THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITE DE TOULOUSE

Délivré par l'Université Toulouse Capitole

École doctorale : **Mathématiques, Informatique et Télécommunications de Toulouse**

Cotutelle internationale avec **Western Sydney University, Australie**

Présentée et soutenue par

XIAO Zhanhao

le 12 décembre 2017

Raffinement des Intentions - Refinement of Intentions

Discipline : **Informatique**

Spécialité : **Intelligence Artificielle**

Unité de recherche : **IRIT (UMR CNRS 5505)**

Codirecteur de thèse : M. Andreas HERZIG, Directeur de Recherches, CNRS

Codirecteur de thèse : M. Dongmo ZHANG, Associate Professor, Western Sydney University

JURY

Rapporteurs M. Brian LOGAN, Associate Professor, Univ. of Nottingham, Royaume-Uni
M. Kewen WANG, Professor, Griffith Univ., Australie

Membres Mme Célia DA COSTA PEREIRA, Maître de Conférences, Univ. de Nice
M. Malik GHALLAB, Directeur de Recherches, LAAS
M. Laurent PERRUSSEL, Professeur, Univ. Toulouse Capitole
M. Abdallah SAFFIDINE, Research Associate, Univ. of New South Wales, Australie
M. Bruno ZANUTTINI, Maître de Conférences, Univ. de Caen

Declaration of Authorship

I, Zhanhao Xiao, declare that this thesis titled, 'Refinement of Intentions' and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a research degree at the two universities.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at the two universities or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Zhanhao Xiao

Date: 18 OCT 2017

Thesis co-supervisors
Andreas Herzig (UT)
Laurent Perrussel (UT)
Dongmo Zhang (WSU)

Author
Zhanhao Xiao

Refinement of Intentions

Abstract

Keywords: BDI logics, Intentions, Refinement, Automated Planning, Hierarchical Task Network Planning

The mental attitudes, such as belief, desire, and intention, play a central role in the design and implementation of autonomous agents. In 1987, Bratman proposed a so-called belief-desire-intention (BDI) theory which inspired a multitude of BDI logics and BDI architectures. Bratman highlighted the fundamental role of an agent's future-directed intentions: they are high-level plans to which the agent is committed. Such high-level plans cannot be executed directly: they have to be stepwise refined into more elaborated plans. Ideally the plans at the end of the refinement process contain only basic actions which the agent can perform directly. The process of intention refinement is crucial to the BDI theory and every step of refinement establishes a means-end relation between the refined intentions and the intentions refining it. However, there are very few accounts of hierarchical intention refinement in the existing BDI logics.

The concept of automated planning is central in Bratman's theory. As a traditional research community of artificial intelligence, automated planning has attracted numerous attentions and it has already been integrated into BDI architectures. In particular, the way of generating solutions in hierarchical task network (HTN) planning is refining high-level actions step-by-step until basic actions. Thus, the idea of HTN planning should be revelent for BDI agents. However, except for the line of work of de Silva and Sardina *et al.*, hierarchical intention refinement has been scarcely considered in the existing BDI agents.

The aim of this thesis is to provide a logic-based comprehensive analysis of the hierarchical refinement process. In this thesis, intention refinement is considered from two perspectives: the entailment and the primitive. From the former, intentions are managed in a co-called belief-intention database where refinement is addressed via the logical entailment based on the action law; while from the latter, refinement is defined in an explicit and static way, just as in HTN planning.

From the entailment perspective, this thesis starts with an extension of Shoham's database framework on beliefs and intentions by introducing the notions of high-level intentions and environmental events. In a belief-intention database, a set of intentions can refine an intention if the set is minimal and suffices to guarantee the satisfaction of the intention. This thesis also investigates the complexity of the decision problems of satisfiability, consequence, refinement, instrumentality in the proposed database framework, which are all **PSPACE**-complete. Furthermore, this thesis explores the relations between the database framework and two logics: propositional linear temporal logic and dynamic logic with propositional assignment. The reductions to these two logics contribute to finding a set of intentions to refine a high-level intention via invoking the automated tools for these two logics.

