\mathscr{P}sf_1) \wedge sf_3$, $(s,\Phi,\pi) \models (\neg\mathscr{E}\mathscr{P}sf_1) \wedge (sf_2 \wedge sf_3)$. If $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_3$, $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}(sf_2 \wedge sf_3)$.

Given that $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_2$, in any cases, $(s,\Phi,\pi) \models \neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}(sf_2 \wedge sf_3)$ is true.

Thus it is known that $(s,\Phi,\pi) \models (sf_1 \lhd sf_2) \wedge (sf_1 \lhd sf_3)$ only if $(s,\Phi,\pi) \models sf_1 \lhd (sf_2 \wedge sf_3)$. Similarly, it can be proved that $(s,\Phi,\pi) \models (sf_1 \lhd sf_3) \wedge (sf_2 \lhd sf_3)$ only if $(s,\Phi,\pi) \models (sf_1 \wedge sf_2) \lhd sf_3$. □

Proposition 19. *Let Φ be a transition graph, s be a state in Φ. Let sf_1 be a state formula in Pref-π-CTL*. $(s,\Phi,\pi) \models (sf_1 \lhd \top)$ iff $(s,\Phi,\pi) \models (\bot \lhd sf_1)$ iff $(s,\Phi,\pi) \models sf_1 \vee \neg\mathscr{E}\mathscr{P}sf_1$.*

Proof. The proof is based on the semantics of Pref-π-CTL*.

$(s,\Phi,\pi) \models (sf_1 \lhd \top)$ iff $(s,\Phi,\pi) \models (sf_1 \wedge \top) \vee ((\neg\mathscr{E}\mathscr{P}(sf_1 \wedge \top)) \wedge sf_1) \vee ((\neg\mathscr{E}\mathscr{P}sf_1) \wedge \top) \vee (\neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}\top)$ iff $(s,\Phi,\pi) \models sf_1 \vee \neg\mathscr{E}\mathscr{P}sf_1$.

Similarly, $(s,\Phi,\pi) \models (\bot \lhd sf_1)$ iff $(s,\Phi,\pi) \models (\bot \wedge sf_1) \vee ((\neg\mathscr{E}\mathscr{P}(\bot \wedge sf_1)) \wedge \bot) \vee ((\neg\mathscr{E}\mathscr{P}\bot) \wedge sf_1) \vee (\neg\mathscr{E}\mathscr{P}\bot \wedge \neg\mathscr{E}\mathscr{P}sf_1)$ iff $(s,\Phi,\pi) \models sf_1 \vee \neg\mathscr{E}\mathscr{P}sf_1$.

Thus $(s,\Phi,\pi) \models (sf_1 \lhd \top)$ iff $(s,\Phi,\pi) \models (\bot \lhd sf_1)$. □

Given the transition system and the initial state, each formula represents the set of states or paths satisfying it. Let the transition graph be Φ, an initial state be s, the set of policies satisfying f_1 is denoted as $P(\Phi, s, f_1)$.

Proposition 20. *Let sf_1 and sf_2 be two state formulas in Pref-π-CTL*.*

1. *If $sf_2 \models sf_1$, it is known that $(s,\Phi,\pi) \models (sf_1 \lhd sf_2)$ iff $(s,\Phi,\pi) \models sf_2 \vee ((\neg\mathscr{E}\mathscr{P}sf_2) \wedge sf_1) \vee \neg\mathscr{E}\mathscr{P}sf_1$.*

2. *If $sf_2 \models \neg sf_1$, or $sf_1 \models \neg sf_2$, it is known that $(s,\Phi,\pi) \models sf_1 \lhd sf_2$ iff $(s,\Phi,\pi) \models sf_1 \vee ((\neg\mathscr{E}\mathscr{P}sf_1) \wedge sf_2) \vee (\neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_2)$.*

3. $(s,\Phi,\pi) \models (sf_1 \lhd sf_2) \wedge (sf_2 \lhd sf_1)$ *iff* $(s,\Phi,\pi) \models (sf_1 \vee \neg\mathscr{E}\mathscr{P}sf_1) \wedge (sf_2 \vee \neg\mathscr{E}\mathscr{P}sf_2)$.

Proof. Each of them is proved based on semantics of Pref-π-CTL*:

1. Given that $sf_2 \models sf_1$, it is known that $(s,\Phi,\pi) \models (sf_1 \lhd sf_2)$ iff $(s,\Phi,\pi) \models (sf_1 \wedge sf_2) \vee ((\neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_2)) \wedge sf_1) \vee ((\neg\mathscr{E}\mathscr{P}sf_1) \wedge sf_2) \vee (\neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_2)$ iff $(s,\Phi,\pi) \models sf_2 \vee ((\neg\mathscr{E}\mathscr{P}sf_2) \wedge sf_1) \vee ((\neg\mathscr{E}\mathscr{P}sf_1) \wedge sf_2) \vee \neg\mathscr{E}\mathscr{P}sf_1$ iff $(s,\Phi,\pi) \models sf_2 \vee ((\neg\mathscr{E}\mathscr{P}sf_2) \wedge sf_1) \vee \neg\mathscr{E}\mathscr{P}sf_1$.

2. Given that $sf_2 \models \neg sf_1$, or $sf_1 \models \neg sf_2$, it is known that $(s,\Phi,\pi) \models (sf_1 \lhd sf_2)$ iff $(s,\Phi,\pi) \models (sf_1 \wedge sf_2) \vee ((\neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_2)) \wedge sf_1) \vee ((\neg\mathscr{E}\mathscr{P}sf_1) \wedge sf_2) \vee (\neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_2)$ iff $(s,\Phi,\pi) \models \bot \vee ((\neg\mathscr{E}\mathscr{P}\bot) \wedge sf_1) \vee ((\neg\mathscr{E}\mathscr{P}sf_1) \wedge sf_2) \vee (\neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_2)$ iff $(s,\Phi,\pi) \models sf_1 \vee ((\neg\mathscr{E}\mathscr{P}sf_1) \wedge sf_2) \vee (\neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_2)$.

3. $(s,\Phi,\pi) \models (sf_1 \lhd sf_2) \wedge (sf_2 \lhd sf_1)$ iff $(sf_1 \wedge sf_2) \vee (\neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_2) \vee ((\neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_2) \wedge sf_1) \vee (\neg\mathscr{E}\mathscr{P}sf_1 \wedge sf_2)) \wedge ((\neg\mathscr{E}\mathscr{P}(sf_1 \wedge sf_2) \wedge sf_2) \vee (\neg\mathscr{E}\mathscr{P}sf_2 \wedge sf_1))$ iff $(sf_1 \wedge sf_2) \vee (\neg\mathscr{E}\mathscr{P}sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_2) \vee (sf_1 \wedge \neg\mathscr{E}\mathscr{P}sf_1) \vee (sf_2 \wedge \neg\mathscr{E}\mathscr{P}sf_2)$ iff $(s,\Phi,\pi) \models (sf_1 \vee \neg\mathscr{E}\mathscr{P}sf_1) \wedge (sf_2 \vee \neg\mathscr{E}\mathscr{P}sf_2)$.

□

These properties of Pref-π-CTL* help users in simplifying a Pref-π-CTL* program. Consider the following example.

Example 24. *Given the transition graph as in the example in Section 5.1, now check whether the policy* $\pi = \{(s_1,a_1),(s_2,a_5),(s_3,a_4)\}$ *satisfies Pref-*π*-CTL** *goal* $(\mathsf{E}_{pol}\Diamond p \wedge \mathsf{E}_{pol}\Box\neg p) \lhd \mathsf{A}_{pol}\Diamond p$. *Let* sf_1, sf_2 *and* sf_3 *be* $\mathsf{E}_{pol}\Diamond p$, $\mathsf{E}_{pol}\Box\neg p$ *and* $\mathsf{A}_{pol}\Diamond p$ *respectively. The goal is* $(sf_1 \wedge sf_2) \lhd sf_3$.

According to Proposition 18, the policy satisfies the goal only if it satisfies the goal $(sf_1 \lhd sf_3) \wedge (sf_2 \lhd sf_3)$. *According to Proposition 20, as* $sf_3 \models sf_1$ *and* $sf_3 \models \neg sf_2$, *the goal is equivalent to P-CTL** *goal* $(sf_3 \vee ((\neg \mathscr{E}\mathscr{P} sf_3) \wedge sf_1) \vee \neg \mathscr{E}\mathscr{P} sf_1) \wedge (sf_2 \vee ((\neg \mathscr{E}\mathscr{P} sf_2) \wedge sf_3) \vee (\neg \mathscr{E}\mathscr{P} sf_2 \wedge \neg \mathscr{E}\mathscr{P} sf_2))$.

As policy π *satisfies* sf_1, *and* sf_2, *and there is no policy in the domain satisfies* sf_3. *The goal above is satisfied. Thus* π *satisfies the Pref-*π*-CTL** *goal* $(\mathsf{E}_{pol}\Diamond p \wedge \mathsf{E}_{pol}\Box\neg p) \lhd \mathsf{A}_{pol}\Diamond p$.

Let sf_1, sf_2, and sf_3 be state formulas. It is noticed that $(s,\Phi,\pi) \models sf_1 \lhd (sf_2 \lhd sf_3)$ does not imply that $(s,\Phi,\pi) \models (sf_1 \lhd sf_2) \lhd sf_3$. For example, given a state s in system Φ. Suppose there are only two policies π_1 and π_2 starting from state s in Φ. Policy π_1 satisfies state formulas sf_1 but not sf_2 and sf_3. Policy π_2 satisfies state formulas sf_1, sf_2 but not sf_3. Thus $(s,\Phi,\pi_1) \models sf_1 \lhd (sf_2 \lhd sf_3)$ but $(s,\Phi,\pi_1) \not\models (sf_1 \lhd sf_2) \lhd sf_3$.

5.4 Compare Pref-π-CTL* with Related Languages

As mentioned in Section 5.1, language Pref-π-CTL* is related to other goal specification languages in non-deterministic domain. Now compare Pref-π-CTL* with π-CTL* and P-CTL*.

Compare Pref-π*-CTL** *with other Goal Specification Languages in Non-deterministic Domain*

Goal specification language Pref-π-CTL* is based on language π-CTL*. According to the definition on one language is Syntax-advanced than the other language in Chapter 3, P-CTL* is syntax-advanced than Pref-π-CTL*, and Pref-π-CTL* is syntax-advanced than π-CTL*.

Further, the following proposition on the set of goals expressed in the languages is true.

Proposition 21. *Given a goal that is a mapping from states and transition graphs to set of set of trajectories, it is known that:*

- *A goal expressed in π-CTL* can be expressed in Pref-π-CTL*;*
- *A goal expressed in Pref-π-CTL* can be expressed in P-CTL*.*

Proof. Let $\varphi(s,\Phi)^{\pi-CTL*}$ be $\varphi(s,\Phi)$ in language π-CTL*. Let $\varphi(s,\Phi)^{Pref-\pi-CTL*}$ be $\varphi(s,\Phi)$ in language Pref-π-CTL*. Suppose a goal g can be expressed in π-CTL* as φ. It is known that $g(s,\Phi) = \varphi(s,\Phi)^{\pi-CTL*}$ for any state s and transition graph Φ. Now prove that $\varphi(s,\Phi)^{\pi-CTL*} = \varphi(s,\Phi)^{Pref-\pi-CTL*}$. As Pref-$\pi$-CTL* is syntax-advanced than π-CTL*, it is known that the set of policies satisfying φ in these two languages are the same, and policies defined in these two languages are the same. As $\varphi(s,\Phi)$ is defined as
$\{\pi_\sigma : (s,\Phi,\pi) \models \varphi$ and π_σ is the set of trajectories that are consistent with policy $\pi\}$, it is known that $\varphi(s,\Phi)^{\pi-CTL*} = \varphi(s,\Phi)^{Pref-\pi-CTL*}$. Thus a goal expressed in π-CTL* can be expressed in Pref-π-CTL*.

Similarly, it can be proved that a goal expressed in Pref-π-CTL* can be expressed in P-CTL*. □

On the other hand, there are some goals in Pref-π-CTL* that cannot be expressed in π-CTL*. This is implied by Proposition 4 in Chapter 3. It is to show that $strong(p) \lhd strongCyclic(p)$ cannot be expressed in pi-CTL*.

Compare Pref-π-CTL* with other Languages with Preferences

Now compare Pref-π-CTL* with other goal specification languages with preferences. It is illustrated in the following example in a deterministic domain.

Example 25. *Tom needs to go to school this morning to attend a semina. He prefers to have breakfast before leaving for school. But if he got up late or have other things to do, he might have to skip the breakfast.*

The goal is presented as $\Diamond(breakfast \wedge \Diamond atSemina) \lhd \Diamond atSemina$*, which states that is there is a plan for Tom to have breakfast before attending the semina, take that plan. Otherwise, try to attend the semina. If there is no plan for Tom to attend the semina, Tom may skip the semina as well. It does not matter whether Tom will have breakfast or not in this case.*

Also note that this goal is different from $\Diamond((breakfast \lhd \top) \wedge \Diamond atSemina)$*, which is equivalent to P-CTL** *formula* $\Diamond((breakfast \vee \neg \mathscr{E}\mathscr{P} breakfast) \wedge \Diamond atSemina)$ *according to Proposition 19.*

Now check how this goal is represented in other goal specification languages with preferences. In PDDL3, attending the semina is considered as a hard goal while having breakfast is considered as a soft goal. Among the plans, as long as Tom has attended the semina, the goal is considered as satisfied, even though Tom may not have had the breakfast.

In $\mathscr{P}\mathscr{P}$*, the goal is represented as* $\Diamond(breakfast \wedge \Diamond atSemina) \lhd \Diamond atSemina$*. Plans satisfying this formula are the same in* $\mathscr{P}\mathscr{P}$ *and in Pref-*π*-CTL***.*

Language $\mathscr{P}\mathscr{P}$ [SP06] is defined for deterministic domains. Now extend it to non-deterministic domains first before comparing with Pref-π-CTL*.

In $\mathscr{P}\mathscr{P}$, an ordering between trajectories w.r.t. single desire (or goal) is defined. Now define an ordering between policies w.r.t. single desire. Note that in a non-deterministic domain, a policy leads to a set of trajectories.

Definition 47 (Ordering between Policies w.r.t. Single Desire)**.** *Let* φ *be a basic desire formula and let* α *and* β *be two policies. Policy* α *is preferred to policy* β *in transition system* Φ *with initial state s if* $(s,\Phi,\alpha) \models \varphi$ *and* $(s,\Phi,\beta) \not\models \beta$*.*

Policy α *and* β *are indistinguishable in transition system* Φ *with initial state s if one of the following two cases occurs: (i)* $(s,\Phi,\alpha) \models \varphi$ *and* $(s,\Phi,\beta) \models \varphi$*, or (ii)* $(s,\Phi,\alpha) \not\models \varphi$ *and* $(s,\Phi,\beta) \not\models \varphi$*.*

With this definition, $\mathscr{P}\mathscr{P}$ can be extended to non-deterministic domains. The preference relation defined in $\mathscr{P}\mathscr{P}$ can be captured by quantifying over policies.

As discussed above, nested comparisons of the formulas are allowed. A formula $\Diamond(p \lhd q)$ or $\mathsf{A}_{pol}\Box(strong(p) \lhd strongCyclic(p))$ in Pref-π-CTL*cannot be captured in $\mathscr{P}\mathscr{P}$. Besides, comparing to language $\mathscr{P}\mathscr{P}$, Pref-π-CTL*allows different preference relations under different conditions. For example, suppose the goal is expressed as: $(c_1 \Rightarrow (f_1 \lhd f_2)) \wedge (c_2 \Rightarrow (f_2 \lhd f_1))$.

Language $\mathscr{P}\mathscr{P}$ and Pref-π-CTL* are different in defining semantics of formulas such as $f_1 \lhd f_2 \lhd f_2$. This formula is undefined in Pref-π-CTL*. Meanwhile, $\mathscr{P}\mathscr{P}$, formulas $(f_1 \lhd f_2) \lhd f_3$ and $f_1 \lhd f_2 \lhd f_3$ have different semantics.

5.5 Discussion

Point-wise Preference

The preferences relation defined in the language is a rule based preferences relation, meaning that when checking the $\lhd$ relations, a policy either satisfies a formula or does not satisfy the formula. There is no definition on partial satisfaction of a formula. It is also unheard that a policy is more "closer" in satisfying a goal than other policies.

However, there are cases where the preference relations are defined on "subgoals" that contradict with each other. In some cases, even though the most preferred goal cannot be satisfied by a policy, users may want partial satisfaction of the goal.

The following example illustrates the case that point-wise preferences might be needed.

Example 26. *Joe always do exercise before dinner. However, Joe get a phone call from a friend to meet him tomorrow before dinner. Joe have to skip the exercise tomorrow but will continue the exercise in the following days.*

The initial goal is represented as $\Box$*exercise. A second rule is appended as* $\bigcirc(\neg exercise \wedge meetFriend)$*. Users might want a revised goal as exercise* $\wedge \bigcirc(\neg exercise \wedge meetFriend) \wedge \bigcirc \bigcirc \Box exercise$*. This revisions cannot be done in Pref-*π*-CTL***. How to define a goal specification language to handle this is still a challenging issue.*

5.6 Summary

This chapter proposed a goal specification language with preference for goal specifications in non-deterministic domain. The language is based on π-CTL*. A binary connective $\lhd$ is introduced to compare state formulas. Comparing to other goal specification languages with preferences, Pref-π-CTL* is the only language for non-deterministic domains. Besides, by treating the $\lhd$ operator the same way as other operators, language Pref-π-CTL* has some interesting properties such as allowing nested preferences and dynamic preferences.

In terms of future work, an interesting topic is to utilize the preference relation in other temporal logics, especially the non-monotonic goal specification languages. Another interesting topic is to make use of the goal specification languages with preferences in defining non-monotonic goal specification languages. Defining goal specification languages that can deal with point-wise preferences is also an interesting topic.

Chapter 6

PLANNING WITH GOALS SPECIFIED IN TEMPORAL LOGICS Π-CTL* AND PREF-Π-CTL*

Given a goal specified in π-CTL* or P-CTL*, planning and plan checking problems are more difficult that traditional planning problems. For example, planning problems with goals in π-CTL* is EXPTIME-hard. However, for specific subsets of goals specified in π-CTL*, Polynomial time algorithms can be found by using the same approach as Baral et. al. proposed for *k*-maintainability problems. The method first encodes the problem in reverse Horn SAT, and then translates it to Horn SAT. Finally, a genuine algorithm is developed by simulating the way of solving the Horn SAT program. This chapter shows that this approach of obtaining polynomial time algorithms for problem solving can be fruitfully applied to finding plans for various π-CTL* goals including weak, strong, strong cyclic plans and a few other π-CTL* goals. Some interesting properties of these planning problems can be found by comparing their reverse Horn SAT encodings. Further, a program solving a particular Pref-π-CTL* program is given.

6.1 Introduction

In recent years, one of the approaches that is used in finding solutions to AI problems is to find "models" of a logical encoding of the problem. Examples of this include finding planning via satisfiability encoding [KS92] or logic programming encodings with answer set semantics [GL91]. The later is now referred to as answer set programming. But in most of these cases, problems solved are in the complexity class NP-complete or beyond. One outlier is the work [BEBN08] which takes advantage of the lower complexity results about specific logic programming and SAT sub-classes to come up with a polynomial-time algorithm for finding maintenance

policies.

In that paper, the authors first give a propositional reverse Horn encoding of the problem and show that the models of the encoding correspond to desired agent policies. They then give a transformation of that encoding to a propositional Horn encoding. The fix-point iteration approach to compute models of Horn theories, which is feasible in linear time, is then exploited to develop a genuine polynomial-time algorithm for finding agent policies. If one were to view the logical encoding as a specification, then the above mentioned approach can be considered as a systematic way to develop algorithms from specifications. The original software engineers dreamed of finding ways where algorithms are obtained from problem specification in a systematic way. This dream is partly come true now.

In recent years, there have been some important work on planning in non-deterministic domains [DLPT02, CPRT03]. In particular, in [CPRT03] the notions of strong planning, weak planning, and strong cyclic planning were introduced, and algorithms for finding such plans were presented. In Chapter 3 of this dissertation, temporal logics are extended to better capture goal specifications in a non-deterministic domain. In particular, languages π-CTL* and P-CTL* are proposed. It is noted that strong planning, weak planning, and strong cyclic planning problems can be encoded in π-CTL* as $\mathsf{E}_{pol}\Diamond p$, $\mathsf{A}_{pol}\Diamond p$, and $\mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)$ respectively. This chapter explores the possibilities of making use of the approach in [BEBN08] in solving strong planning, weak planning, strong cyclic planning and a few other π-CTL* goals. Encodings inspired by the encoding in [BEBN08] are developed, leading to polynomial time algorithms for finding plans. Further, the relations of these problems are studied by comparing their encodings.

The approach is generalized to obtain polynomial time algorithms for a few other π-CTL* goals in a non-deterministic domain.

Contributions of this chapter are as follows:

- This chapter illustrates the novel algorithm design approach of [BEBN08] to systematically develop an algorithm from a logical specification. Passing through Horn SAT specifications, new polynomial time algorithms for weak, strong, and strong cyclic planning are developed; thus shedding additional insights about these notions. As part of that, the encoding for finding weak plans is a subset of the encoding for finding strong cyclic plans.

- Show how strong cyclic plans can be declaratively generated with answer set programming at an abstract level. Discuss how particular properties of the encodings and features of answer set solvers can be exploited for computing (most) preferred plans among alternative candidate plans. In particular, based on the encoding, maximal plans and least defined plans can be found in polynomial time.

- How this approach can lead to algorithms for other kind of goals in non-deterministic domains is discussed.

- Complexity results about weak, strong and strong cyclic planning are given. (No such results appear in previous papers.)

6.2 Background: Strong, Weak, and Strong Cyclic Plans in Non-deterministic Domains

This chapter start with recalling the notions of weak, strong, and strong cyclic plans from [CPRT03]. Such plans manifest in non-deterministic domains. In such domains, plans map states to actions or to sets of actions. A weak plan to achieve p is a plan that says that at least one of the paths (based on following that plan) leads to p. A strong plan to achieve p is a plan that says that all paths (based on following that plan) would lead to p. A strong cyclic plan to achieve p is a plan that says all along the path (based on following that plan) there is at least one of the paths

(based on following that plan) that would lead to p. These goals are expressed as $\mathsf{E}_{pol}\Diamond p$, $\mathsf{A}_{pol}\Diamond p$, and $\mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)$ in language π-CTL* [BZ04], respectively; where $\Diamond$ means eventually, $\Box$ means always, E_{pol} means exists a path following the plan under consideration, and A_{pol} means all paths following the plan under consideration.

Now give the formal definitions.

Definition 48 (Planning problem)**.** *Let $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$ be a system. A planning problem for $\mathscr{D}$ is a triple $\langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$ where $\mathscr{I} \subseteq \mathscr{S}$, and $\mathscr{G} \subseteq \mathscr{S}$.*

Definition 49 (Execution structure)**.** *Let π be a control policy, or a plan of a planning problem $\langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$ where $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$. The execution structure induced by π from the set of initial states $\mathscr{I} \subseteq \mathscr{S}$ is a tuple $K = \langle Q, T \rangle$ with $Q \subseteq \mathscr{S}$ and $T \subseteq \mathscr{S} \times \mathscr{S}$ inductively defined as follows:*

1. *If $s \in \mathscr{I}$, then $s \in Q$, and*
2. *If $s \in Q$, action $a \in \pi(s)$, and $s' \in \Phi(s,a)$, then $s' \in Q$ and $(s,s') \in T$.*

A state $s \in Q$ is a terminal state of K if there is no $s' \in Q$, $s' \neq s$, such that $(s,s') \in T$.

In the following, it is assumed that there is always an action *nop* in each state s_i, such that $\Phi(s_i, nop) = \{s_i\}$, thus the planning problem $\langle \mathscr{S}, \mathscr{A}, \Phi, poss \rangle$ can be simplified as $\langle \mathscr{S}, \mathscr{A}, \Phi \rangle$.

A state $s_2 \in Q$ is reachable from state $s_1 \in Q$ if there is a path from s_1 to s_2 in T.

Definition 50 (Plans with respect to a planning problem)**.** *Let $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$ be a planning domain, $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$ be a planning problem, π be a plan in $\mathscr{D}$. Let $K = \langle Q, T \rangle$ be the execution structure induced by π from $\mathscr{I}$.*

1. *π is a weak plan with respect to P iff for any state in $\mathscr{I}$, some terminal state in $\mathscr{G}$ is reachable from the state.*

2. *π is a strong plan with respect to P iff K is acyclic and all the terminal states of K are in $\mathscr{G}$.*

3. *π is a strong cyclic plan with respect to P iff from any state in Q some terminal state is reachable and all the terminal states of K are in $\mathscr{G}$.*

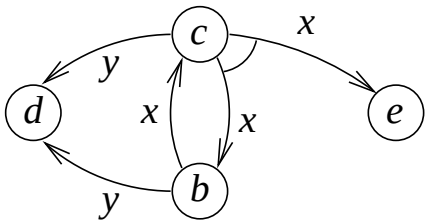

Figure 6.1: Transition diagram of the planning domain $\mathscr{D}$

Example 27. *Consider a planning domain $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$. Let $\mathscr{S} = \{b,c,d,e\}$, $\mathscr{A} = \{x,y\}$, and the transition function Φ as in Figure 6.1. Then, poss$(b) = \{x,y\}$ while poss$(e) = \emptyset$. For the planning problem $\langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$ where $\mathscr{I} = \{b\}$ and $\mathscr{G} = \{e\}$, the mapping π such that $\pi(c) = x$ and $\pi(b) = x$, is a strong cyclic plan. Its execution structure is $K = \{\{b,c,e\}, \{(b,c),\ (c,b),\ (c,e)\}\}$. In this planning problem, no strong plan exists, while π is also a weak plan.*

6.3 Finding Strong Cyclic Plans

This section uses the approach in [BEBN08] to develop algorithms that construct strong cyclic plans. To start with, a propositional SAT encoding of a planning problem is given. It is shown that the models of this theory encode strong cyclic plans, if one exists, and vice versa.

SAT Encoding S-Cyclic(P)

In the SAT encoding, for each state s and action a, propositions s_i and s_a_i are used, where $i \geq 0$ is an integer. Intuitively, s_i will mean that there is a path from s to $\mathscr{G}$, following T of the execution structure $K = \langle Q, T \rangle$, of length at most i. Similarly, s_a_i will intuitively mean that there is a path from s to $\mathscr{G}$ of length at most i, following T of the execution structure $K = \langle Q, T \rangle$, and with a as its first action. Let an upper bound $max = |\mathscr{S}| - 1$ for i, depending on the number of states in $\mathscr{S}$; if there is no path of length at most max, there is no path at all.

[SAT encoding of strong cyclic planning: *S-Cyclic*] Suppose a planning problem $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$ is given where $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$. Let $max = |\mathscr{S}| - 1$. P is translated into a SAT encoding *S-Cyclic*(P) as follows:

(0) for all $s \in \mathscr{S}$ and i, $0 < i \leq max$: $s_{i-1} \Rightarrow s_i$

(1) for every state $s \in \mathscr{S} \backslash \mathscr{G}$, and for all i, $0 < i \leq max$: $s_i \Rightarrow \bigvee_{a \in poss(s)} s\_a_i$

(2) for every states s, $s' \in \mathscr{S}$ such that $s' \in \Phi(s, a)$ for some action a: $s\_a_{max} \Rightarrow s'_{max}$

(3) for every state $s \in \mathscr{S}$, action $a \in poss(s)$, and for all i, $0 < i \leq max$: $s\_a_i \Rightarrow \bigvee_{s' \in \Phi(s,a)} s'_{i-1}$

(4) for every state $s \in \mathscr{S}$, action $a \in poss(s)$, and $1 < i \leq max$: $s\_a_{i-1} \Rightarrow s\_a_i$

(5) for $s \in \mathscr{I}$: s_{max}

(6) for $s \in \mathscr{S} \setminus \mathscr{G}$: $\neg s_0$

The encoding in Algorithm 6.3 uses the step numbers in [BEBN08] so as to reflect the closeness between this encoding and the encoding sat' of [BEBN08] for k-maintainability. In case of sat', the number k is part of the input. The clauses in (0), (5) and (6) are the same as in $sat'(I)$ with one exception; instead of k, $max-1$ is used. The clauses in (1) and (4) are also very similar to the corresponding clauses in $sat'(I)$. The main difference are the genuine clauses in (2) and (3), and that (2) of $sat'(I)$ is missing in this encoding (because there are no exogenous actions here).

The intuition behind this encoding is as follows. The clauses in (0) state that if there is a path from s to $\mathscr{G}$ of length at most i-1, then there is a path of length at most i. The clauses in (4) make a similar statement for paths with first action a. The clauses in (1) state that if there is a path from s to $\mathscr{G}$ of length at most i, then there must exist an action a which is the first action of such a path. The clauses in (2) state that for any state s, there is a path from s to $\mathscr{G}$ of length at most *max* with a as its first action only if from every state $s' \in \Phi(s,a)$ a path to $\mathscr{G}$ of length at most *max* exists. This takes into account the possibility that s may be in the closure Q of the execution structure $\langle Q,T\rangle$. This rule makes sure that in the resulted plan, for any state reachable from the initial state by following the plan, there is a path to a state in $\mathscr{G}$. The clauses in (3) state that a path from s to $\mathscr{G}$ of length at most i with a as its first action exists only if there is a path from some state $s' \in \Phi(s,a)$ to $\mathscr{G}$ of length at most i-1. The clauses in (5) state that every initial state must have a path of length at most *max*. Finally, the clauses in (6) exclude paths of length zero for non-goal states.

Strong cyclic plans with respect to P and the models of *S-Cyclic*(P) are formally connected as follows.

Lemma 3. *For any state t, a model of the program M, if $M \models t_i$ where $0 \leq i < max$, then there is a path in T_M from t to a final state in $\mathscr{G}$ and the length of the path is not larger than i, where T_M is in the execution structure corresponding to M.*

Proof. It is proved by the induction on i.

In the base case, $M \models t_0$. From the clauses in (6), it is known that t has to be in $\mathscr{G}$. According to the definition of *S-Cyclic*, there is a path in T_M of length 0 from state t to a final state in $\mathscr{G}$. Thus the statement holds for the base case.

For the induction step, suppose for all states s where $M \models s_j$, for $j < i$, there is a path in T_M from s to a final state in $\mathscr{G}$ with the length no larger than j. For a state

s', if $M \models s'_i$ and $M \models s'_{i-1}$, there are two different cases. If $s \in \mathscr{G}$, the induction step is proved. On the other hand, if $s \in \mathscr{S} \backslash \mathscr{G}$. Since M must satisfy the clauses in (1), there is an action s_a such that $M \models s\_a_i$. Since M must satisfy the clauses in (3), there is a state $s' \in \Phi(s,a)$ with $M \models s'_{i-1}$. Based on the induction, there is a path no larger than $i-1$ steps from s' to a terminal goal state. By taking action s_a, there is a path from state s to a terminal goal state and the length of such a path is less than or equals to i. The induction step is proved. □

Proposition 22. *1. P has a strong cyclic plan iff S-Cyclic(P) is satisfiable;*

2. For any model M of S-Cyclic(P), the partial function $\pi_M : \mathscr{S} \to 2^{\mathscr{A}}$ defined by $\pi_M(s) = \{a \mid M \models s\_a_j, j = \min_i M \models s_i\}$ on all states $s \in \mathscr{S} \setminus \mathscr{G}$ such that $M \models s_i$ for some i, is a strong cyclic plan of P.

Proof. First prove (1). Suppose P has a strong cyclic plan π.

Let $\mathscr{T}(\pi)$ be the set of terminal states of policy π. A policy π' is defined such that $\pi'(s) = \pi(s)$ for all $s \notin \mathscr{G}$; $\pi'(s)$ are not defined for $s \in \mathscr{G}$. It is clear that $\mathscr{T}(\pi) \subseteq \mathscr{T}(\pi') \subseteq \mathscr{G}$ and π' is also a strong cyclic plan of P. Denote the execution structure induced by π' by $\langle Q_{\pi'}, T_{\pi'} \rangle$.

According to the definition of strong cyclic plan, for any state s in $Q_{\pi'}$, there is a path (via $T_{\pi'}$) from s to a state in $\mathscr{T}(\pi')$. Consider states in $Q_{\pi'}$. Let $d(s,\mathscr{G})$ be the length of one of the shortest path (via $T_{\pi'}$) from s to any state in $\mathscr{G}$. For each state $s \in Q_{\pi'}$, if $d(s,\mathscr{G})$ is n, then define $s_0, \cdots, s_{n-1}$ to be false and $s_n, \cdots, s_{max-1}$ to be true. For each action a such that $\pi'(s) = a$ and $d(s,\mathscr{G}) = n$, $s\_a_1, \cdots, s\_a_{n-1}$ are defined to be false and $s\_a_n, \cdots, s\_a_{max-1}$ to be true. All other s_i and s_a_i atoms are assigned false. Denote this propositional interpretation by N. Now argue that N satisfies all clauses in *S-Cyclic*(P). By construction of N all clauses in (0) are satisfied by N. All clauses in (1) are satisfied by N because when their left hand side is true, that means $s \in Q_{\pi'}$. Since $s \notin \mathscr{G}$, and $\mathscr{T}(\pi') \subseteq \mathscr{G}$, s cannot be a terminal

state, and thus $\pi'(s)$ must be defined. Then by the construction of N the right hand side of (1) must be satisfied by N. For the clauses in (2) the left hand side is satisfied by N only when $s \in Q_{\pi'}$, and $a \in \pi'(s)$. By definition of $Q_{\pi'}$, all $s' \in \Phi(s,a)$ are going to be in $Q_{\pi'}$. Thus N must satisfy the right hand side of the clause whose left hand side it satisfies. Consider the clauses in (3). If N satisfies its left hand side then there must be a path from s to $\mathscr{G}$ via (via $T_{\pi'}$). Let the length of the shortest such path be m, and a is the first action in one such path. Since this is one of the shortest paths, one of the states $s' \in \Phi(s,a)$ must have a path of length m-1 to $\mathscr{G}$. Thus the right hand side of the corresponding clause in (3) is satisfied by N. Hence, N satisfies the clauses in (3). By construction of N, it is easy to see that N satisfies the clauses in (4), (5) and (6). This proves part (1) of the proposition.

To prove part (2) of the proposition, let $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$, with $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$, and $|\mathscr{S}| = max$. Let M be a model of $S\text{-}Cyclic(P)K_M$ be as defined, and $\langle Q_M, T_M \rangle$ be the execution structure induced by K_M from $\mathscr{I}$. From the construction of K_M and since it is not defined on states in $\mathscr{G}$, it is needed to show that for any state in Q_M, there is a path from this state to a final state in $\mathscr{G}$.

Let the distance $d_{K_M}(s, \mathscr{I})$ be the length of the shortest path (via T_M) from any state in $\mathscr{I}$ to s. By using induction on $d_{K_M}(s, \mathscr{I})$, and the above lemma, that for every state in Q_M, there is a path (via T_M) from this state to a final state in $\mathscr{G}$.

The base case, $d_{K_M}(s, \mathscr{I}) = 0$, is about $s \in \mathscr{I}$. From the clauses in (5), for these states s, $M \models s_{max-1}$. Thus, by using the lemma, there is a path from s to a final state in $\mathscr{G}$. Thus the statement holds in the base case.

Now for the induction step, assume that if $d_{K_M}(s, \mathscr{I}) < d$, then there is a path in Q_M from s to a final state in $\mathscr{G}$. Now prove the case where $d_{K_M}(s', \mathscr{I}) = d$. Since $d_{K_M}(s', \mathscr{I}) = d$, there is a state s such that $d_{K_M}(s, \mathscr{I}) = d - 1$ and $s' \in \Phi(s,a)$ where $a \in K_M(s)$). But then $M \models s\_a_j$ for some j. Due to the clauses in (4) $M \models s\_a_{max-1}$. Using the clauses in (2) and the fact that $s' \in \Phi(s,a)$, it is known that $M \models s'_{max-1}$.

Thus, using the lemma, there is a path in T_M from s' to a final state in $\mathscr{G}$. For a state $s \in \mathscr{S}$, if there is a path from $\mathscr{I}$ to s, then the length of the shortest path is at most $max-1$. This implies that induction step considers all states that are reachable from $\mathscr{I}$. This concludes the induction and the proof of (2). □

The following example illustrates the encoding and its use in computing strong cyclic plans.