From the primitive perspective, this thesis provides a logical semantics for HTN planning based on propositional dynamic logic. In the dynamic framework, refinement is captured by the program inclusion operator. By equipping high-level actions with pre- and postcondition, a coherence condition for HTN domains is proposed to evaluate the correctness of the predefined refinement methods. Moreover, the postulates of soundness and completeness for high-level actions are given under the dynamic semantics. When it comes to the completeness, it is usually a big challenge to define all possible refinement methods in HTN planning. It is promising to relax some restrictions on solutions: Geier & Bercher's HTN planning with task insertion (TIHTN) allows solutions obtained by inserting actions. To capture the pre- and postcondition of actions, this thesis further extends TIHTN by introducing state constraints and calls the extension TIHTNS planning. It has been shown that the property of acyclic decomposition still holds in TIHTNS planning and then an acyclic progression operator for finding a plan is proposed. Based on the progression operator, it is proved that the additional consideration of state constraints does not increase the complexity of the plan-existence problem, staying in 2-**NEXPTIME**-complete.

Raffinement des Intentions

Résumé

Mots-clés: Logique BDI, Intentions, Raffinement, Planification Automatisée, Planification Hiérarchique

La conception et l'implémentation d'agents autonomes font appel aux concepts cognitifs centraux de croyances, désirs et intentions. En 1987, Bratman a proposé une théorie alliant ces trois notions et fondant la notion d'agents BDI (belief-desire-intention) aujourd'hui encore très populaire en intelligence artificielle. Les intentions des agents ont deux caractéristiques essentielles : (i) elles décrivent ce que sera le futur de l'agent, autrement dit elles peuvent être vues comme des plans ou actions de haut niveau et (ii) elles engagent l'agent à réaliser celles-ci. Ces plans de haut niveau ne peuvent toutefois pas être exécutés directement : l'agent va raffiner ceux-ci afin d'obtenir des plans plus élaborés. Cette procédure de raffinement doit être comme un processus hiérarchique allant d'un niveau abstrait très général à un niveau détaillé et concret. L'aboutissement de ce raffinement est l'obtention d'un ensemble d'intentions basiques représentées par des actions directement exécutables. Cette thèse a pour objectif de proposer une analyse logique de ce raffinement, thème qui a été peu abordé dans les logiques BDI.

La notion de planification est centrale selon Bratman et quelques travaux ont montré comment la planification automatique pouvait être intégrée à un agent BDI. La planification hiérarchique (Hierarchical Task Network ou HTN planning) permet une génération de plans selon une approche allant d'actions de haut niveau vers des actions de bas niveau. Exceptés les travaux de da Silva, Sardina *et al.*, très peu de travaux ont exploré le rapprochement entre planification HTN et raffinement d'intentions.

Dans cette thèse, nous considérons une double perspective sur le raffinement d'intentions: raffinement "par implication" et raffinement "déclaratif". Dans la perspective "par implication", les intentions sont représentées par une "base de données" de croyances et d'intentions et la relation de raffinement est déduite à partir des lois d'actions. Dans la perspective "déclaratif", le raffinement est explicitement défini de manière similaire à ce qui est fait dans la planification hiérarchique.

Dans la perspective “par implication”, le point de départ est une extension des bases de données “croyances-intentions” proposée par Shoham. Cette approche est plus simple que les formalismes logiques basés sur les logiques modales mais plus riche que les formalismes ayant pour objectif l’implémentation d’agents BDI. Notre première contribution est l’introduction de hiérarchie entre intentions afin d’expliciter la notion de raffinement. Nous avons aussi considérer l’agent et son environnement, en proposant d’introduire dans la base de données la notion d’évènements. Nous avons ensuite logiquement analysé le raffinement et proposé une analyse de la complexité des problèmes de décision pour la satisfaction, conséquence, raffinement et instrumentalité. Nous montrons que tous ces problèmes de décision sont **PSPACE**-complete. L’analyse logique est aussi effectuée en montrant comment la base de données-peut être encodées en logique temporelle et logique dynamique. Les réductions dans ces logiques contribuent à résoudre le problème du choix du raffinement en profitant des outils logiciels pour ces logiques.