Example 28. *Consider the strong cyclic planning problem in Example 27. Its SAT encoding is as follows:*

Clauses (0): $b_0 \Rightarrow b_1.\ b_1 \Rightarrow b_2.\ b_2 \Rightarrow b_3.$

$c_0 \Rightarrow c_1.\ c_1 \Rightarrow c_2.\ c_2 \Rightarrow c_3.$

$d_0 \Rightarrow d_1.\ d_1 \Rightarrow d_2.\ d_2 \Rightarrow d_3.$

$e_0 \Rightarrow e_1.\ e_1 \Rightarrow e_2.\ e_2 \Rightarrow e_3.$

Clauses (1): $b_1 \Rightarrow b\_x_1 \vee b\_y_1.\ b_2 \Rightarrow b\_x_2 \vee b\_y_2.$

$b_3 \Rightarrow b\_x_3 \vee b\_y_3.\ c_1 \Rightarrow c\_x_1 \vee c\_y_1.$

$c_2 \Rightarrow c\_x_2 \vee c\_y_2.\ c_3 \Rightarrow c\_x_3 \vee c\_y_3.$

$d_1 \Rightarrow \bot.\ d_2 \Rightarrow \bot.\ d_3 \Rightarrow \bot.$

Clauses (2): $b\_x_3 \Rightarrow c_3.\ b\_y_3 \Rightarrow d_3.$

$c\_x_3 \Rightarrow e_3.\quad c\_x_3 \Rightarrow b_3.\ c\_y_3 \Rightarrow d_3.$

Clauses (3) $b\_x_1 \Rightarrow c_0.\ b\_x_2 \Rightarrow c_1.\ b\_x_3 \Rightarrow c_2.$

$b\_y_1 \Rightarrow d_0.\ b\_y_2 \Rightarrow d_1.\ b\_y_3 \Rightarrow d_2.$

$c\_x_1 \Rightarrow b_0 \vee e_0.\ c\_x_2 \Rightarrow b_1 \vee e_1.$

$c\_x_3 \Rightarrow b_2 \vee e_2.$

$c\_y_1 \Rightarrow d_0.\ c\_y_2 \Rightarrow d_1.\ c\_y_3 \Rightarrow d_2.$

Clauses (4): $b\_x_0 \Rightarrow b\_x_1.\ b\_x_1 \Rightarrow b\_x_2.\ b\_x_2 \Rightarrow b\_x_3.$

$b\_y_0 \Rightarrow b\_y_1.\ b\_y_1 \Rightarrow b\_y_2.\ b\_y_2 \Rightarrow b\_y_3.$

$c\_x_0 \Rightarrow c\_x_1.\ \ c\_x_1 \Rightarrow c\_x_2.\ \ c\_x_2 \Rightarrow c\_x_3.$

$c\_y_0 \Rightarrow c\_y_1.\ \ c\_y_1 \Rightarrow c\_y_2.\ \ c\_y_2 \Rightarrow c\_y_3.$

Clauses (5): b_3.

Clauses (6): $b_0 \Rightarrow \bot .\ \ c_0 \Rightarrow \bot .\ \ d_0 \Rightarrow \bot .$

This SAT instance is solvable, and one of its models is $M = \{b_3, c_2, c_3, e_1, e_2, e_3, b\_x_3, c\_x_2, c\_x_3\}$. *Using Proposition 22, A strong cyclic plan* π *can be constructed from M given by* $\pi(b) = \{x\}$ *and* $\pi(c) = \{x\}$.

Horn SAT Encoding

[Horn SAT encoding of strong cyclic planning] Let a planning problem $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$, where $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$. Suppose $max = |\mathscr{S}| - 1$. P is translated into a Horn encoding $\overline{S\text{-}Cyclic}(P)$:

(0) for all $s \in \mathscr{S}$ and i, $0 < i \leq max$: $\overline{s_i} \Rightarrow \overline{s_{i-1}}$

(1) for every state $s \in \mathscr{S} \backslash \mathscr{G}$, and for all i, $0 < i \leq max$: $\bigwedge_{a \in poss(s)} \overline{s\_a_i} \Rightarrow \overline{s_i}$.

(2) for every states s, $s' \in \mathscr{S}$ such that $s' \in \Phi(s,a)$ for some action a: $\overline{s'_{max}} \Rightarrow \overline{s\_a_{max}}$

(3) for every state $s \in \mathscr{S}$, action $a \in poss(s)$, and for all i, $0 < i \leq max$: $\bigwedge_{s' \in \Phi(s,a)} \overline{s'_{i-1}} \Rightarrow \overline{s\_a_i}$

(4) for every state $s \in \mathscr{S}$, action $a \in poss(s)$, and for all i, $1 < i \leq max$: $\overline{s\_a_i} \Rightarrow \overline{s\_a_{i-1}}$

(5) for $s \in \mathscr{I}$: $\overline{s_{max}} \Rightarrow \bot$

(6) for $s \in \mathscr{S} \setminus \mathscr{G}$: $\overline{s_0}$

While $S\text{-}Cyclic(P)$ is constructible in polynomial time from P, it cannot automatically be inferred that finding strong cyclic plans is polynomial, since SAT is a canonical NP-hard problem. However, a closer look at the structure of the clauses in $S\text{-}Cyclic(P)$ reveals that this instance is solvable in polynomial time. Indeed, it is a *reverse Horn* theory; i.e., after reversing the propositions, the theory is Horn.

Using propositions $\overline{s_i}$, which intuitively mean the converse of s_i, the Horn theory corresponding to *S-Cyclic*(P), denoted $\overline{\textit{S-Cyclic}}(P)$, is illustrated in Algorithm 6.3.

As computing a model of a Horn theory is a well-known polynomial problem [DG84], the following result holds.

Theorem 2. *Strong cyclic plans can be computed in polynomial time.*

Maximal Plan

An interesting aspect of the above is that, as well-known, each satisfiable Horn theory T has the least model, $M^*(T)$, which is given by the intersection of all its models. Moreover, the least model is computable in linear time, cf. [DG84]. This model not only leads to a strong cyclic plan, but also leads to a *maximal* plan, in the sense that the control is defined on a greatest set of states outside $\mathscr{G}$ among all possible strong cyclic plans for initial states $\mathscr{I}'$ and goal states $\mathscr{G}$ such that $\mathscr{I} \subseteq \mathscr{I}'$. This gives a clear picture of which other states may be added to $\mathscr{I}$ while the property of strong cyclic is preserved.

Lean Plans

On the other hand, intuitively a strong cyclic plan constructed from some maximal model of $\overline{\textit{S-Cyclic}}(P)$ with respect to the propositions $\overline{s_k}$ is undefined to a largest extent, and works merely for a smallest extension. Starting from any model of T, such a maximal model of T can be generated by trying to flip step by step all propositions $\overline{s_k}$ which are false to true, and change other propositions as needed for satisfiability. In this way, a maximal model of T on $\{\overline{s_k} \mid s \in \mathscr{S} \setminus \mathscr{G}\}$ can be generated in polynomial time, from which a "lean" control can also be extracted in polynomial time.

Genuine Procedural Algorithm

From the encoding to Horn SAT above, a direct algorithm Strong Cyclic Plan can be distilled to construct a strong cyclic plan, if one exists. It mimics the steps which a SAT solver might take in order to solve $\overline{S\text{-}Cyclic(P)}$. For each state $s \in \mathscr{S}$ and action $a \in poss(s)$, counters $c[s]$ and $c[s\_a]$ ranging over $\{-1, 0, \cdots, max\}$ and $\{0, 1, \cdots, max\}$, respectively, are used. Intuitively, $c[s] = i$ represents that so far $\overline{s_0}$, $\overline{s_1}$, $\cdots$, $\overline{s_i}$ are assigned true; in particular, $i = -1$ represents that no $\overline{s_i}$ is assigned true yet. Similarly, $c[s\_a] = i$ represents that so far $\overline{s\_a_1}$, $\overline{s\_a_2}$, $\cdots$, $\overline{s\_a_i}$ are assigned true. In particular, $c[s\_a_i] = 0$ means that no $\overline{s\_a_i}$ is assigned true yet.

Based on Proposition 22 and the fact that Strong Cyclic Plan mimics the computation of the least model of $\overline{S\text{-}Cyclic}(P)$ in Algorithm 6.3, the following proposition is true.

Proposition 23. *Algorithm 6.3 on Strong Cyclic Plan finds a strong cyclic plan in a planning problem. Furthermore, for every input $\mathscr{D}$ and P, it terminates in polynomial time.*

Remark that algorithm Strong Cyclic Plan can be made more efficient by pruning in a linear time preprocessing all states which are not on a path between some states $s \in \mathscr{I}$ and $s' \in \mathscr{G}$.

A more detailed account of the complexity of Strong Cyclic Plan and possible improvements are given in Section 6.6.

Strong Cyclic Planning Using an Answer Set Solver

This section, shows how computing strong cyclic plans can be encoded as a logic program, based on the results of the previous section. More precisely, an encoding to non-monotonic logic programs under the Answer Set semantics [GL91] is described, which can be executed on one of the available Answer Set solvers such

[Strong cyclic plan]

Input: A planning domain $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$, and a planning problem $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$.

Output: A strong cyclic plan of P if such plan exists. Otherwise, output that no such plan exists.

(Step 1) Initialization:

(i) For every $s \in \mathscr{G}$, set $c[s] := -1$.

(ii) For every $s \in \mathscr{S} \backslash \mathscr{G}$, if $poss(s) = \emptyset$ then set $c[s] := max$ else set $c[s] := 0$.

(iii) For each $s \in \mathscr{S} \backslash \mathscr{G}$ and $a \in poss(s)$, set $c[s\_a]$:=0.

(Step 2) Repeat until no change or $c[s] = max$ for some $s \in \mathscr{I}$:

(i) For every state $s \in \mathscr{S} \backslash \mathscr{G}$ such that $poss(s) \neq \emptyset$, $c[s] := max(c[s], i)$ where $i = \min_{a \in poss(s)} c[s\_a]$.

(ii) For every state $s \in \mathscr{S}$, $a \in poss(s)$, and $s' \in \Phi(s,a)$, if $c[s'] = max$, then $c[s\_a] := max$.

(iii) For every state $s \in \mathscr{S}$ and $a \in poss(s)$, $c[s\_a] := max(c[s\_a], i+1)$ where $i = \min_{s' \in \Phi(s,a)} c[s']$.

(Step 3) If $c[s] = max$ for some $s \in \mathscr{I}$, then output that there is no strong cyclic plan; halt.

(Step 4) Output the plan $\pi : \mathscr{S} \rightarrow 2^{\mathscr{A}}$ defined on the states $s \in \mathscr{S} \backslash \mathscr{G}$ with $c[s] \leq max$ and $\pi(s) = \{a \mid a \in poss(s), c[s\_a] = \min_{b \in poss(s)} c[s\_b]\}$.

---

as DLV [PFE+06] or Smodels [SNS02]. These solvers support the computation of answer sets (models) of a given program, from which solutions (in this case, strong cyclic plans) can be extracted.

The encoding is generic, i.e., given by a *fixed program* which is evaluated over instances I represented by input facts $F(I)$. It makes use of the fact that non-monotonic logic programs can have multiple models, which correspond to different solutions, i.e., different strong cyclic plans.

The following first describes how a system is represented in a logic program,

and then develops the logic programs for both deterministic and general, nondeterministic domains. The syntax of DLV is adopted here. Only minor revisions are needed to adopt other Answer Set Solvers (e.g. Smodels).

Input Representation $F(I)$

The input I can be represented by facts $F(I)$ as follows.

- The following facts represent the planning domain $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$ and the planning problem $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$:
  - *state(s)*, for each $s \in \mathscr{S}$;
  - *action(a)*, for each $a \in \mathscr{A}$;
  - *trans(s,a,s')*, for each $s, s' \in \mathscr{S}$ and $a \in \mathscr{A}$ such that $s' \in \Phi(s,a)$;
- the set of states $\mathscr{I}$ is represented by using a predicate *start* by facts *start(s)*, for each $s \in \mathscr{I}$;
- the set of states $\mathscr{G}$ is represented by using a predicate *goals* by facts *goal(s)*, for each $s \in \mathscr{G}$;
- finally, the ranges $1 \ldots max$ and $2 \ldots max$ are represented using predicates *range*1 and *range*2, respectively.

Program P_{SC}

The program P_{SC}, executable on the DLV engine, for computing a strong cyclic plan is as follows.

```
%ranges
 range1(N) :- #int(N), N>0.
 range2(N) :- #int(N), N>1.
% 0
```

```
 s_bar(S,J1) :-  s_bar(S,J), J=J1+1.
% 1
 s_bar(S,I) :- state(S), not goal(S), range1(I),
      not some_path(S,I).
 some_path(S,I) :- range1(I), trans(S,A,Y),
     not s_a_bar(S,A,I).
% 2
 s_a_bar(S,A,#maxint) :- trans(S,A,Y), s_bar(Y,#maxint).
% 3
 s_a_bar(S,A,I) :- trans(S,A,Y), range1(I),
              not some_a_path(S,A,I).
 some_a_path(S,A,I) :- range1(I), I=I1+1, trans(S,A,Y),
              not s_bar(Y,I1).
% 4
 s_a_bar(S,A,I1) :- range2(I), I=I1+1, s_a_bar(S,A,I).
% 5
 :- s_bar(S,#maxint), start(S).
% 6
 s_bar(S,0) :- state(S), not goal(S).
% single out a plan
 pi(S,A) :- not s_a_bar(S,A,J), not goal(S), range1(J),
        not neg_max(S,A,J), trans(S,A,Y).
 neg_max(S,A,J) :- s_a_bar(S,A,J), range1(J), range1(J1),
       s_a_bar(S,A,J1), J < J1, trans(S,A,Y).
```

Besides the input predicates of $F(I)$, the program employs predicates s_bar(S,I) and s_a_bar(S,A,I) which intuitively correspond to $\overline{S_I}$ and $\overline{S\_A_I}$ respectively. The predicates fail_body(S,I) and fail_a_body(S,A,I) are used to uniformly represent clauses in (1) and (3), respectively, with varying body size; they amount to the negation of

s_bar(S,I) and s_a_bar(S,A,I), respectively. The plan is computed in the predicate pi(S,A).

Example 29. *The logic program encoding $F(I)$ of the strong cyclic planning problem in Example 27 is as follows:*

```
#maxint=3.
state(b). state(c). state(d). state(e).
start(b).  goal(e). action(x). action(y).
trans(b,x,c). trans(c,x,b). trans(c,x,e).
trans(b,y,d). trans(c,y,d).
```

The program $P_{SC} \cup F(I)$ has one answer set. Filtered to the atoms fail_a_body(s,a,i) and pi(s,a), the output is:

```
{ some_a_path(c,x,1), some_a_path(b,x,2),
  some_a_path(c,x,2), some_a_path(b,x,3),
  some_a_path(c,x,3), pi(b,x), pi(c,x)  }
```

Hence, the strong cyclic plan π given by $\pi(b) = \{x\}$ and $\pi(c) = \{x\}$ is obtained.

Preferred Plans

In general, there can be multiple answer sets, each corresponding to a different plan. Moreover, π can be non-deterministic; if in Example 29 a further action *z* would lead from *c* to *e*, then $\pi(c,e)$ would be in the result computed, and thus $\pi(c) = \{x,z\}$. By adding further rules in P_{SC}, A deterministic plan π_{det} can be generated, e.g. by nondeterministically selecting one action from $\pi(s)$:

```
pi_det(S,A) :- pi(S,A), not drop(S,A).
drop(S,A) v drop(S,B) :- pi(S,A), pi(S,B), A<>B.
```

For the case where multiple solutions exist, features available in Answer Set Solvers can be explored to select preferred plans. For example, using weak con-

straints offered by DLV, prioritization between different actions can be expressed. For illustration, the weak constraints

```
:~ pi_det(c,x). [:1]   :~ pi_det(c,z). [:2]
```

express that as for π_{det}, taking action z in state c is preferred over taking x. Using weak constraints, users can also easily model *costs* for action execution, possibly dependent on the state, which add up in execution. In this way, optimal (i.e., most preferred) plans among the candidates can be computed, possibly combining different criteria like deterministic actions and execution cost.

6.4 Finding Strong Plans

Finding strong plans can be approached in three ways: (i) as a special case of finite maintainability, when there are no exogenous actions; (ii) further constraining strong cyclic planning; or (iii) by a generic SAT encoding.

As for (ii), a Horn SAT encoding and genuine algorithm for strong planning are as follows:

Horn SAT Encoding $\overline{Strong}(P)$**:** The clauses (0), (1), (4), (5), (6) from $\overline{S\text{-}Cyclic}(P)$ and the following clauses:

(7) For every state $s \in \mathscr{S}$ and action $a \in \mathscr{A}$, for all $s' \in \Phi(s,a)$, and for all i, $0 < i \leq max$: $\overline{s'_{i-1}} \Rightarrow \overline{s\_a_i}$

Genuine procedure Strong Plan: Steps 1, 2.(i), 3, and 4 from Strong Cyclic Plan plus the new Step:

(Step 2) (ii′) For any state $s \in \mathscr{S}$, if $s' \in \Phi(s,a)$ for $a \in poss(s)$ and $c[s'] = i$ such that $0 \leq i \leq max$, then do $c[s\_a] := max(c[s\_a], i+1)$.

As discussed later, this yields algorithms of the same order as for strong cyclic planning.

The Horn SAT encoding in Algorithm 6.4 and the corresponding genuine procedure is more efficient.

[Horn SAT encoding of strong planning] Given a planning problem $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$, where $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$, the Horn instance $\overline{Strong}^{+}(P)$ contains:

(0) for every $s \in \mathscr{G}$: s

(1) for every state $s \in \mathscr{S} \backslash \mathscr{G}$ and action $a \in poss(s)$ such that $\Phi(s,a) = \{s'_1, \ldots, s'_m\}$, $m > 0$:
$s'_1 \wedge \cdots \wedge s'_m \Rightarrow s$ and $s'_1 \wedge \cdots \wedge s'_m \Rightarrow s\_a$.

(2) For $\mathscr{I} = \{s_1, \ldots, s_l\}$: $s_1 \wedge \cdots \wedge s_l \Rightarrow \bot$.

Theorem 3. *For a planning problem $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$,*

(i) a strong solution exists iff $\overline{Strong}^{+}(P)$ is unsatisfiable iff $\bot$ is derivable from $\overline{Strong}^{+}(P)$.

(ii) $\pi = \{\langle s,a \rangle \mid s\_a \in T^i_{P'}, s \notin T^{i-1}_{P'}$, for some $i \geq 1\}$, is a (non-deterministic) strong solution, where $T^1_{P'} = \mathscr{G}$ and $T^{i+1}_{P'} = \{\ell \mid \ell_1 \wedge \cdots \wedge \ell_l \Rightarrow \ell \in \overline{Strong}^{+}(P)$ and $\ell_1, \ldots, \ell_l, \in T^i_{P'}\}$ for $i \geq 1$, are the powers $T^i_{P'}$ of the logic programming immediate consequence operator $T_{P'}$ (see e.g. [DEGV01]) for the program $P' = \overline{Strong}^{+}(P)$ (viewing $\bot$ as atom).