Dans la perspective “déclaratif”, nous proposons dans un premier temps une caractérisation d’un fragment de la planification hiérarchique en logique dynamique propositionnelle: les actions sont vues comme des programmes et les méthodes sont caractérisées à l’aide d’un opérateur d’inclusion entre programmes. A l’aide de cette représentation en logique, nous pouvons exprimer des notions sémantiquement riches comme la modularité ainsi que l’expression de pre et post conditions pour les actions de haut niveau. Nous montrons que cette formalisation en logique dynamique permet d’exprimer les notions d’adéquation et complétude des méthodes HTN, notions difficiles à exprimer avec une sémantique opérationnelle.

Dans un second temps, nous abordons le problème de la définition préalable de l’ensemble des décompositions possibles (bibliothèque de méthodes). Cette étape est cruciale et délicate à effectuer. La planification HTN avec insertion de tâches (TIHTN) permet de répondre partiellement à cette difficulté en permettant l’obtention des solutions de décomposition qui n’auraient pas été préalablement encodées dans la bibliothèque de méthodes. Cela est possible en insérant dynamiquement des tâches dans les méthodes de décomposition prédéfinies. Dans cette thèse, nous proposons une extension de la planification TIHTN par la prise en compte des pre et post conditions sur les actions de haut niveau. Nous montrons que la complexité pour l’existence d’un plan n’est pas impactée par ces contraintes d’état et reste 2-**NEXPTIME**-complet. Cela est démontré à l’aide d’un opérateur de progression acyclique.

Acknowledgements

I wish to express my special gratitude to my principal supervisor Dr. Andreas Herzig, for his continuous support of my PhD study and related research, for his patience, motivation, and immense knowledge. I really appreciate him for encouraging my research and for allowing me to grow as a research scientist. I have learnt from him to become more optimistic and more confident to face the life.

I would also like to thank my co-supervisor Prof. Laurent Perrussel for his tremendous guidance on my research and for his generous assistance on my daily life in France. He is always humorous, penitent and kindly. I have learnt great deal from him, not only in research, but also a lot which will guide my whole life.

I am also sincerely grateful to my another co-supervisor A/Prof. Dongmo Zhang who has been an excellent mentor for me. He is always patient and careful when we are discussing. I thank him for his insightful guidance on my work and for his enthusiastic support on my stay in Sydney.

A special thanks to my family. Words cannot express how thankful I am to my mother, my father and my sister for all of the sacrifices that they have made on my life.

I want to thank the teachers from our research group – LILaC, especially Philippe Balbiani, Joseph Boudou, Yannick Chevalier, Sylvie Doutre, Umberto Grandi, Emiliano Lorini and Jean-Marc Thévenin. I also need to appreciate my colleagues Martin Dieguez, Julien Hay, Arianna Novaro, Christos Rantsoudis, Maryam Rostamigiv, Maël Valais and Julien Vianey for their help in my stay in France.

I would like to thank Yun Bai, Guifei Jiang, Yan Zhang who have been helpful when I stayed in Sydney.

I thank my dearest friends: Lunde Chen, Zhouye Chen, Yu Chen, Iris, Jiahui Lai, Aiden Lo, Yifei Li, Usman Younus Muhammad, Jiahui Rao, Tongming Wang, Ping Xie, Jiaxin Zhao for their companion, caring and sharing, which made my life in Sydney and Toulouse memorable.

Finally, I am grateful for the scholarships from Chinese Scholarship Council (CSC) and Western Sydney University, which have supported my PhD research, and to

University of Toulouse and Western Sydney University for providing me with a wonderful learning environment. With the help of my supervisors, I have also profoundly benefited from the Cotutelle Program between the two universities.

Zhanhao Xiao

Toulouse

Contents

List of Figures

List of Tables

Abbreviations

HTN	**H**ierarchical **T**ask **N**etwork
TIHTN	**H**ierarchical **T**ask **N**etwork with **T**ask **I**nsertion
TIHTNS	**H**ierarchical **T**ask **N**etwork with **T**ask **I**nsertion and **S**tate **C**onstraints
HGN	**H**ierarchical **G**oal **N**etwork
HR-HGN	**H**ierarchy-**R**elaxed **H**ierarchical **G**oal **N**etwork
GTN	**G**oal **T**ask **N**etwork
PLTL	**P**ropositional **L**inear **T**emporal **L**ogic
DL-PA	**D**ynamic **L**ogic with **P**ropositional **A**ssignement
PDL	**P**ropositional **D**ynamic **L**ogic
ASP	**A**nswer **S**et **P**rogramming
NNF	**N**egation **N**ormal **F**orm

Chapter 1

Introduction

1.1 Motivation

A mental state is a state of mind that an agent is in.[1] The fundamental mental states include the attitudes of beliefs, desires, and intentions, which play a central role in the design and implementation of autonomous agents. Beliefs have a 'mind-to-world' direction of fit: agents try to adapt their beliefs to the truths of the world, while intentions have a 'world-to-mind' direction of fit: agents try to make the world match their goals. These concepts have been well studied in philosophy, psychology and cognitive science but were new to computer scientists until being introduced by the researchers of artificial intelligence and multi-agent systems.

In 1987, Bratman proposed a so-called belief-desire-intention (BDI) theory [Bratman, 1987] which inspired a multitude of BDI logics and BDI architectures. Bratman highlighted the fundamental role of an agent's future-directed intentions: they are *high-level plans* to which the agent is *committed*. Being high-level plans, intentions typically cannot be executed directly: they have to be *refined* as time goes by, resulting in more and more elaborate plans. Ideally the plans at the end of the refinement process contain only *basic actions*, which are the actions the agent can directly execute. The process of intention refinement is crucial to

[1] Refer to https://en.wikipedia.org/wiki/Mental_state (accessed on 27 Sep. 2017).

the BDI theory. As pointed out in [Rao and Georgeff, 1991], "the potential of non-primitive events for decomposition into primitive events can be used to model hierarchical plan development". For example, assume I have a high-level intention to go to Melbourne. In order to achieve this intention, I further refine it and I decide to fly to Melbourne by plane. The intention to fly to Melbourne can be refined into going to the airport and taking the plane. Figure 1.1 shows such a process of refinement.

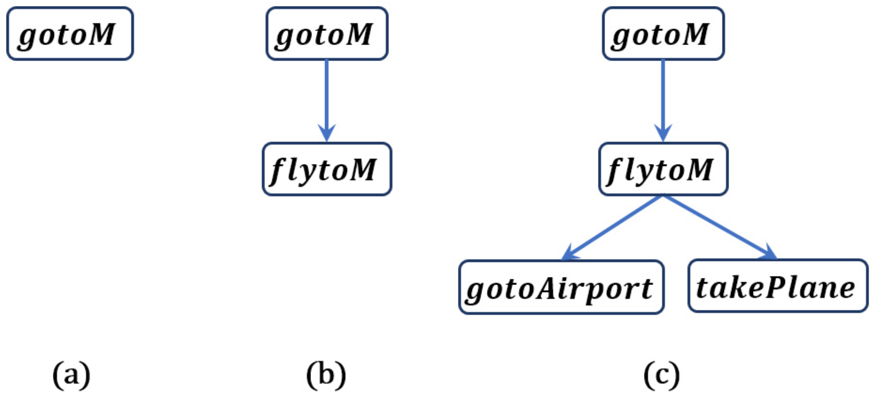

FIGURE 1.1: An Example of Intention Refinement

Forming future-directed intentions enables agents to extend the influence of their deliberations beyond the present moment. This is important given that the cognitive capacities and time for deliberation are limited or the environment in which the agent stay is frequently changing. Specifically, it would be the case that at some moment an agent has little time to deliberate and think through all her options. For example, I may decide on Sunday what to do for the next weekend since I know that I will be very busy during weekdays and will have no time to make my plan for the weekend.