A strong plan π as in the theorem can be constructed in $O(|\Phi| + |\mathscr{S}|)$ time starting from P, since $\overline{Strong}^{+}(P)$ is easily constructed and, as well-known, the powers of $T_{P'}$ are incrementally computable in linear time using proper data structures, cf. remarks in [DEGV01].

6.5 Finding Weak Plans

One way to think about finding weak plans is as relaxing strong cyclic planning. A respective Horn SAT encoding and genuine algorithm for Weak planning are as follows:

Horn SAT Encoding $\overline{Weak}(P)$: The clauses (0), (1), (3), (4), (5), (6) from $\overline{S\text{-}Cyclic}(P)$.

Genuine procedure: It consists of Steps 1, 2.(i), 2.(iii), 3, and 4 of algorithm Strong Cyclic Plan. (It does not contain the Step 2 (ii).)

Again, this yields algorithms of the same order as for strong cyclic planning. More efficient ones emerge from the encoding in Algorithm 6.5.

[Horn SAT encoding of weak planning] Given a planning problem $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$, where $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$, the Horn instance $\overline{Weak}^+(P)$ is as follows:

(0) for every $s \in \mathscr{G}$: s

(1) for every state $s \in \mathscr{S} \setminus \mathscr{G}$, action $a \in poss(s)$, and $s' \in \Phi(s,a)$: $s' \Rightarrow s\_a$ and $s' \Rightarrow s$.

Theorem 4. *For a planning problem $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$,*

(i) a weak solution exists iff for each $s \in \mathscr{I}$, $\overline{Weak}^+(P) \cup \{\neg s\}$ is unsatisfiable if and only if each $s \in \mathscr{I}$ is true in $M^*(\overline{Weak}^+(P))$, the least model of $\overline{Weak}^+(P)$.

(ii) $\pi = \{\langle s,a \rangle \mid s\_a \in M^*(\overline{Weak}^+(P))\}$, is a (non-deterministic) strong solution, if any strong solution exists.

Note that $\overline{Weak}^+(P)$ is definite Horn, and thus its least model $M^*(\overline{Weak}^+(P))$ does exist. Furthermore, it is computable in linear time in the size of $\overline{Weak}^+(P)$. Since the latter is easily constructed, finding a weak plan w.r.t. P is thus feasible in time $O(|\Phi| + |\mathscr{S}|)$, i.e., in linear time.

6.6 Complexity Analysis and Relations with Existing Algorithms

This section starts with the complexity analysis of the algorithms in this chapter.

For any planning domain $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$ and planning problem $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$, denote by $\|\mathscr{D}\| = |\mathscr{S}| + |\mathscr{A}| + |\Phi|$ and $\|P\| = \|\mathscr{D}\| + |\mathscr{I}| + |\mathscr{G}|$ the representation size of $\mathscr{D}$ and P, respectively (where Φ is viewed as set of triples $\langle s, a, s' \rangle$).

Proposition 24. *Strong Cyclic Planning can be solved, via the Horn SAT encoding* $\overline{\textit{S-Cyclic}}(P)$ *and, by a suitable implementation of Algorithm StrongCyclicPlan, in time* $O(|\mathscr{S}| \cdot \|P\|)$ *and* $O(|\mathscr{S}| \cdot |\Phi|)$*, respectively.*

Proof. (Sketch) As for the first part, the clauses in (0), (1), (2), (3), (4), (5), and (6) of $\overline{\textit{S-Cyclic}}(P)$ can be generated in time $O(|S|^2)$, $O(|S| \cdot |\Phi|)$, $O(|\Phi|)$, $O(|S| \cdot |\Phi|)$, $O(|S| \cdot |\Phi|)$, $O(|\mathscr{I}|)$, and $O(|S|)$, respectively. Hence $\overline{\textit{S-Cyclic}}(P)$ can be generated in time $O(|S|(|\Phi| + |S|)) = O(|S| \cdot \|P\|)$. Moreover, it can be solved in linear time in its size, i.e., in time $O(|S| \cdot \|P\|)$. From any model M obtained, C_M, l_M can be computed in time $O(|M|)$, and thus also K_M is computable in time $O(|M|)$. In summary, some control K_M as in Proposition 22 is computable in time $O(|S| \cdot \|P\|)$.

For the second part, Step 1 of *Strong Cyclic Plan* can be done in time $O(|S| + |\Phi|)$. For efficient realization of Step 2, employ auxiliary variables and data structures: a variable $\textit{Min\_act}(s) := \min(c[s\_a] \mid a \in \textit{poss}(s))$ for each $s \in \mathscr{S}$, a variable $\textit{Min\_next}(s,a) := \min(c[s'] \mid s' \in \Phi(s,a))$ for each $s \in \mathscr{S}$ and $a \in \textit{poss}(s)$, such that for each s_a $\textit{Min\_act}(s)$ is accessible in one step and likewise for each $s' \in \mathscr{S}$ a list $L_{s'}$ of all $\textit{Min\_next}(s,a)$ such that $s' \in \Phi(s,a)$. Furthermore, a set *Upd* of counters $c[s]$ and $c[s\_a]$ is maintained which are inspected for possible update. *Upd* has $O(1)$ membership, inclusion and exclusion tests (e.g., it is organized as a ring-list with an additional index), and is initialized with all counters (in $O(|\mathscr{S}| + |\Phi|)$ time). While *Upd* is not empty, a counter $c[s]$ resp. $c[s\_a]$ is removed from it for inspection. For the former, the update in 2.(i) and a possible follow update in 2.(ii) are efficiently

possible in $O(1)$ time. For the latter, the update in 2.(iii) is also feasible in $O(1)$ time.

Whenever one of the counters $c[s]$ resp. $c[s\_a]$ is increased, the elements $Min\_next(s',a)$ in L_s resp. $Min\_act(s)$ are updated, and the corresponding counters $c[s'\_a]$ resp. $c[s]$ are inserted in *Upd* upon a change. If $c[s]$ increased to $max-1$ and $s \in \mathscr{I}$, then the computation branches without this update to Step 3 (and halts); upon empty *Upd*, Step 3 can be skipped.

The number of updated $Min\_next(s',a)$ resp. inserted $c[s'\_a]$ for one update of counter $c[s]$ is $|\{\langle s,a\rangle \mid \langle s,a,s'\rangle \in \Phi\}|$; since $c[s]$ can increase no more than $|\mathscr{S}|$ times, over all $s \in \mathscr{S}$ the total number of such updates resp. inserts is bounded by $|\mathscr{S}|\cdot|\Phi|$. The total number of updated $Min\_act(s)$ resp. inserted $c[s]$ via $c[s\_a]$ is also bounded by $|\mathscr{S}|\cdot|\Phi|$. In total, Step 2 can be executed in time $O(|S|\cdot|\Phi|)$.

Step 4 can be done, using $Min\_act(s)$, in $O(|\Phi|)$ time.

In total, the time for Steps 1-4 is $O(|\mathscr{S}|\cdot|\Phi|)$. □

Remark that algorithm *Strong Cyclic Plan* can be made more efficient by pruning in a linear time preprocessing all states which are not on a path between some states $s \in \mathscr{I}$ and $s' \in \mathscr{G}$.

Comparing to [CPRT03], the algorithm for strong cyclic planning in this chapter works differently. Basically, their algorithm iteratively computes weak plans by backtracking from the goal states and prunes the planning problem until a weak plan which is also a strong cyclic plan is obtained. The algorithm, instead, has no such intuition and simply aims at establishing the necessary logical conditions, as in the seminal planning as satisfiability approach [KS92]. A simple implementation of the Cimatti *et al.* algorithm has $O(|\mathscr{S}|^2|\Phi|)$ time complexity, while a sophisticated one has $O(|\mathscr{S}|\cdot|\Phi|)$ comparable to ours. Section 6.6 compares these two algorithms in detail.

For finding strong plans and weak plans by constrained and relaxed strong cyclic planning, respectively, the following proposition holds.

Proposition 25. *Strong Planning (resp., Weak Planning) can be solved, via the encoding* $\overline{Strong}(P)$ *(resp.,* $\overline{Weak}(P)$*) in time* $O(|\mathscr{S}|\cdot\|P\|)$*, and by a properly implemented algorithm Strong Plan (resp., Weak Plan), in time* $O(|\mathscr{S}|\cdot|\Phi|)$.

Proof. (Sketch) The clauses (3.2') in $\overline{Strong}(P)$ can be generated in $O(|S|\cdot|\Phi|)$ time, and $\overline{Weak}(P)$ is a subset of $\overline{S\text{-}Cyclic}(P)$. The proof of the first part is thus very similar as in Proposition 24. The second part is also shown similarly as the second part of the Proposition 24. □

Simple implementations of the algorithms for strong and weak planning in [CPRT03] have time complexity $O(|S|\cdot|\Phi|)$, while more sophisticated ones have $O(\|P\|)$, i.e., linear time. For the special Horn encodings $\overline{Strong}+(P)$ and $\overline{Weak}+(P)$, the same time bound is obtained. They are closely related to the respective algorithms in [CPRT03] and may be viewed as declarative descriptions of the plan construction method. Nicely, an efficient implementation comes for free by the efficient algorithms for solving Horn theories.

As for the computational complexity of the planning problems, the following proposition is true. [1]

Proposition 26. *Deciding whether a given planning problem* $\langle\mathscr{D},\mathscr{I},\mathscr{G}\rangle$ *has*

1. *a strong cyclic solution is* ***P****-complete,*
2. *a strong solution is* ***P****-complete, and*
3. *a weak solution is* ***NLOG****-complete.*

[1] Reference for these results cannot be found, which might be known to the specialists, though.

The **P**-hardness results are an easy consequence of complexity results on k-maintainability [BEBN08]. The **NLOG**-membership of weak solutions is explained by the fact that as shown above, this reduces to solving for each $s \in \mathscr{I}$ a Horn SAT instance (Theorem 4) that is also a 2-SAT instance, which is feasible in **NLOG**. The **NLOG**-hardness follows from a simple reduction from the canonical graph teachability problem. Exploiting Theorem 4, also computing some weak plan is feasible in nondeterministic logspace.

Characteristics of the Algorithm

Now discuss the difference of this algorithm and the algorithm proposed in [CPRT03] on finding strong cyclic plans.

The algorithm is based on evolving from the set of goal states. Labels are assigned to states to indicate that there is a path from the state to a state in $\mathscr{G}$. Besides, states that do not have a path to a state in $\mathscr{G}$ are removed as well when their labels are increased to *max* in the algorithm. On the other hand, the algorithm in [CPRT03] is proceed by iteratively removing states and actions that are not able to reach a goal state. The "envelope" of possible solutions is reduced rather than being extended for computing the greatest fix point.

In the case that a plan can be easily found and the plan involves a small subset of states in transition graph, this approach is more efficient. With the approach in [CPRT03], the whole transition system still need to be explored thoroughly before a plan can be found.

This chapter first encodes the problems in SAT. One thing keep in mind that any heuristics are avoided in the encoding so that the approach can be named as "finding algorithms from specification". However, in most cases, heuristics are the basis for the encodings to be solved faster. For example, in the strong cyclic planning encoding, if users want to find a maximal or lean plan, they need to encode

beforehand on choices of actions in a state.

Now consider a few possible modifications to improve the performance of the algorithm here.

- One limitation of this algorithm is that it is not "guided". The algorithm is based on a SAT encoding thus no search heuristics are encoded in the algorithm. Performance of the approach proposed here heavily depends on ordering in exploring states and in changing labels. In the iteration step in Algorithm 6.3, if all neighbors of a state do not change their labels, it is not possible that the state will change its label. However, this algorithm may still need to check these states repeatedly. One approach to improve the performance of this algorithm is to prefer states or actions whose neighbors change their labels recently when examining labels of states and actions.

  Another observation is that the not "guided" algorithm might exploit part of the transition graph that not related to the finial plan. For example, if there is a sub-graph in the transition graph such that nodes in the sub-graph connect to each other but none of these states has a path to a state in $\mathscr{G}$. This algorithm needs to increase labels of these nodes repeatedly until their labels reach *max* before they can be excluded from consideration. However, as there is no weak plan from state *s* to a state in $\mathscr{G}$, state *s* can be removed from consideration and the label of state *s* is set to *max* directly.

Also note that in some cases, there is no need to set variable *max* to $|\mathscr{S}|-1$. The following proposition illustrates one such case.

Proposition 27. *Iff there is a strong cyclic plan such that the length of the longest path from the initial state to the goal state involves at most k nodes, a strong cyclic plan can be found by setting* $max = k-1$ *in Algorithm 6.3.*

Based on Proposition 27, a parameter k can be introduced in the logic program encoding of the algorithm. This simplifies the algorithm. However, given a planning problem, the value of k is not known. What can be done is to increase the value of k incrementally before a solution to the planning problem is found.

6.7 Applying the Approach to other π-CTL* Goals

Consider applying the approach to other π-CTL* goals. Consider some variations of the strong cyclic planning. As encoding the problem in reverse Horn SAT in the most critical step, in this section, only the reverse Horn SAT encoding of each problem is considered. The rest steps of translating to Horn SAT or extracting a genuine algorithm follow the same approach as strong cyclic planning in Section 6.3.

Planning for Goal $\mathsf{A}_{pol}\Box(\mathsf{E}\Diamond p)$

The goal $\mathsf{A}_{pol}\Box(\mathsf{E}\Diamond p)$ is considered. It is different from strong cyclic planning $\mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)$ in that symbol E states that the path satisfying $\Diamond p$ may not be one path of the agent. This can be done by the Horn SAT specification in Algorithm 6.7.

In the SAT encoding, for each state s and action a, propositions s_i, $\underline{s_i}$, and s_a_i are used, where $i \geq 0$ is an integer. Intuitively, s_i will mean that there is a path from s to a state satisfying $\mathsf{E}\Diamond p$, following the execution structure $K = \langle Q, T\rangle$, of length at most i. Similarly, s_a_i will intuitively mean that there is a path from s to a state satisfying $\mathsf{E}\Diamond p$ of length at most i, following T of the execution structure $K = \langle Q, T\rangle$, and with a as its first action. $\underline{s_i}$ means that there is a path from state s to $\mathscr{G}$, of length i, not necessarily following T of the execution structure.

max is defined as $|\mathscr{S}| - 1$. It is the upper bound of i. If there is no path of length at most *max*, there is no path at all.

This planning problem is easier than the strong cyclic planning. Finding one such plan is not very interesting since if there is a weak plan from the initial state, then the plan that takes the action "*nop*" in the initial state is a valid plan.

[Reverse Horn SAT encoding for planning with the goal $\mathsf{A}_{pol}\Box(\mathsf{E}\Diamond p)$]
Suppose given a planning problem $P=\langle \mathscr{D}, \mathscr{I}, \mathscr{G}\rangle$ where $\mathscr{D}=\langle \mathscr{S}, \mathscr{A}, \Phi\rangle$. Suppose $max=|\mathscr{S}|-1$. P is translated into a reverse Horn SAT encoding as follows:

(0) for all $s \in \mathscr{S}$ and i, $0 < i \leq max$: $s_{i-1} \Rightarrow s_i$; $\underline{s_{i-1}} \Rightarrow \underline{s_i}$.

(1) for every state $s \in \mathscr{S} \backslash \mathscr{G}$: $s_i \Rightarrow \bigvee_{a \in poss(s)} s\_a_i$

(1.2) for every state $s \in \mathscr{S} \backslash \mathscr{G}$, and for any state $s' \in \mathscr{S}$, if $s' \in \Phi(s,a)$ for any action a: $\underline{s'_i} \Rightarrow \underline{s_{i+1}}$

(3') for every state $s \in \mathscr{S}$, action $a \in poss(s)$, and for all i, $0 < i \leq max$: $s\_a_i \Rightarrow s'_{i-1}$

(3")for every state $s \in \mathscr{S}$, $s_0 \Rightarrow \underline{s_{max}}$

(4) for every state $s \in \mathscr{S}$, action $a \in poss(s)$, and $1 < i \leq max$: $s\_a_{i-1} \Rightarrow s\_a_i$

(5) for $s \in \mathscr{I}$: s_{max}

(6) for $s \in \mathscr{S} \setminus \mathscr{G}$: $\neg \underline{s_0}$

(7) for $s \in \mathscr{G}$: $\underline{s_0}$

Planning for Goal $\mathsf{A}_{pol}\Diamond(\mathsf{E}\Diamond p)$

The encoding of this problem is the same as the encoding of $\mathsf{A}_{pol}\Box(\mathsf{E}\Diamond p)$. It is easy to check that in a domain, a strong cyclic plan also satisfy the goal $\mathsf{A}_{pol}\Box(\mathsf{E}\Diamond p)$ and $\mathsf{A}_{pol}\Diamond(\mathsf{E}\Diamond p)$.

Planning for Goal $\mathsf{A}\Box(\mathsf{E}_{pol}\Diamond p)$

A policy satisfy this goal if for any state that is reachable from the initial state, there is always a path to a state with p being true by following the policy. This goal differs from the strong cyclic plan in that it takes care of all states that are reachable from the initial states besides the states that are reachable from the initial states by following the policy. If states in $\mathscr{G}$ are all the states that have proposition p true, then the goal can be presented in π-CTL* as $\mathsf{A}\Box(\mathsf{E}_{pol}\Diamond p)$. A plan satisfying this goal is also a strong cyclic plan.

Suppose given a planning problem $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$ where $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$. To solve the problem, a new planning problem is defined such that $P' = \langle \mathscr{D}, \mathscr{I}', \mathscr{G} \rangle$ where $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$. Let I' be the set of states that are reachable from I. Any weak plan of P' is a plan for $\mathsf{A}\Box(\mathsf{E}_{pol}\Diamond p)$ in P. This observation is utilized in the following encoding.

In the SAT encoding, for each state s and action a, define propositions s_i, $\underline{s_i}$, and s_a_i, where $i \geq 0$ is an integer. Intuitively, s_i will mean that there is a path from s to $\mathscr{G}$, following the execution structure $K = \langle Q, T \rangle$, of length at most i. Similarly, s_a_i will intuitively mean that there is a path from s to $\mathscr{G}$ of length at most i, following T of the execution structure $K = \langle Q, T \rangle$, and with a as its first action. $\underline{s_i}$ means that there is a path from state s to $\mathsf{E}_{pol}\Diamond p$, of length i, not necessarily following T of the execution structure.