Intention refinement should be a central ingredient of any theory of autonomous agents. However, there are few accounts of hierarchical intention refinement in the existing BDI logics or BDI agents. The aim of this thesis is to provide a logic-based comprehensive analysis of the hierarchical refinement process.

A notable exception that considers intention refinement is the strand of work related the agent programming language CANPLAN [Sardina et al., 2006] which import ideas from hierarchical task network (HTN) planning. HTN planning concerns a hierarchy over actions ('tasks') that are either basic ('primitive') or high-level ('compound') [Erol et al., 1994a]. The HTN solutions are generated by decomposing higher-level actions step-by-step into lower-level actions. This decomposition process actually is the refinement process from higher-level intentions towards lower-level intentions, until basic actions. In HTN planning, the decomposition of actions is restricted in an explicit and predefined way where the step-by-step decomposition is done according to the given decomposition methods. Every decomposition method actually defines a refinement relation between a high-level action and a set of actions which are called subtasks. It requires that the HTN domain designer considers all possible decomposition methods for all actions and provides the complete domain knowledge, which depends on the expertise of the designer. However, it is always a big challenge for the HTN domain designer. As pointed out in [Kambhampati et al., 1998], "most real-world domains tend to be 'partially hierarchical' in that human expertise, in the form of task reduction knowledge, exists for only some parts of the domain." In the case of numerous actions, the refinement relation may be hidden among various actions and it is easy to neglect some subtasks which are necessary or helpful to accomplish the high-level actions. Taking the example of Figure 1.1, an action of buying the flight ticket may be forgotten in the decomposition method with respect to the action of flying to Melbourne. Formalizing the refinement relation in a logical framework, which is the aim of this thesis, contributes to finding the potential refinement relation. The logic-based analysis of the refinement relation would assist the HTN domain designer to improve the HTN domain: the discovery of the implicit refinement relation between actions provides the clues for the decomposition methods. Additionally considering the action of buying the ticket as a subtask is necessary to achieve the action of flying to Melbourne.

When we start to build up a logical framework for the refinement of intentions, there are several questions occurring to us:

1. *How can we model high-level intentions?*

2. *How to model the refinement of a high-level intention?*

3. *How to refine an intention into executable plans?*

4. *Is the refinement operation correct?*

The above questions are essential for the operation of intention refinement. We will keep these questions throughout the thesis and answer them one by one in the end of this thesis.

1.2 BDI Logics and BDI Implementations: A Summary of the State of the Art

Since Bratman's BDI theory [1987], numerous approaches were built on the BDI paradigm, both practical (BDI architectures and BDI agents) and theoretical (BDI logics). In this section, we first introduce the two main BDI logics and then summarize the shortcomings of the existing BDI agent languages.

1.2.1 Cohen & Levesque's Logic and Rao & Georgeff's Logic

The logical approaches that were most influential are due to [Cohen and Levesque, 1990] and [Rao and Georgeff, 1991].

Cohen & Levesque's Linear Time Logic Cohen and Levesque [1990][2] provided a seminal logical modeling of Bratman's BDI model. Their approach accounts for achievement intentions (as opposed to maintenance intentions). It distinguishes intention-to-do and intention-to-be and mainly focuses on the latter. The definition of intention-to-be comes in four steps—chosen goals, achievement

[2]The paper was awarded the IFAAMAS most influential paper award in 2006.

goals, persistent goals and intentions—that are couched in a quantified modal logic of linear time, action, and belief.

1. Chosen goals correspond to future states where the agent would like to be.
2. Achievement goals are chosen goals that are not true yet (more precisely: that the agent believes to be false now).
3. Persistent goals are achievement goals that are only abandoned when they are either achieved, or learned to be non-achievable, or 'for some other reason'.
4. Intentions are persistent goals for which the agent is prepared to act; this excludes persistent goals to which the agent cannot contribute anything, such as my persistent goal that there be snow at Christmas.