An upper bound *max* for i is defined, depending on the number of states in $\mathscr{S}$; if there is no path of length at most *max*, there is no path at all.

Planning for Goal $\mathsf{A}\Diamond(\mathsf{E}_{pol}\Diamond p)$

Remove Item (5") from Algorithm 6.7.

6.8 Planning with a Pref-π-CTL* Goal

This section finds plans for the planning problem $\mathsf{A}_{pol}\Box((\mathsf{A}_{pol}\Diamond p \lhd \mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)) \lhd \mathsf{E}_{pol}\Diamond p)$ in Pref-π-CTL*. This goal states that in any state of the plan starting from the initial state, a strong plan is always preferred to a strong cyclic plan, which is in turn preferred to a weak plan. Thus it is possible that the agent is starting with a weak plan, but switch to a strong cyclic plan if it happens to get to a state with a strong cyclic plan. The agent may further switch to a strong plan if he is lucky enough to get to a state with a strong plan exists. This goal states that in any state of the agent, the agent checks all policies available to him and choose the best one. This is different from the goal $(\mathsf{A}_{pol}\Diamond p \lhd \mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)) \lhd \mathsf{E}_{pol}\Diamond p$ that finds the

[Reverse Horn SAT encoding for planning of the goal $\mathsf{A}\Box(\mathsf{E}_{pol}\Diamond p)$]
Suppose given a planning problem $P=\langle\mathscr{D},\mathscr{I},\mathscr{G}\rangle$ where $\mathscr{D}=\langle\mathscr{S},\mathscr{A},\Phi\rangle$. Let $max=|\mathscr{S}|-1$. P is translated into a SAT encoding *S-Cyclic*(P) as follows:

(0) for all $s\in\mathscr{S}$ and i, $0<i\leq max$: $s_{i-1}\Rightarrow s_i$

(1) for every state $s\in\mathscr{S}\backslash\mathscr{G}$, and for all i, $0<i\leq max$: $s_i\Rightarrow\bigvee_{a\in poss(s)}s\_a_i$

(3) for every state $s\in\mathscr{S}$, action $a\in poss(s)$, and for all i, $0<i\leq max$: $s\_a_i\Rightarrow\bigvee_{s'\in\Phi(s,a)}s'_{i-1}$

(4) for every state $s\in\mathscr{S}$, action $a\in poss(s)$, and $1<i\leq max$: $s\_a_{i-1}\Rightarrow s\_a_i$

(5) for $s\in\mathscr{I}$: $\underline{s_{max}}$

(5') for $s\in\mathscr{S}$, $s'\in\Phi(s,a)$: $\underline{s_{i+1}}\Rightarrow\underline{s'_i}$

(5") for $s\in\mathscr{S}$, $\underline{s_{max}}\Rightarrow s_{max}$

(6) for $s\in\mathscr{S}\setminus\mathscr{G}$: $\neg s_0$

(7) for $s\in\mathscr{G}$: s_0

(8) for $s\in\mathscr{S}$, $\underline{s_0}\Rightarrow s_{max}$

strong planning from the initial state first, if there is no strong plan from the initial state, finds alternative plans such as strong cyclic plans or weak plans.

Weak, strong, and strong cyclic planning problems are first investigated in [CPRT03]. Later, in [BEZ05], a different approach was taken by following the method first proposed in [BEBN08] that first encode each problem in reverse Horn. Later, an algorithm was extracted by simulating the approach of solving the reverse Horn. Strong, weak, and strong cyclic planning problems all can be solved in $O(S\cdot P)$, where S is the number of states in the transition graph, and P is the total number of states, actions, and transitions in the transition graph.

One way of solving the problem given above is to find strong, weak, and strong cyclic plans from all states in the transition graph, and then merge the plans found. With this approach, the Pref-π-CTL* goal above can be solved in $O(S^2\cdot P)$. Now show that based on the properties of these planning problems, a better solution can

be found. Note that the algorithm proposed is not a reverse Horn encoding that solves this Pref-π-CTL* goal. The algorithm is composed of a few steps:

Input: A planning domain $\mathscr{D} = \langle \mathscr{S}, \mathscr{A}, \Phi \rangle$, and a planning problem $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$.
Output: A plan π to Pref-π-CTL* goal $\mathsf{A}_{pol}\Box((\mathsf{A}_{pol}\Diamond p \lhd \mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)) \lhd \mathsf{E}_{pol}\Diamond p)$ if such plan exists. Otherwise, output that no such plan exists.

1. Step 1: (Strong plan extension): For state $s \in \mathscr{S} \setminus \mathscr{G}$ and action a in the transition system, if $\Phi(s,a) \subseteq \mathscr{G}$, add state s to $\mathscr{G}$, and add the pair (s,a) to the plan π, which is a set of pairs of states and actions.

2. Step 2: (Strong cyclic plan extension): For state $s \in \mathscr{S} \setminus \mathscr{G}$ with action a such that $\Phi(s,a) \cap \mathscr{G} \neq \emptyset$. Run Algorithm 6.3 to find a strong cyclic plan from s to $\mathscr{G}$ if there is one. Suppose the output of the algorithm is π' and π' is defined on a set of sates S'. Let $\mathscr{G} = \mathscr{G} \cup \{s | s \in S'\}$, and $\pi = \pi \cup \{(s, \pi'(s)) | s \in S'\}$. Repeat the process until no states can be added to $\mathscr{G}$.

3. Step 3: (Weak plan extension): For state $s \in \mathscr{S} \setminus \mathscr{G}$ and action a in the transition system, if $\Phi(s,a) \cap \mathscr{G} \neq \emptyset$, add state s to $\mathscr{G}$, and add the pair (s,a) to the plan π, which is a set of pairs of states and actions.

4. Step 4: (Output) If $\mathscr{S} \subseteq \mathscr{G}$, return the policy π. Otherwise, output that no such plan exists.

In the worst case, time complexity of the algorithm is $O(s^2 \cdot P)$. But it is faster than the approach of finding strong, weak, and strong cyclic plans from all states in the transition graph, and merging the plans found. The algorithm can be implemented more efficiently as follows:

1. In Each step of the algorithm in expanding the current plan, a few state-action pairs are added to the plan after each step.

2. In the process of growing the plan, an index is maintained such that only actions leads to states in $\mathscr{G}$ are considered in the checking process.

3. Alternate Step 1 and Step 2, as after extending $\mathscr{G}$ in Step 2, there may be more states that have strong plans to the current $\mathscr{G}$.

A Program Simulating the Algorithm

Weak, strong, and strong cyclic planning problems are encoded in dlv [PFE+06] logic program [GL88]. A Python program as described in Algorithm 6.8 is given. It invokes DLV for solving planning problems with different initial states and goal states. [2]

In particular, the logic program encoding of strong cyclic plan is the same as in Section 6.3. Logic program encoding of strong plan is as follows:

```
% ranges
range1(N) :- #int(N), N>0.
range2(N) :- #int(N), N>1.
% 0
s_bar(S,I1) :- s_bar(S, I), I=I1+1, range1(I).
% 1
s_bar(S, I) :- state(S), not goal(S), range1(I), not some_path(S,I).
some_path(S,I) :- range1(I), trans(S,A,Y), all_a_path(S,A,I).
all_a_path(S,A,I) :- not s_a_bar(S,A,I), not self_loop(S,A),
    trans(S,A,Y), range1(I).
self_loop(S,A) :- trans(S,A,S).
% 7
s_a_bar(S,A,I1) :- s_bar(Y,I), I1=I+1, range1(I), trans(S,A,Y).
% 4
s_a_bar(S,A,I1) :- range2(I), I=I1+1, s_a_bar(S,A,I).
% 5
:- s_bar(S, #maxint), start(S).
% 6
s_bar(S,0) :- state(S), not goal(S).
```

[2] The Python program and logic program encodings of strong, weak, and strong cyclic planning are available at: http://www.public.asu.edu/∼jzhao6/find-best-plan.rar

```
% single out the plan
pi(S,A) :- all_a_path(S,A,J), not goal(S), range1(J),
    not neg_l_M(S,A,J), trans(S,A,Y).
neg_l_M(S,A,J) :- s_a_bar(S,A,J), range1(J), range1(J1),
    s_a_bar(S,A,J1), J < J1, trans(S,A,Y).
```

One thing to note is that in the logic program encoding of the strong planning, rules corresponding to Item (1) are different from the corresponding rules in strong cyclic planning and weak planning. Apparently, a strong plan cannot have actions that leads a self-loop. An action a in state s with $s \in \Phi(s,a)$ need to be removed from any strong plan.

Running the program on the transition graph in Figure 3.1 is illustrated in Example 30.

Example 30. *Consider the transition domain $\mathscr{D}$ in Figure 3.1. Initially, in the planning problem $P = \langle \mathscr{D}, \mathscr{I}, \mathscr{G} \rangle$, $\mathscr{I} = \{s_1\}$, $\mathscr{G} = \{s_4\}$ and the policy $\pi = \emptyset$.*

In Step 1 of the algorithm, state s_2 and s_3 are checked as they are states which have actions lead to states in $\mathscr{G}$. s_2 is the only state that has a strong plan to $\mathscr{G}$. Thus $\mathscr{G}$ now is $\{s_2, s_4\}$ and $\pi = \{(s_2, a_2)\}$.

In Step 2, check states s_1 and and s_3 as they are states which have actions lead to states in $\mathscr{G}$. s_3 is the only state that has a strong cyclic plan to current $\mathscr{G}$. Thus $\mathscr{G}$ now is $\{s_2, s_3, s_4\}$ and $\pi = \{(s_2, a_2), (s_3, a_3)\}$.

In Step 3, the only state to be check is s_1. There is a weak plan from s_1 to $\mathscr{G}$. Thus after adding s_1 to $\mathscr{G}$ and (s_1, a_1) to the policy, it is known that $\mathscr{G} = \{s_1, s_2, s_3, s_4\}$ and $\pi = \{(s_1, a_1), (s_2, a_2), (s_3, a_3)\}$.

As $\mathscr{S} \subseteq \mathscr{G}$, it is known that the plan π is a plan satisfies the goal $\mathsf{A}_{pol}\Box((\mathsf{A}_{pol}\Diamond p \lhd \mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)) \lhd \mathsf{E}_{pol}\Diamond p)$ in the transition domain in Figure 3.1.

Also note that the policy return in the algorithm is a "power policy" where

multiple actions might be defined for the same state.

6.9 Related Work

This work belongs to the reasoning about action community. One major part of the work is to define languages for expressing goals of agents in non-deterministic domains and then study the relations of goals and policies in complicated domains for semi-automatic agents. With the goals expressed, it is an interesting topic to explore the approach of finding plans for some special temporal goals by following a "representation, translation, and simulation" approach.

A few work in planning community relate to what this chapter is doing. One direction is to have temporal domain knowledge in planning as in [BK98, NN01, SBTM02]. HTN planning [NCLMA99] also loosely related to this as it involves temporal logic in defining strategies. Recently PDDL extension with temporal aspects and other work also related to what this chapter is doing.

Another direction related to this work is to have an representation of the problem firstly, and then translating the encoding to a similar problem with known techniques. Early work in this direction are planning via satisfiability encoding [KS92] or logic programming encoding with answer set semantics [GL91]. The symbolic representation such as BDD of the planning problem is also related to this work. In recent years, there have been some work on planning in non-deterministic domains for particular temporal formulas [DLPT02, CPRT03, JVB04].

The central motivations in the first direction mentioned above is to use known planning techniques for temporal domain. Some of them translate the temporal goal so as to use traditional planning techniques. Some of them use temporal logics as heuristics to guild the search. In this work, in stead of finding planning heuristics, the focus is on how the goals are precisely defined when the domain is becoming more and more complicated.

The central motivations in the second direction mentioned above is to use symbol representation to reduce search space or use existing general purpose symbolic solver. Different from them, in stead of finding general approach that is good for any planning problem, we focus on a subset of planning problems that can be solved in polynomial time when the input is the state space.

6.10 Discussion

This section discusses a few issued related to the approach of finding plans for π-CTL* goals. As planning with goals in π-CTL* is EXPTIME-hard, no plans can be found for any π-CTL* with this approach. However, there are still some planning problems that can be found in polynomial time by applying the approach.

Applying the Approach to Other Planning Problems

As goal specification languages becoming more and more expressive, more properties of the domain and the agent can be captured in the goal specification language. Thus planning with goals expressed in these language are more difficult. On the other hand, as goal specification languages becoming more and more expressive, some goals expressed in the language might become easier solve as more restrictions are enforced on the goal.

One motivation of the work is that there is a need of pointing out a set of desired states. By analyzing the relations of these desired states, initial states, and other states, a SAT program and, further, an Horn program is specified. The general idea of making use of the approach is that different labels can be defined for each state and then consider the relations of labels among related states. For example, in strong cyclic planning and *k*-maintain problem, there are only two labels: True and False. The indexes in algorithms in this chapter are also used in defining the right ordering of propagating the labels. Initially, only the set of goals states are labeled as True. These labels are propagated hence the relations of different labels

are encoded as reverse Horn rules.

Due to the property that there is a need of pointing out a set of final state in the SAT encoding, it is not easy, if possible, to use the approach for P-CTL* goals due to the lack of dealing with quantifying over policies in the approach.

Reasoning and Planning as Goal Specification Revision

In general, a goal is to define what the plan is regardless of the transition system. While a plan is generated by given a specific transition system.

By considering a policy as a strategy taken for the agent, the goal specification is to have some requirement on properties of such a structure in any domain. As the goal specification languages becoming more and more specific, given a domain and a goal specification, it might be easier to find out a policy that satisfies such a goal. In the other word, the difference of goal specification and the planning is minimized and they only differ in the availability of the domain configuration and the availability of the action formula. For example, in the extreme case, if the goal specification defines the action to take in any possible situations, then given the domain and the possible actions in the domain, the plan can be naturally deduced by a table lookup. Apparently, users do not want to give such a too specific specification but only want to give a general direction in the goal specification. One problem is that to what extend users think a goal specification expressive enough and general enough?

6.11 Summary

This chapter shows that the methodology in [BEBN08] can be used to develop polynomial time planning algorithms for various kinds of problems in a non-deterministic domain, viz. for weak, strong, and strong cyclic planning. Small modifications to the algorithm obtained for strong cyclic planning, whose complexity is comparable to a sophisticated implementation of the Cimatti *et al.* algorithm, yield polynomial

algorithms for strong and weak planning. Furthermore, simple, genuine Horn encodings give efficient (linear time) implementations of Cimatti *et al.*'s strong and weak plan construction method at an abstract level. This matches with a complexity analysis of the problems which is provided in this chapter. This chapter also shows how strong cyclic planning can be declaratively done in non-monotonic logic programming, using an Answer Set Solver. By exploiting features of such solvers, a (most) preferred among multiple candidate plans, depending on criteria like deterministic actions, action preference, or action cost might be singled out.

Finally, the approach in this chapter can be considered as another illustration of automatically generating algorithms from specifications. The propositional encoding of k-maintainability, and weak, strong, and strong cyclic planning can be thought of as a specification of these problems. Thus the results here and the results in [BEBN08] illustrate the realization of a long standing goal of many software engineers and algorithm designers who were interested in the problem of automatically obtaining algorithms or programs from specifications.

Chapter 7

CONCLUSION

Over the previous chapters, a few temporal logics for representing goals of an agent are proposed. Logics are defined in giving directions to agents in non-deterministic domains. As the domain or the intension users have for the agent may change after the initial goal was given to the agent, languages are proposed to handle non-monotonic aspects of goal specification. Besides, as an agent may have different preference relations among its sub-goals at different stages of its plan, a language capable of dealing with dynamic preferences is defined. Planning algorithms for a few goals represented in these logics are also given in previous chapters.

This chapter summarizes contributions of this dissertation, and points out some future directions.

7.1 Summary

A systematic design of an autonomous agent has three main aspects: (i) domain description: actions that an agent can do, their impacts, environment, and etc.; (ii) control execution of an agent; and (iii) directives for an agent. Focus of this dissertation is on the goal specification aspect in autonomous agent design, and its relation with other aspects.

In defining a goal specification language, the following questions need to be answered: What is a goal? What is a goal specification language? Whether a goal is represented in a language? Whether the set of goals expressed in one language is a superset of the goals expressed in the other language? Whether the set of goals expressed in a language depends on the ability (or the policy structure) of the agent? These questions are formally answered in this question. A framework for checking goals expressed in a language and for comparing goal specification languages is

proposed.

In a non-deterministic domain, many interesting goals cannot be expressed using existing temporal logics such as LTL and CTL*. A formal proof of this is given in the dissertation. A policy in a non-deterministic domain leads to a set of paths thus users need to distinguish the paths in a policy from all paths of the domain. This is captured in the proposed language π-CTL*. In order to compare policies that are available to an agent so that the agent can choose the most fit ones, language P-CTL* is proposed to capture the intuition of quantifying over policies. Besides, policies of an agent play an important role in defining goal specification languages. There are also paths in the domain that is not in any policy of the agent. Goal specification languages with different policy structures are also defined to address this issue.

One interesting aspect of this work is that it illustrates the difference between program specification and goal specification. Temporal logics are developed in the context of program specification, where the program statements are deterministic and there are no goals of the kind "trying one's best". (There is no specifications for a program to try its best to do something.) In cognitive robotics, actions have non-deterministic effects and sometimes one keeps trying until one succeeds, and similar attempts to try one's best. The proposed language P-CTL* allows the specification of such goals. P-CTL* has the ability of letting the agent to compare and analyze policies and "adjust" its expectations accordingly.

Also, the policy structure of an agent plays an important rule on what goals can be expressed in the language, and what goals can be achieved by the agent. This work is the first one pointing out the impact of policies on the set of goals expressed in the language. A policy structures can be defined as a mapping from states to actions, and as a mapping from histories to actions. Different policy structures can be defined for different agents. For example, for agents with sensing actions, or

agents who can reason about knowledge of other agents, policy structures can be defined by taking the sensing actions and other abilities of the agent into account. It is a challenge problem to define languages for different agents and compare these languages.

The second part of the dissertation is about defining goals that can change non-monotonically. In many domains, users need to specify goals that might be further revised or partially retracted due to incomplete information users have about the domain. Thus non-monotonic temporal logics are needed to specify goals which can then be revised in an elaboration tolerant manner. Two non-monotonic extensions of LTL are proposed. Labels are used to denote sub-goals. Sub-goals can be defeated when there are exceptions. This work borrows the idea of completion and exception from logic programming. It borrows the idea of a surface non-monotonic logic from Reiter. Their applications in modeling revisions are illustrated. The way of progressing an ER-LTL program is also discussed. This is important as agents receiving new requirements might have already executed some actions to satisfy earlier goals. Thus, the agents need to progress the previous requirements and the new requirements based on their earlier states. A program of translating an ER-LTL program to LTL is given.

In defining non-monotonic goal specification languages, it is challenge to handle temporal operators in a formula. It is common to define a non-monotonic logic as a set of rules, and semantics are based on models entailed from the rules. Similarly, in defining N-LTL and ER-LTL, a goal is considered to be a set of rules. Instead of computing models for the program, labels are used to connect rules into one temporal formula. These labels are also used to denote exceptions. These labels enable users in representing many interesting changes to the initial goals.

The third direction considered in this dissertation is the preferences in goal specification. A goal specification language with preferences is proposed. The language

is based on π-CTL*. A binary connective $\lhd$ is introduced to compare state formulas. Comparing to other goal specification languages with preferences, the new language Pref-π-CTL* is the only one for non-deterministic domains. Besides, by treating the $\lhd$ operator the same way as other temporal operators, language Pref-π-CTL* has some interesting properties such as allowing nested preferences and dynamic preferences. For example, different preference relations among sub-goals can be defined in one formula. More importantly, the preferences relations might change as the agent proceeds in satisfying other sub-goals.

This dissertation also examines some planning problems in the proposed goal specification languages. This dissertation follows the approach proposed in [BEBN08] that solves planning problems by encoding the problem in a reverse Horn SAT, translating the reverse Horn to Horn SAT, and then extracting algorithms by simulating the steps of finding models of the Horn SAT. New algorithms algorithms for strong, weak, and strong cyclic plan are found. Logic program encodings of these planning problems are also proposed. By writing a program of calling DLV solvers on these different logic programs, plans can be found for one interpretation of the goal of "trying one's best".

The work on goal specification has great impact in autonomous agent design, especially for designing agents in a non-deterministic domain or a open world where states or goals of the agent may be changed dynamically.

7.2 Future Directions

It is important to represent and reason about goals of an agent. In order to design an autonomous agent, goals and policies of the agent need to be incorporated with the domain of the agent. There are some directions in goal specification deserve more investigation. They are listed in the following:

- In Chapter 4, it is mentioned that the definition of the policy structure affects

the set of goals expressed in a language. The impact of policy structures on goal specification languages is investigated. It is interesting to consider goal specification languages with other definitions of the policy structure. It is also interesting to define a language that can incorporate multiple definitions of the policy structure.

- There are a lot of well known non-monotonic logics defined for domains other than temporal logics. In [PSBZ10], authors attempted to apply default logic on defining a non-monotonic temporal logic. Whether other non-monotonic logics can be directly used in defining non-monotonic temporal logics is still unclear. On the other hand, the mechanism in N-LTL and ER-LTL can also be applied to other logics such as propositional logic. Comparing the resulted logic with well known non-monotonic logics is a work needs further investigations. These studies will reveal more insights on non-monotonic temporal logics, and on non-monotonicity in general.

- There are also work on goal specification with a different transition system. Recently there are work in planning in an open world [TBS$^+$10]. Non-monotonic properties of goal specifications in such a domain is an interesting topic.

- In defining preferences of a goal specification language, it is unclear how to define a goal specification language that can deal with point-wise preferences where distances between trajectories and the partial satisfaction of a temporal formula are considered.

- Also, in logic Pref-π-CTL* that deals with dynamic preferences, binary connective $\lhd$ is defined for comparing states. $sf_1 \lhd sf_2$ states that policies satis-

fying sf_1 are preferred to policies only satisfying sf_2, etc. Other semantics of the binary connective $\lhd$ are interesting to look at.

- Finally, in Chapter 6, a few goals that can be solved in polynomial time are studied. Which other subsets of goals can be solved in polynomial time is a direction of big impact.

BIBLIOGRAPHY

[AH93] Rajeev Alur and Thomas A. Henzinger. Real time logics: complexity and expressiveness. *Inform. and Comput.*, 104(1):35–77, 1993.

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49:672–713, 2002.

[All84] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[BBG96] Fahiem Bacchus, Craig Boutilier, and Adam J. Grove. Rewarding behaviors. In *AAAI 96*, pages 1160–1167, 1996.

[BBG97] F. Bacchus, C. Boutilier, and A. Grove. Structured solution methods for non-markovian decision processes. In *AAAI 97*, pages 112–117, 1997.

[BCGR99] S. Bistarelli, P. Codognet, Y. Georget, and F. Rossi. Labeling and partial local consistency for soft constraint programming. In *Second International Workshop on Practical Aspects of Declarative Languages*, pages 230–248, 1999.

[BDH99] Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Joural of Artificial Intelligence Research*, 11:1–94, 1999.

[BEBN08] Chitta Baral, Thomas Eiter, Marcus Bjäreland, and Mutsumi Nakamura. Maintenance goals of agents in a dynamic environment: formulation and policy construction. In *Artificial Intelligence*, volume 172, pages 1429–1469, 2008.

[BEH95a] A. Bouajjani, R. Echahed, and P. Habermehl. On the verification problem of nonregular properties for non-regular procsses. In *Symposium on logics in computer science*, pages 123–133, 1995.

[BEH95b] Ahmed Bouajjani, Rachid Echahed, and Peter Habermehl. Verifying infinite state processes with sequential and parallel composition. In *POPL '95: Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 1995.

[BEZ05] Chitta Baral, Thomas Eiter, and Jicheng Zhao. Using SAT and logic programming to design polynomial-time algorithms for planning in non-deterministic domains. In *AAAI-05*, pages 578–583, 2005.

[BG00] B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *AIPS 2000*, pages 52–61, 2000.

[Bil98] David Billington. Defeasible deduction with arbitrary propositions. In *Poster Proceedings of the 11th Australian Joint Conference on Artificial Intelligence*, pages 3–14, 1998.

[bis] www.gnu.org/software/bison.

[BK98] Fahiem Bacchus and Froduald Kabanza. Planning for temporally extended goals. *Annals of Math and AI*, 22:5–27, 1998.

[BKSD95] M. Barbeau, F. Kabanza, and R. St-Denis. Synthesizing plan controllers using real-time goals. In *IJCAI*, pages 791–800, 1995.

[BKT01] Chitta Baral, Vladik Kreinovich, and Raul Trejo. Computational complexity of planning with temporal goals. In *IJCAI-01*, pages 509–514, 2001.

[BZ04] Chitta Baral and Jicheng Zhao. Goal specification in presence of non-deterministic actions. In *Proceedings of ECAI'04*, pages 273–277, 2004.

[BZ06] Chitta Baral and Jicheng Zhao. Goal specification, non-deterministic, and quantifying over policies. In *AAAI-06*, pages 231–237, 2006.

[BZ07] Chitta Baral and Jicheng Zhao. Non-monotonic temporal logics for goal specification. In *IJCAI-07*, pages 236–242, 2007.

[BZ08] Chitta Baral and Jicheng Zhao. Non-monotonic temporal logics that facilitate elaboration tolerant revision of goals. In *AAAI-08*, 2008.

[CE81] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching-time tempral logic. In *Workshop on Logic of programs. Lecture Notes in Computer Science*, volume 131, pages 52–71, 1981.

[CM88] K. Chandy and J. Misra. *Parallel program design: a foundation*. Addison Wesley, 1988.

[CPRT03] A. Cimatti, M. Pistore, M. Roveri, and P. Travrso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

[DCDS01] Pallab Dasgupta, P. P. Chakrabarti, Jatindra Kumar Deka, and Sriram Sankaranarayanan. Min-max computation tree logic. *Artificial Intelligence*, 127(1):137–162, 2001.

[DEGV01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[DG84] W. Dowling and H. Gallier. Linear time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1:267–284, 1984.

[dLPdB08] Silvio do Lago Pereira and Leliane de Barros. Using α-ctl to specify complex planning goals. In Wilfrid Hodges and Ruy de Queiroz, editors, *Logic, Language, Information and Computation*, volume 5110 of *Lecture Notes in Computer Science*, pages 260–271. Springer Berlin / Heidelberg, 2008.

[DLPT02] U. Dal Lago, M. Pistore, and P. Traverso. Planning with a language for extended goals. In *AAAI'02*, pages 447–454, 2002.

[EC82] E. A. Emerson and E. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer programming*, 2:241–266, 1982.

[EL85] E.A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proc. 18th Hawaii International Conference on System Sciences*, pages 277–288, 1985.

[Eme90] E. Allen Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 997–1072, 1990.

[EMSS92] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. *Real-Time System*, 4(4):331–352, 1992.

[ES84] E. A. Emerson and Prasad Sistla. Deciding branching time logic. In *ACM STOC*, pages 14–24, 1984.

[ES89] E. Allen Emerson and Jai Srinivasan. Branching time temporal logic. In J.W. de Bakker, W. P. de Roever, and G. Roezenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 123–172. Springer-Verlag, Berlin, 1989.

[ET99] E. Allen Emerson and Richard J. Trefler. Parametric quantitative temporal reasoning. *Logic in Computer Science*, pages 336–343, 1999.

[FG00] P. Ferraris and E. Giunchiglia. Planning as satisfiability in nondeterministic domains. In *AAAI-00*, pages 748–753, 2000.

[FH91] Yasushi Fujiwara and Shinichi Honiden. A nonmonotonic temporal logic and its kripke semantics. *J. Inf. Process.*, 14(1):16–22, 1991.

[FN71] Richard Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Aritificial Intelligence*, 2(3-4):189–208, 1971.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proc. ICLP*, pages 1070–1080, 1988.

[GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[GL98a] Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on Artificial Intelligence*, 2(3-4):193–210, 1998.

[GL98b] Michael Gelfond and Vladimir Lifschitz. Action languages. *Electronic Transactions on AI*, 3, 1998.

[GL05] Alfonso Gerevini and Derek Long. Plan constraints and preferences in pddl3 - the language of the fifth international planning competition. Technical report, University of Brescia, 2005.

[GV99] G. De Giacomo and M. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *ECP*, pages 226–238, 1999.

[HF85] Joseph Y. Halpern and Ronald Fagin. A formal model of knowledge, action, and communication in distributed systems: preliminary report. In *4th ACM symp on principles of distributed computing*, pages 224–236, 1985.

[HNSY92] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:394–406, 1992.

[JL03] Jan Johannsen and Martin Lange. CTL+ is complete for double exponential time. In *Proc. 30th Int. Coll. on Automata, Logics and Programming*, volume 2719 of LNCS, pages 767–775, 2003.

[JVB04] Rune M. Jensen, Manuela M. Veloso, and Randal E. Bryant. Fault tolerant planning: Toward probabilistic uncertainty models in symbolic non-deterministic planning. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *Proceedings 14th International Conference on Automated Planning and Scheduling (ICAPS 2004), Whistler, British Columbia, Canada, June 3-7, 2004*, pages 335–344, 2004.

[KS86] Robert Kowalski and Marek Sergot. A logic based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[KS92] H. Kautz and B. Selman. Planning as satisfiability. In *ECAI-92*, pages 359–363, 1992.

[LMO06] François Laroussinie, Nicolas Markey, and Ghassan Oreiby. Expressiveness and complexity of ATL. Research Report LSV-06-03, Laboratoire Spécification et Vérification, ENS Cachan, France, February 2006. 20 pages.

[LPV01] V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2001.

[McC59] John McCarthy. Programs with common sense. In *Proc. Teddington Conf. on the Mechanization of Thought Processes*, pages 75–91, 1959.

[McC98] John McCarthy. Elaboration tolerance. In *COMMON SENSE 98, Symposium On Logical Formalizations Of Commonsense Reasoning*, January 1998.

[McD82a] Drew McDermott. Non-monotonic logic II: Nonmonotonic modal theories. *Journal of the ACM*, 29:33–57, 1982.

[McD82b] Drew McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[McD00] D. McDermott. The 1998 ai planning systems competition. *Artificial Intelligence Magazine*, 21(2):35–56, 2000.

[MH69] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[Moo85] Robert C. Moore. Semantical considerations on nonmonotonic logic. *Artif. Intell.*, 25(1):75–94, 1985.

[MP92] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems: specification*. Springer Verlag, 1992.

[MW84] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic specification. *ACM Transactions on Programming Languages and Systems*, 6(1):68–93, Jan 1984.

[NCLMA99] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP: Simple hierarchical ordered planner. In *IJCAI-99*, page 968?973, 1999.

[NN01] Keijo Neljanko and Ilkka Niemelä. Bounded LTL model checking with stable models. In *LPNMR*, pages 200–212, 2001.

[NS00] Rajdeep Niyogi and Sudeshna Sarkar. Logical specification of goals. In *Proc. of 3rd international conference on Information Technology*, pages 77–82, 2000.

[Nut87] Donald Nute. Defeasible reasoning. In *Proceedings of the 20th Hawaii International Conference on System Science*, pages 470 – 477, 1987.

[Ped87] E. P. D. Pednault. Formulating multiagent, dynamic-world problems in the classical planning framework. In *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, pages 47–82, 1987.

[Ped89] Edwin 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.

[PFE+06] Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning nicola leone. *ACM transactions on Computational Logic (TOCL)*, 7(3):1–57, 2006.

[Pnu77] A. Pnueli. The temporal logic of programs. In *18th IEEE Symp. on Foundation of Computer Science*, pages 46–57, 1977.

[PR89] A. Pnueli and R. Rosner. On the synthsis of a reactive module. In *ACM POPL*, pages 179–190, 1989.

[PSBZ10] Enrico Pontelli, Tran Cao Son, Chitta Baral, and Jicheng Zhao. Goal default theory with priorities as a non-monotonic goal specification language. In *NonMon'30*, 2010.

[PT01] M. Pistore and P. Traverso. Planning as model checking for extended goals in non-deterministic domains. In *IJCAI'01*, pages 479–486, 2001.

[Rei87] R. Reiter. *Readings in nonmonotonic reasoning*, chapter A logic for default reasoning, pages 68–93. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[Rei91] Ray Reiter. The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial intelligence and mathematical theory of computation: papers in honour of John McCarthy*, pages 359–380. Academic Press Professional, 1991.

[Rei01] Ray Reiter. *Knowledge in action : logical foundations for specifying and implementing dynamical systems*. MIT Press, 2001.

[Sae87] M. Saeki. Non-monotonic temporal logic and its application to formal specifications (in japaneese). *Transactions of IPS Japan*, 28(6):547–557, 1987.

[SBTM02] Tran Cao Son, Chitta Baral, Nam Tran, and Sheila Meilraith. Domain-dependent knowledge in answer set planning. CS 007, New Mexico State University, 2002.

[SC79] L. J. Stockmeyer and A. K. Chandra. Provably difficult combinatorial games. *SIAM Journal on Computing*, 8(2):151–174, 1979.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[Sch87] M. Schoppers. Universal plans for reactive robots in unpredictable environments. In *IJCAI 87*, pages 1039–1046, 1987.

[Sch03] Ph. Schnoebelen. The complexity of temporal logic model checking. *Advances in Modal Logic*, 4:393–436, 2003.

[Sho87] Y. Shoham. *Reasoning about change*. MIT Press, Boston, MA., 1987.

[SNS02] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

[SP06] Tran Cao Son and Enrico Pontelli. Planning with preferences using logic programming. *TPLP*, 6:559–608, 2006.

[SSD00] Rajdeep Niyogi Sudeshna Sarkar, P.P. Chakrabarti and P. Dasgupta. Specification of planning goals in branching time logic in stochastic systems. In *Proceedings of the International Conference, KBCS2000*, pages 422–433, 2000.

[TBS+10] Kartik Talamadupula, J. Benton, Paul Schermerhorn, Subbarao Kambhampati, and Matthias Scheutz. Integrating a closed world planner with an open world robot: A case study. In *AAAI*, 2010.

[TGS+06] S. Thiebaux, C. Gretton, J. Slaney, D. Price, and F. Kabanza. Decision-theoretic planning with non-markovian rewards. *Journal of AI Research, pages 17–74, 2006.*, 25:17–74, 2006.

[Thi98] Michael Thielscher. Introduction to the fluent calculus. *Electronic Transactions on Artificial Intelligence*, 2(3-4):179–192, 1998.

[vdHJW05] Wiebe van der Hoek, Wojciech Jamroga, and Michael Wooldridge. A logic for strategic reasoning. In *AAMAS-05*, pages 157–164, 2005.

[WD05] Michael Wooldridge and Paul E. Dunne. The complexity of agent design problems: Determinism and history dependence. *Ann. Math. Artif. Intell.*, 45(3-4):343–371, 2005.

Appendix A

DEFINITION ON DEPTH OF A FORMULA

The depth of a formula used in proofs is defined here.

Definition 51. *Let sf, sf_1 and sf_2 be state formulas, pf, pf_1 and pf_2 be path formulas in CTL*. Let the depth of a CTL* formula g be $depth(g)$.*

The depth of a state formula is given as follows:

- *The depth of an atomic proposition is 1;*
- $depth(\neg sf) = 1 + depth(sf)$;
- $depth(sf_1 \wedge sf_2) = depth(sf_1) + depth(sf_2) + 1$;
- $depth(sf_1 \vee sf_2) = depth(sf_1) + depth(sf_2) + 1$;
- $depth(\mathsf{E}pf) = 1 + depth(pf)$;
- $depth(\mathsf{A}pf) = 1 + depth(pf)$;

The depth of a path formula is given as follows:

- *if the path formula pf is defined in terms of a state formula sf, then $depth(pf) = depth(sf)$;*
- $depth(\neg pf) = depth(pf) + 1$;
- $depth(\bigcirc pf) = depth(pf) + 1$;
- $depth(\Diamond pf) = depth(pf) + 1$;
- $depth(\Box pf) = depth(pf) + 1$;
- $depth(pf_1 \wedge pf_2) = depth(pf_1) + depth(pf_2) + 1$;
- $depth(pf_1 \vee pf_2) = depth(pf_1) + depth(pf_2) + 1$;
- $depth(pf_1 \mathsf{U} pf_2) = depth(pf_1) + depth(pf_2) + 1$; □

Definition 52. *Let sf, sf_1 and sf_2 be state formulas, pf, pf_1 and pf_2 be path formulas in π-CTL*. Let the depth of a formula g be $depth(g)$.*

The depth of state formulas and path formulas are the same as that defined for CTL* formulas in Definition 51. Besides that, depth of formulas with two new operators A_{pol} and E_{pol} are as follows:

- $depth(\mathsf{E}_{pol}\, pf) = depth(pf) + 1$;
- $depth(\mathsf{A}_{pol}\, pf) = depth(pf) + 1$. □

Definition 53. *Let sf, sf_1 and sf_2 be state formulas, pf, pf_1 and pf_2 be path formulas in P-CTL*. Let the depth of a formula f be $depth(f)$.*

Depth of state formulas of new operators are given as follows:

- $depth(\mathscr{E}\mathscr{P} sf) = depth(sf) + 1$;
- $depth(\mathscr{A}\mathscr{P} sf) = depth(sf) + 1$.