While Cohen & Levesque's approach is much cited, it is fair to say that it is rather complicated. Some early criticisms of technical details can be found in [Singh, 1992]. In Shoham and Leyton-Brown's textbook [2008], the approach to model intentions is called "the road to hell". It speaks for itself that its mathematical properties—such as axiomatizability, decidability and complexity of fragments—were never investigated. Cohen & Levesque's approach moreover has three major shortcomings. First, it does not provide a solution to the frame problem[3]: what is true at different time points t and t' may vary wildly and is not determined by the actions occurring between t and t'. Second, it does not account for intention refinement. Third, it does not fully account for revision; indeed, while Cohen & Levesque provide some criteria for the abandonment of intentions through the notion of *rational balance* (forbidding to intend something that is true or believed to be impossible to achieve), it does not further analyze the 'other reasons' for which a persistent goal is abandoned. These reasons should mainly cover abandonment of goals that are instrumental for another, higher-level goal that is dropped, and more generally intention reconsideration.

[3] The frame problem, one of the main and oldest problems in reasoning about actions, concerns the specification of the effects of actions [McCarthy and Hayes, 1969]. The main challenge is to characterize these effects without explicitly specifying which conditions are not affected by executing actions.

Rao & Georgeff's Branching Time Logic Contrarily to Cohen & Levesque's logic, [Rao and Georgeff, 1991] embrace a primitive notion of intention in their branching time logic which is based on computational tree logic CTL*.

Just as Cohen & Levesque's approach, Rao & Georgeff's suffers from the shortcomings that we have listed above: intention revision is basically absent from the picture and the frame problem is not solved. Indeed, due to the temporal logic framework agents can perform actions whose effects are not further specified. It is also not described how beliefs are preserved while agents act.

Rao & Georgeff's approach was fleshed out by Michael Winikoff *et al.* [2002] who link intentions[4] to the actions associated to them by means of transition rules in a predefined way. The logical framework they propose, called Conceptual Agent Notation, is defined in terms of a declarative and an operational semantics. Together, they allow to reason about the relations between goals, such as dependence, mutual consistency, and mutual support. Overall, the framework is rather complex and, just as all other existing BDI logics, the frame problem remains unsolved: the framework describes how sub-goals may be inferred (with respect to some library of plans) but does not keep track of these steps. In other words, no means-end relation between the ongoing goals can be exhibited and consequently revision cannot be handled in a rational way.

1.2.2 BDI Implementations and Their Shortcomings

After Bratman's book [1987], the BDI paradigm inspired a multitude of models and platforms aiming at the implementation of software agents: so-called *BDI agents*, such as LORA [Wooldridge, 2000], KARO [Meyer et al., 2001], 3APL [Dastani et al., 2003], dMars [d'Inverno et al., 2004], AgentSpeak-Jason [Bordini and Hübner, 2010], GOAL [Hindriks et al., 2012], CANPLAN [Sardina et al., 2006, Sardina and Padgham, 2011] and its extension CANPLAN+ [Bauters et al., 2014a]. All these software platforms are made up of a 'B', a 'D', and an 'I' component

[4] They use the term goals.

that are interfaced appropriately. Such architectures are inspired by the Intelligent Resource-bounded Machine Architecture proposed by [Bratman et al., 1988a].

1.2.2.1 Lack of Formal Logical Semantics

Most BDI software models and platforms are semi-formal: while they provide a taxonomy of basic concepts and their relationships, the agent programming languages are usually equipped with an operational semantics only and lack a formal logical semantics. Typically, they support the specification of BDI agents with respect to some specific BDI implementations. For instance, the language of AgentSpeak [Bordini et al., 2007] enables to express what are the initial beliefs, actions and plans available in an AgentSpeak-Jason implementation of a multi-agent system. More generally speaking, there are only few attempts to formally relate BDI implementations and BDI logics. For example, AgentSpeak does not enable reasoning about the consequences of an action. The main exception is the line of work on the KARO framework [Hindriks and Meyer, 2006, Hustadt et al., 2001, Meyer et al., 2001]. However, it seems to be fair to say that this logic and its mathematical properties are not well understood yet.