Depth of state and path formulas with other operators are the same as that defined in Definition 52. Depth of P_σ-CTL* and π_σ-CTL* formulas are the same as that in P-CTL* and π-CTL*. □

Appendix B

YET ANOTHER APPROACH OF DEFINING THE EXPRESSIVENESS OF A GOAL-SPECIFICATION LANGUAGE

There are other ways of defining expressiveness of a goal specification language. This section elaborate on one of such alternatives. Each goal specification language defines a set of formulas, each formula maps a transition graph and an initial state to sets of trajectories. Two goal specification languages may differ in that one has more formulas defined, or each formula is mapped to a different set of trajectories for each initial state and transition graph. We now present an approach for defining expressiveness of goal specification languages.

Some notations are defined first. Let L be a goal specification language. Let g be a formula in L, Φ be a transition function and s_0 be a state in it, $\models_L$ be the entailment relation in language L. We use $Pset(g, s_0, \Phi, \models_L)$ to denote the set $\{\pi : (s_0, \Phi, \pi) \models_L g\}$ as the set of policies satisfying g in L. By $Gset(\pi, s_0, \Phi, \models_L)$, we denote the set $\{g : (s_0, \Phi, \pi) \models_L g\}$ as the set of goal formulas satisfied by policy π in L. Let G_L be all formulas in language L. Let $P_L(s_0, \Phi)$ be all policies in language L in Φ starting from s_0.

A goal g, which is a mapping from pairs of transition graph and initial state to sets of trajectories, is *not expressible* in a goal specification language L if there are Φ_1, Φ_2, s_0^1, s_0^2 such that

1. For any policy π_1 that is valid in Φ_1 starting from s_0^1 and valid in Φ_2 starting from s_0^2, we have that π_1 is mapped by the same set of formulas in (s_0^1, Φ_1) and (s_0^2, Φ_2).

2. However, a policy π_1 is mapped by the goal g in Φ_1 and s_0^1 but π_1 is not mapped by the goal g in Φ_2 and s_0^2.

The reason is that if g can be expressed in language L as ϕ_g, according to Item 1, we know ϕ_g maps (s_0^1, Φ_1) and (s_0^2, Φ_2) to the same set of policies. Thus a policy is mapped by the goal g in Φ_1 and s_0^1 iff it is mapped by the goal g in Φ_2 and s_0^2.

This and Item 2 are contradict each other, thus the goal g cannot be expressed in language L.

With this definition, we are able to compare languages that rely on the same definition of policies. However, we are not able to compare two languages that rely on irrelevant definitions of policy. We will discuss the comparisons of languages with different, but related notions of policies in Section B.2.

B.1 Notation on Comparing Languages

With the proof that some goals cannot be expressed in a goal specification language, we can define the expressiveness of a goal specification language. It can be used to compare two different languages based on sets of goals can be expressed in them. Besides, to compare two languages that are similar, we may consider whether one language is a "superset" of the other. When one language has more constructs than the other, we can define a notion of equivalence between them by considering only the common subset of goal formulas and policies in them. Formally, we have the following definition for comparing languages that are similar in both goal formulas and policy structures.

Definition 54 ($\succeq_{equalSyntax,equalPolicy}$)**.** *Consider two languages L_1 and L_2. $L_1 \succeq_{equalSyntax,equalPolicy} L_2$ if*

1. $G_{L_2} \subseteq G_{L_1}$;
2. $\forall \Phi, \forall s_0, P_{L_2}(s_0, \Phi) \subseteq P_{L_1}(s_0, \Phi)$;
3. $\forall g \in G_{L_2}, \forall \Phi, \forall s_0, Pset(g, s_0, \Phi \models_{L_1}) \cap P_{L_2}(s_0, \Phi) = Pset(g, s_0, \Phi, \models_{L_2})$;
4. $\forall \pi \in P_{L_2}, \forall \Phi, \forall s_0, Gset(\pi, s_0, \Phi, \models_{L_1}) \cap G_{L_2} = Gset(\pi, s_0, \Phi, \models_{L_2})$. □

Intuitively, $L_1 \succeq_{equalSyntax,equalPolicy} L_2$ if for any formula g_2 in L_2, there is a formula g_1 in L_1 such that the set of policies satisfying g_1 in L_1 is the same as the set of

policies satisfying g_2 in L_2. The subscript "syntax,policy" in $\succeq_{equalSyntax,equalPolicy}$ indicates that the two languages share a comparable syntax and a comparable notion of policy.

Proposition 28. *In Definition 54, Items (1-3) and Items (1-2, 4) are equivalent.*

Proof. We first prove Item (4) given items (1) - (3).

For any Φ and s_0, for any policy $\pi \in P_{L_2}(s_0, \Phi)$, if a goal formula $G \in Gset(\pi, s_0, \Phi, \models_{L_2})$, according to the definitions, we know $\pi \in Pset(G, s_0, \Phi, \models_{L_2})$. From Item (3), we know $\forall G \in G_{L_2}$, $\forall \Phi$, $\forall s_0$, $Pset(G, s_0, \Phi, \models_{L_1}) \cap P_{L_2}(s_0, \Phi) = Pset(G, s_0, \Phi, \models_{L_2})$. Thus, $\pi \in Pset(G, s_0, \Phi, \models_{L_1})$. This is the same as $G \in Gset(\pi, s_0, \Phi, \models_{L_1})$. Since $G \in Gset(\pi, s_0, \Phi, \models_{L_2})$, we know $G \in G_{L_2}$. Thus $G \in Gset(\pi, s_0, \Phi, \models_{L_1}) \cap G_{L_2}$. This implies that $Gset(\pi, s_0, \Phi, \models_{L_1}) \cap G_{L_2} \supseteq Gset(\pi, s_0, \Phi, \models_{L_2})$.

On the other hand, for any Φ and s_0, for any policy $\pi \in P_{L_2}(s_0, \Phi)$, if $G \in Gset(\pi, s_0, \Phi, \models_{L_1}) \cap G_{L_2}$, then according to the definitions, we have $\pi \in Pset(G, s_0, \Phi, \models_{L_1})$. According to Definition 54, since $\pi \in Pset(G, s_0, \Phi, \models_{L_1}) \cap P_{L_2}(s_0, \Phi)$, we know $\pi \in Pset(G, s_0, \Phi, \models_{L_2})$. This is equivalent to $G \in Gset(\pi, s_0, \Phi, \models_{L_2})$. Thus $Gset(\pi, s_0, \Phi, \models_{L_1}) \cap G_{L_2} \subseteq Gset(\pi, s_0, \Phi, \models_{L_2})$.

By combining the results above, we have $Gset(\pi, s_0, \Phi, \models_{L_1}) \cap G_{L_2} = Gset(\pi, s_0, \Phi, \models_{L_2})$. Thus Item (4) is true if Item (1) - (3) are true.

Similarly, if Item (1), (2), and (4) are true, Item (3) is true. □

Note that when we define $L_1 \succeq_{equalSyntax,equalPolicy} L_2$, we require that these two languages have similar policy structures and goal formulas. However, in general, two languages may differ in policy structures and goal formulas. For example, in comparing π-CTL* and π_σ-CTL*, even though for any policy in π-CTL*, we can construct another policy in π_σ-CTL* that corresponds to the same set of trajectories, these two policies are quite different: A policy in π-CTL* is a mapping from states to actions while a policy in π_σ-CTL* is a mapping from state sequences to actions.

We now define a more general notion for comparing two languages that may differ in policy structures or goal formulas.

Definition 55 ($\succeq_{equalPolicy}$)**.** *Given two languages L_1 and L_2, $L_1 \succeq_{equalPolicy} L_2$ if there is a one-to-one mapping φ from G_{L_2} to G_{L_1} such that for all Φ and s_0, and for all $g \in G_{L_2}$, $Pset(g, s_0, \Phi, \models_{L_2}) = Pset(\varphi(g), s_0, \Phi, \models_{L_1}) \cap P_{L_2}(s_0, \Phi)$.* □

Definition 56 ($\succeq_{equalSyntax}$)**.** *Given two languages L_1 and L_2, $L_1 \succeq_{equalSyntax} L_2$ if there is a one-to-one mapping ψ from $P_{L_2}(s_0, \Phi)$ to $P_{L_1}(s_0, \Phi)$ for any transition system Φ and state s_0 such that for all policy π, $Gset(\pi, s_0, \Phi, \models_{L_2}) = Gset(\psi(\pi), s_0, \Phi, \models_{L_1})) \cap G_{L_2}$.* □

The subscript "equalPolicy" in $\succeq_{equalPolicy}$ indicates that the two languages share a comparable notion of policies. The subscript "equalSyntax" in $\succeq_{equalSyntax}$ indicates that the two languages share a comparable syntax, formulas defined in these two languages are the same. It is easy to check that $\succeq_{equalSyntax,equalPolicy}$, $\succeq_{equalSyntax}$, and $\succeq_{equalPolicy}$ are all partial orders.

Before we compare specific languages, now consider the relationship between the various notions.

Proposition 29. *Let L_1, L_2 be two goal specification languages. If $L_1 \succeq_{equalSyntax,equalPolicy} L_2$, then $L_1 \succeq_{equalPolicy} L_2$ and $L_1 \succeq_{equalSyntax} L_2$.*

Proof. 1. If $L_1 \succeq_{equalSyntax,equalPolicy} L_2$, then

a) $G_{L_2} \subseteq G_{L_1}$;

b) $\forall g \in G_{L_2}, \forall \Phi, \forall s_0, Pset(g, s_0, \Phi, \models_{L_1}) \cap P_{L_2}(s_0, \Phi) = Pset(g, s_0, \Phi, \models_{L_2})$;

Let φ be a one-to-one mapping from G_{L_2} to G_{L_1} such that $\varphi(g) = g$. We have $\forall g \in G_{L_2}$, $\forall \Phi$, $\forall s_0$, $Pset(g, s_0, \Phi, \models_{L_2}) = Pset(\varphi(g), s_0, \Phi, \models_{L_1}) \cap P_{L_2}(s_0, \Phi)$. Thus $L_1 \succeq_{equalPolicy} L_2$.

2. If $L_1 \succeq_{equalSyntax,equalPolicy} L_2$, then

   a) $\forall \Phi$, $P_{L_2}(\Phi) \subseteq P_{L_1}(\Phi)$;

   b) $\forall \pi \in P_{L_2}$, $\forall \Phi$, $\forall s_0$, $Gset(\pi, s_0, \Phi, \models_{L_1}) \cap G_{L_2} = Gset(\pi, s_0, \Phi, \models_{L_2})$.

   Let ψ be a one-to-one mapping from $P_{L_2}(s_0, \Phi)$ to $P_{L_1}(s_0, \Phi)$ for any Φ and s_0 such that $\psi(\pi) = \pi$ for any policy π. Thus, for all s_0, and π, $Gset(\pi, s_0, \Phi, \models_{L_2}) = Gset(\psi(\pi), s_0, \Phi, \models_{L_1}) \cap G_{L_2}$. As a result, $L_1 \succeq_{equalSyntax} L_2$.

□

Similarly, we can also define the comparison of languages having different sets of formulas or policies by defining a mapping from policies in one language to policies in the other language, or a mapping from goals in one language to goals in the other language. We are not going to elaborate on them.

B.2 Compare Different Goal Specification Languages

We defined when one language is more general than the other language, it is related to the the notion of a goal is not expressive in a language.

Proposition 30. *If language $L_1 \succeq_{equalSyntax} L_2$, and for all Φ and state s, $P_{L_1}(s, \Phi) = P_{L_2}(s, \Phi)$, then any goal that can be expressed in L_2 can be expressed in L_1.*

Proof. If language $L_1 \succeq_{equalSyntax} L_2$, and for all Φ and state s, $P_{L_1}(s, \Phi) = P_{L_2}(s, \Phi)$, then we have the following conditions:

1. For all Φ and state s, $P_{L_1}(s, \Phi) = P_{L_2}(s, \Phi)$;

2. $\forall \Phi$ and state s, $\forall \pi \in P_{L_2}(s, \Phi)$, $\forall s_0$, $Gset(\pi, s_0, \Phi, \models_{L_2}) = Gset(\pi, s_0, \Phi, \models_{L_1}) \cap G_{L_2}$.

If there is a goal g that cannot be expressed in L_1, there are Φ_1, Φ_2, s_0^1, s_0^2 such that

1. Exists $\pi_1 \in P_{L_1}(s_0^1, \Phi_1) \cap P_{L_1}(s_0^2, \Phi_2)$ such that

   $Gset(\pi_1, s_0^1, \Phi_1, \models_{L_1}) = Gset(\pi_1, s_0^2, \Phi_2, \models_{L_1})$;

2. Goal g is satisfied by the policy π_1 w.r.t. $(s_0^1, \Phi_1, \models_{L_1})$ but not w.r.t. $(s_0^2, \Phi_2, \models_{L_1})$.

We now prove that such a goal g cannot be expressed in L_2:

1. Since for all Φ and state s, $P_{L_1}(s, \Phi) = P_{L_2}(s, \Phi)$, we know $\pi_1 \in P_{L_2}(s_0^1, \Phi_1) \cap P_{L_2}(s_0^2, \Phi_2)$.

$$\begin{aligned} Gset(\pi_1, s_0^1, \Phi_1, \models_{L_2}) &= Gset(\pi_1, s_0^1, \Phi_1, \models_{L_1}) \cap G_{L_2} \\ &= Gset(\pi_1, s_0^2, \Phi_2, \models_{L_1}) \cap G_{L_2} \\ &= Gset(\pi_1, s_0^2, \Phi_2, \models_{L_2}); \end{aligned}$$

2. For all Φ and state s, $P_{L_1}(s, \Phi) = P_{L_2}(s, \Phi)$. Thus, goal g is satisfied by the policy π_1 w.r.t. $(s_0^1, \Phi_1, \models_{L_2})$ but not w.r.t. $(s_0^2, \Phi_2, \models_{L_2})$.

Thus g cannot be expressed in L_2. □

Similarly, we have the following relation on two languages:

Proposition 31. *If language $L_1 \succeq_{equalPolicy} L_2$, and $G_{L_1} = G_{L_2}$, then any goal that can be expressed in L_2 can be expressed in L_1.*

Now we know there are two related approaches of comparing goal specification languages:

1. find a goal that is not expressible in one language while is expressible in the other, or

2. compare the policy-goal relations in two languages.

Compare Different Languages

We now utilize these notions in comparing proposed languages. A goal specification language is considered as a mapping from pairs of transition system and initial state to sets of trajectories.

We now use the definitions we have to compare the languages listed above to formally prove the relations of the languages. We first compare pairs of languages that have the same set of policies while the syntax of one language in each pair is a superset of the other. We then compare pairs of languages that have the same syntax.

Compare π-CTL* with P-CTL*

Given a transition graph and an initial state, languages π-CTL* and P-CTL* have the same set of policies. On the other hand, language P-CTL* has more goal formulas than π-CTL*. But each formula in π-CTL*, there is a formula in P-CTL* that maps to the same set of policies. That is, if we restrict language P-CTL* on a subset of goal formulas, the resulted mapping from formulas to policies is identical to π-CTL*.

This means that more goals can be represented in P-CTL*, and for any goal that can be represented in π-CTL*, the same goal can be represented as the same formula in P-CTL*.

Lemma 4. *Consider languages π-CTL* and P-CTL*.*

(i) For any transition function Φ, state s_0, a policy π as a mapping from states to actions, and state formula φ in π-CTL*, $(s_0, \Phi, \pi) \models \varphi$ in language π-CTL* iff $(s_0, \Phi, \pi) \models \varphi$ in language P-CTL*;

(ii) For any transition function Φ, state s_0, policy π as a mapping from states to actions, path formula ψ in π-CTL* and path σ in Φ, $(s_0,\Phi,\pi,\sigma) \models \varphi$ in language π-CTL* iff $(s_0,\Phi,\pi,\sigma) \models \varphi$ in language P-CTL*.

Proof. The proof is based on the induction on depth of formulas.

Base case: It is easy to see that for any state formula or path formula of depth 1, the conditions (i) and (ii) hold.

Induction: Assume that it is true for formulas of depth less than *n*, and show that it is true for formulas of depth *n*.

Consider state formulas of depth *n*. It can be of the following forms: (a) $sf_1 \wedge sf_2$ (b) $sf_1 \vee sf_2$ (c) $\neg sf_1$ (d) $\mathsf{E}pf$ (e) $\mathsf{A}pf$ (f) $\mathsf{E}_{pol}pf$ (g) $\mathsf{A}_{pol}pf$, where sf_1, sf_2 and pf have depth less than *n*.

Consider (d) $\mathsf{E}pf$. By definition, $(s_0,\Phi,\pi) \models \mathsf{E}\ pf$ in π-CTL* iff there exists a path σ in Φ starting from s_1 such that $(s_0,\Phi,\pi,\sigma) \models pf$ in π-CTL*. By definition, $(s_0,\Phi,\pi) \models \mathsf{E}\ pf$ in P-CTL* iff there is a path σ in Φ starting from s_1 satisfying $(s_0,\Phi,\pi,\sigma) \models pf$ in P-CTL*. According to induction hypothesis, we know $(s_0,\Phi,\pi,\sigma) \models pf$ in π-CTL* iff $(s_0,\Phi,\pi,\sigma) \models pf$ in P-CTL* since $depth(pf) < n$. Hence, $(s_0,\Phi,\pi,\sigma) \models \mathsf{E}pf$ in π-CTL* iff $(s_0,\Phi,\pi,\sigma) \models \mathsf{E}pf$ in P-CTL*.

The proofs for formulas of other forms are similar.

Consider path formulas of depth *n*. It can be of the following forms: (a) $pf_1 \wedge pf_2$ (b) $pf_1 \vee pf_2$ (c) $\neg pf_1$ (d) $pf_1\ \mathsf{U}\ pf_2$ (e) $\bigcirc pf_1$ (f) $\Diamond pf_1$ (g) $\Box pf_1$, where depth of pf_1 and pf_2 are less than *n*. The proof of each of these cases is similar to the proof of state formulas. □

Proposition 32. *P-CTL* $\succ_{equalSyntax,equalPolicy}$ π-CTL*.*

Proof. It is easy to know that $G_{\pi-CTL^*} \subseteq G_{P-CTL^*}$ and for all Φ, $P_{\pi-CTL^*}(\Phi) \subseteq P_{P-CTL^*}(\Phi)$. We now need to prove that for any goal $g \in G_{\pi-CTL^*}$, for any Φ and s_0,

$Pset(g, s_0, \Phi, \models_{P-CTL^*}) \cap P_{\pi-CTL^*} = Pset(g, s_0, \Phi, \models_{\pi-CTL^*})$. Since $P_{\pi-CTL^*}(\Phi) = P_{P-CTL^*}(\Phi)$, this is equivalent to $Pset(g, s_0, \Phi, \models_{P-CTL^*}) = Pset(g, s_0, \Phi, \models_{\pi-CTL^*})$. That is, for all $g \in G_{\pi-CTL^*}$, for all transition function Φ, for all state s_0, $(s_0, \Phi, \pi) \models_{\pi-CTL^*} g$ iff $(s_0, \Phi, \pi) \models_{P-CTL^*} g$. This is the result of Lemma 4. Thus P-CTL* $\succeq_{equalSyntax,equalPolicy}$ π-CTL*.

Further, as $G_{\pi-CTL^*} \subseteq G_{P-CTL^*}$, we know P-CTL* $\neq_{equalSyntax,equalPolicy}$ π-CTL*, thus P-CTL* $\succ_{equalSyntax,equalPolicy}$ π-CTL*. □

According to Proposition 30, since P-CTL* $\succeq_{equalSyntax,equalPolicy}$ π-CTL*, we know that all goals expressed in π-CTL* can be expressed in P-CTL*, and there is a goal in P-CTL* that cannot be expressed in π-CTL*.

Compare π_σ-CTL* with P_σ-CTL*

The relation between π-CTL* and P-CTL* holds for π_σ-CTL* and P_σ-CTL* as well. This means that more goals can be represented in P_σ-CTL*, and for any goal that can be represented in π_σ-CTL*, the same goal can be represented as the same formula in P_σ-CTL*.

Lemma 5. *Consider languages π_σ-CTL* and P_σ-CTL*.*

(i) For any transition function Φ, state s_0, policy π that maps from state sequences to actions, and state formula φ in π_σ-CTL*, $(s_0, \Phi, \pi) \models \varphi$ in π_σ-CTL* iff $(s_0, \Phi, \pi) \models \varphi$ in P_σ-CTL*;

(ii) For any transition function Φ, state s_0, policy π that maps from states sequences to actions, path formula ψ in π_σ-CTL* and path σ in Φ, $(s_0, \Phi, \pi, \sigma) \models \varphi$ in π_σ-CTL* iff $(s_0, \Phi, \pi, \sigma) \models \varphi$ in P_σ-CTL*.

Proof. The proof is based on the induction on depth of formulas.

Base case: It is easy to see that (i) and (ii) are true for any state formula or path formula of depth 1.

Induction: Assume that it is true for formulas of depth less than n, and show that it is true for formulas of depth n.

Consider state formulas of depth n. It can be of the following forms: (a) $sf_1 \wedge sf_2$ (b) $sf_1 \vee sf_2$ (c) $\neg sf_1$ (d) $\mathsf{E}pf$ (e) $\mathsf{A}pf$ (f) $\mathsf{E}_{pol}pf$ (g) $\mathsf{A}_{pol}pf$, where sf_1, sf_2 and pf have depth less than n.

Consider (d) $\mathsf{E}\ pf$. By definition, $(s_0,\Phi,\pi) \models_{\pi_\sigma-CTL^*} \mathsf{E}\ pf$ iff there exists a path σ in Φ starting from s_1 such that $(s_0,\Phi,\pi,\sigma) \models_{\pi_\sigma-CTL^*} pf$. By definition, $(s_0,\Phi,\pi) \models_{P_\sigma-CTL^*} \mathsf{E}\ pf$ iff there exists a path σ in Φ starting from s_1 such that $(s_0,\Phi,\pi,\sigma) \models_{P_\sigma-CTL^*} pf$. According to the induction hypothesis, we know $(s_0,\Phi,\pi,\sigma) \models_{\pi_\sigma-CTL^*} pf$ iff $(s_0,\Phi,\pi,\sigma) \models_{P_\sigma-CTL^*} pf$ since $depth(pf) < n$. Hence, $(s_0,\Phi,\pi,\sigma) \models_{\pi_\sigma-CTL^*} \mathsf{E}pf$ iff $(s_0,\Phi,\pi,\sigma) \models_{P_\sigma-CTL^*} \mathsf{E}pf$.

The proofs for formulas of other forms are similar.

Consider path formulas of depth n. It can be of the following forms: (a) $pf_1 \wedge pf_2$ (b) $pf_1 \vee pf_2$ (c) $\neg pf_1$ (d) $pf_1\ \mathsf{U}\ pf_2$ (e) $\bigcirc pf_1$ (f) $\Diamond pf_1$ (g) $\Box pf_1$, where pf_1 and pf_2 have depth less than n. The proof of each of these cases is similar to the proof for state formulas. □

Similar to the relations in π-CTL*, we have the following result.

Proposition 33. *P_σ-CTL* $\succ_{equalSyntax,equalPolicy}$ π_σ-CTL*.*

Proof. It is easy to know that $G_{\pi_\sigma-CTL^*} \subseteq G_{P_\sigma-CTL^*}$ and for all Φ, $P_{\pi_\sigma-CTL^*}(\Phi) \subseteq P_{P_\sigma-CTL^*}(\Phi)$. We now need to prove that for any goal $g \in G_{\pi_\sigma-CTL^*}$, for any Φ and s_0, $Pset(g,s_0,\Phi,\models_{P_\sigma-CTL^*}) \cap P_{\pi-CTL^*} = Pset(g,s_0,\Phi,\models_{\pi-CTL^*})$. Since $P_{\pi-CTL^*}(\Phi) = P_{P-CTL^*}(\Phi)$, this is equivalent to $Pset(g,s_0,\Phi,\models_{P-CTL^*}) = Pset(G,s_0,\Phi,\models_{\pi-CTL^*})$. That is, for all $g \in G_{\pi-CTL^*}$, for all transition function Φ, for all state s_0, $(s_0,\Phi,\pi) \models_{\pi-CTL^*} g$ iff $(s_0,\Phi,\pi) \models_{P-CTL^*} g$. This is the result of Lemma 5. Thus P_σ-CTL* $\succeq_{equalSyntax,equalPolicy}$ π_σ-CTL*.

Since $G_{\pi_\sigma - CTL^*} \subseteq G_{P_\sigma - CTL^*}$, we know P_σ-CTL* $\neq_{equalSyntax,equalPolicy}$ π_σ-CTL*, thus P_σ-CTL* $\succ_{equalSyntax,equalPolicy}$ π_σ-CTL*. □

According to Proposition 31, we know all goals expressible in π_σ-CTL* are expressible in P_σ-CTL*. We now show there is a goal in P_σ-CTL* but not in π_σ-CTL*.

Lemma 6. *Consider Φ_1, Φ_2 in Figure 3.4, and $\pi = \{(s_1, a_2), (s_1, s_2, a_2), (s_1, s_2, s_2, a_2), \cdots\}$.*

(i) For any state formula φ in π_σ-CTL*, $(s_1, \Phi_1, \pi) \models \varphi$ iff $(s_1, \Phi_2, \pi) \models \varphi$.

(ii) For any path formula ψ in π_σ-CTL* and any path σ in Φ_1 (or Φ_2) $(s_1, \Phi_1, \pi, \sigma) \models \psi$ iff $(s_1, \Phi_2, \pi, \sigma) \models \psi$.

Proof. The proof is based on the induction on the depth of formulas.

Base case: It is easy to see that (i) and (ii) are true for formulas of depth 1.

Induction: Assume that (i) and (ii) are true for formulas of depth less than *n*, and show that (i) and (ii) are true for formulas of depth *n*.

Consider state formulas of depth *n*. It can be of the following forms: (a) $sf_1 \wedge sf_2$ (b) $sf_1 \vee sf_2$ (c) $\neg sf_1$ (d) $\mathsf{E}pf$ (e) $\mathsf{A}pf$ (f) $\mathsf{E}_{pol}pf$ (g) $\mathsf{A}_{pol}pf$, where sf_1, sf_2 and pf have depth less than *n*.

Consider (d) $\mathsf{E}pf$. By definition, $(s_1, \Phi_1, \pi) \models \mathsf{E}\ pf$ iff there exists a path σ in Φ_1 starting from s_1 such that $(s_1, \Phi_1, \pi, \sigma) \models pf$. *It is observed that σ is a path starting from s_1 in Φ_1 iff σ is a path starting from s_1 in Φ_2.* Since depth of pf is less than *n*, by induction hypothesis, $(s_1, \Phi_1, \pi, \sigma) \models pf$ iff $(s_1, \Phi_2, \pi, \sigma) \models pf$. Hence, $(s_1, \Phi_1, \pi, \sigma) \models \mathsf{E}pf$ iff $(s_1, \Phi_2, \pi, \sigma) \models \mathsf{E}pf$.

The proofs for formulas of other forms are similar.

Consider path formulas of depth *n*. It can be of the following forms: (a) $pf_1 \wedge pf_2$ (b) $pf_1 \vee pf_2$ (c) $\neg pf_1$ (d) $pf_1\ \mathsf{U}\ pf_2$ (e) $\bigcirc pf_1$ (f) $\Diamond pf_1$ (g) $\Box pf_1$, where depth of pf_1 and pf_2 are less than *n*. The proof of each of these cases is similar to the proof for state formulas. □

Proposition 34. *There exists a goal in P_σ-CTL* which cannot be expressed in π_σ-CTL*.*

Proof. Consider the following goal G:

"*All along your trajectory*

<u>if</u> from any state p can be achieved for sure

<u>then</u> the policy being executed must achieve p,

<u>else</u> the policy must make p reachable from any state in the trajectory."

It can be expressed in P_σ-CTL* as $\mathsf{A}_{pol}\Box((\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p \Rightarrow \mathsf{A}_{pol}\Diamond p)\wedge(\neg\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond p \Rightarrow \mathsf{A}_{pol}\Box(\mathsf{E}_{pol}\Diamond p)))$. Assume that G can be expressed in π_σ-CTL* and let φ_G be its encoding in π_σ-CTL*.

Consider Φ_1 and Φ_2 as described in Lemma 6, and
$\pi = \{(s_1,a_2),(s_1,s_1,a_2),(s_1,s_2,a_2),(s_1,s_1,s_1,a_2),(s_1,s_1,s_2,a_2),(s_1,s_2,s_2,a_2),\cdots\}$.
The policy π is a policy for goal G and initial state s_1 with respect to Φ_2 as neither from s_1 nor from s_2, p can be achieved for sure (by any policy), and π makes p reachable from any state in the trajectory.
Thus, $(s_1,\Phi_2,\pi) \models \varphi_G$. (1)

But π is not a policy for goal G and initial state s_1 with respect to Φ_1 as from s_1, p is guaranteed achievable by another policy $\pi_2 = \{(s_1,a_1),(s_1,s_2,a_2),(s_1,s_2,s_2,a_2),\cdots\}$. With the policy π, we cannot guarantee the achievement of p.
Thus, $(s_1,\Phi_1,\pi) \not\models \varphi_G$. (2)

Lemma 6 contradicts with (1) and (2). Hence, the assumption that G can be expressed in π-CTL* is wrong. G cannot be expressed in π-CTL*. □

Compare Languages having the Same Syntax

We now compare languages that have the same syntax but with different definition of the policy structure.

Given a transition graph and an initial state, goal formulas defined in π-CTL*

and π_σ-CTL* are the same. For each policy in π-CTL*, there is a policy in π_σ-CTL* that maps to the same set of goal formulas. That is, if we restrict the set of policies in π_σ-CTL*, it can be isomorphic to the one of π-CTL*. We will also prove that it is not the case for P-CTL* and P_σ-CTL*.

Proposition 35. *π_σ-CTL* $\succ_{equalSyntax}$ π-CTL*.*

Proof. To prove that π_σ-CTL$^* \succeq_{equalSyntax} \pi$-CTL*, we need a one-to-one mapping ψ from $P_{\pi_\sigma-CTL^*}(\Phi)$ to $P_{\pi-CTL^*}(\Phi)$ for any Φ such that for all s_0, and π, we have $Gset(\pi, s_0, \Phi, \models_{\pi-CTL^*}) = Gset(\psi(\pi), s_0, \Phi, \models_{\pi_\sigma-CTL^*}) \cap G_{\pi-CTL^*}$. Since $G_{\pi-CTL^*} = G_{\pi_\sigma-CTL^*}$, we need to prove that $Gset(\pi, s_0, \Phi, \models_{\pi-CTL^*}) = Gset(\psi(\pi), s_0, \Phi, \models_{\pi_\sigma-CTL^*})$. That is, for each policy *pi* in $\pi - CTL^*$, there is one policy π' in $\pi_\sigma - CTL^*$ such that $(s_0, \Phi, \pi) \models_{\pi-CTL^*} g$ iff $(s_0, \Phi, \pi') \models_{\pi_\sigma-CTL^*} g$, and two different πs map to two different π's. We define the mapping ψ such that for a policy $\pi \in \pi$-CTL* that is a mapping from states to actions, we construct a policy that is a mapping from state sequences to actions such that these two policies correspond to the same set of trajectories from the initial state. They are satisfied by the same set of goal formulas in these two languages.

To prove that π-CTL$^* \not\succeq_{equalSyntax} \pi_\sigma$-CTL*, we need to find a policy $\pi \in P_{\pi_\sigma-CTL^*}$, a transition function Φ, and a state s_0, such that there is no $\pi' \in P_{\pi-CTL^*}$ where $Gset(\pi, s_0, \Phi, \models_{\pi_\sigma-CTL^*}) = Gset(\pi', s_0, \Phi, \models_{\pi-CTL^*}) \cap G_{\pi_\sigma-CTL^*} = Gset(\pi', s_0, \Phi, \models_{\pi-CTL^*})$.

In the transition function denoted by Figure 3.5, the policy π_2 in π_σ-CTL* such that $\{\pi_2(s_1) = a_1; \pi_2(s_1, s_2) = a_2; \pi_2(s_1, s_2, s_1) = a_3; \cdots\}$ satisfies the goal $g_3 = \mathsf{A}_{pol}\Diamond(p \wedge \Diamond q)$ thus $g_3 \in Gset(\pi_2, s_0, \Phi, \models_{\pi_\sigma-CTL^*})$. By enumerating all policies π' in π-CTL*, we know that there is no policy π' in π-CTL* such $g_3 \in Gset(\pi', s_0, \Phi, \models_{\pi-CTL^*})$. Thus π-CTL$^* \not\succeq_{equalSyntax} \pi_\sigma$-CTL*. □

From the proof, we know that a goal $\mathsf{A}_{pol}\Diamond(p \wedge \Diamond q)$ in π_σ-CTL* cannot be ex-

pressed in π-CTL*. From Proposition 30, we know all goals that can be represented in π-CTL* can be represented in π_σ-CTL*.

Proposition 36. *P_σ-CTL* $\not\succeq_{equalSyntax}$ P-CTL*. P-CTL* $\not\succeq_{equalSyntax}$ P_σ-CTL*.*

Proof. To prove that P_σ-CTL* $\not\succeq_{equalSyntax}$ P-CTL*, we need to find a policy $\pi \in P_{P-CTL^*}$, a transition function Φ, and a state s_0, such that there is no $\pi' \in P_{P_\sigma-CTL^*}$ where $Gset(\pi, s_0, \Phi, \models_{P-CTL^*}) = Gset(\pi', s_0, \Phi, \models_{P_\sigma-CTL^*}) \cap G_{P-CTL^*}$.

Let the policy π be $\{\pi(s_1) = nop\}$, the transition function be that corresponding to Figure 3.5, the initial state $s_0 = s_1$. It is easy to know that the goal $g_1 = \mathsf{A}_{pol}\Box(\neg\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond(p \wedge \Diamond q))$ is in $Gset(\pi, s_0, \Phi, \models_{P-CTL^*})$. We now prove that no policy $\pi' \in P_{P_\sigma-CTL^*}$ satisfies this goal. According to the definition of language P_σ-CTL*. $(s_0, \Phi, \pi) \models_{P_\sigma-CTL^*} g_1$ if $\neg\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond(p \wedge \Diamond q)$ is true in all states reachable from the initial states by following the policy. However, the initial state s_1 is one state in the policy. There is a policy in P_σ-CTL* that does not satisfy $\neg\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond(p \wedge \Diamond q)$. Such a policy is π_2 such that $\{\pi_2(s_1) = a_1; \pi_2(s_1, s_2) = a_2; \pi_2(s_1, s_2, s_1) = a_3; \cdots\}$. Thus there is no policy satisfies g_1 in P_σ-CTL*. P_σ-CTL* $\not\succeq_{equalSyntax}$ P-CTL*.

To prove that P-CTL* $\not\succeq_{equalSyntax}$ P_σ-CTL*, we need to find a policy $\pi \in P_{P_\sigma-CTL^*}$, a transition function Φ, and a state s_0, such that there is no $\pi' \in P_{P-CTL^*}$ where $Gset(\pi, s_0, \Phi, \models_{P_\sigma-CTL^*}) = Gset(\pi', s_0, \Phi, \models_{P-CTL^*}) \cap G_{P_\sigma-CTL^*} = Gset(\pi', s_0, \Phi, \models_{P-CTL^*})$.

The policy π_2 in P_σ-CTL* such that $\{\pi_2(s_1) = a_1; \pi_2(s_1, s_2) = a_2; \pi_2(s_1, s_2, s_1) = a_3; \cdots\}$ satisfies the goal $g_3 = \mathsf{A}_{pol}\Diamond(p \wedge \Diamond q)$ thus $g_3 \in Gset(\pi_2, s_0, \Phi, \models_{P_\sigma-CTL^*})$. We now prove that there is no policy π' in P-CTL* such $g_3 \in Gset(\pi', s_0, \Phi, \models_{P-CTL^*})$. This can be proved by enumerating all policies π' in P-CTL*. Thus P-CTL* $\not\succeq_{equalSyntax}$ P_σ-CTL*. □

The above result is mildly surprising. From the proof, we have that the goal

$\mathsf{A}_{pol}\Box(\neg\mathscr{E}\mathscr{P}\mathsf{A}_{pol}\Diamond(p \wedge \Diamond q))$ in P-CTL* cannot be expressed in P_σ-CTL*. Similarly, the goal $\mathsf{A}_{pol}\Diamond(p \wedge \Diamond q)$ in P_σ-CTL* cannot be expressed in P-CTL*.