A further weak point of BDI architectures is that their associated agent language is often severely restricted: it consists of literals, i.e., propositional variables or their negations. Typically, the agent language dMars requires that beliefs are only sets of literals. In our view this is a major obstacle to the use of BDI agents, for two reasons. First, it does not allow second-order beliefs, i.e., beliefs about other agents' beliefs. Such beliefs—and more generally higher-order beliefs—are however central for the reasoning of a socially intelligent agent. Their fundamental role in human intelligence was highlighted in experiments such as false belief tasks [Bolander, 2014]. In Game Theory, higher-order beliefs are at the heart of the definition of notion of equilibrium as each agent has to assume that the other agents are rational [Lorini and Moisan, 2011, Strzalecki, 2014, Weinstein and Yildiz, 2007].

Second, except for some agent languages that allow disjunctions, such as 3APL, KQML [Labrou and Finin, 1994] and FIPA-ACL [O'Brien and Nicol, 1998], a big part of them do not allow disjunctions. For example, AgentSpeak does not allow to express disjunctive beliefs. This is clearly a disadvantage: for example, goals to *know whether* some proposition is true cannot be expressed. Furthermore, this is highly problematic if one wants to employ BDI agents as conversational agents. For instance, agent i's yes-no question whether φ is conditioned by i's goal to know whether φ is true. Another example is that i's speech act of informing j that φ is conditioned by i's belief that j does not know whether φ. These situations are quite common in game playing.

1.2.2.2 Lack of Intention Refinement

As we have said, the concept of intention refinement is crucial in the BDI theory. However, the literature on BDI logics and BDI agents only contain few contributions to intention refinement. Indeed, from [Ingrand et al., 1992] to recent work [Waters et al., 2015], mainstream implementations of BDI-agents have adopted plan libraries: functions associating to each intention the set of plans that can achieve it. Actually, the inter-relationships between intentions are taken into account for handling plan execution failures and avoiding conflicts [Clement and Durfee, 1999a,b, Shaw and Bordini, 2007, Yao et al., 2016]. However, the existing approaches on BDI agents do not formalize the refinement relation among intentions which is a kind of means-end relation. This is also the case even when the focus is on the dynamics of the intention base [Schut et al., 2004]. We believe that this is a major shortcoming of such approaches.

As mentioned above, a notable exception is the line of work of de Silva and Sardina *et al.* [de Silva, 2017, de Silva and Padgham, 2004, de Silva et al., 2009, Sardina et al., 2006] which import ideas from HTN planning. In [de Silva and Padgham, 2004], the authors showed through experiments that BDI systems are more suitable when facing highly dynamic environments, while HTN solutions are more efficient in a static context. In [Sardina et al., 2006], Sardina *et al.* integrated a BDI agent system with a offline HTN planner as a "lookahead" component and developed a BDI agent language CANPLAN. In their architecture, an agent is allowed to perform lookahead deliberation from user-defined points in the agent's goal-plan hierarchy where an intention is a program consisting of primitive actions and operations of testing, sequence, concurrency and backup. The intention is considered to be successfully executed if its corresponding HTN network task is accomplished. Later in [de Silva et al., 2009], the authors proposed a notion of 'ideal' (precisely, minimal non-redundant maximally-abstract) plan and compute a suboptimal 'ideal' plan, which is non-redundant and preserves abstract as much as possible, based on the decomposition hierarchy of HTN planning. While [Sardina et al., 2006] showed how CANPLAN libraries is translated into equivalent HTN planning domains, [de Silva, 2017] showed the other way around. The above approaches inherently restrict intentions to be handled by an underlying predefined set of decomposition methods in a static way. However, as we have noted above, defining all possible decompositions in the beginning is a challenge for the designer as the expertise is usually partial in many real-world scenarios.

To sum up, the account of hierarchical intention refinement in the existing BDI implementations is still underdeveloped. We believe that it is crucial to incorporate means-end relations between intentions into the new generation of BDI models and that the untapped potentials in intention refinement will contribute to closing the gap between BDI theories and BDI implementations.

1.3 BDI Theories and Automated Planning

As a traditional research community of artificial intelligence, automated planning has gained numerous attentions. Being a central concept in BDI theories, automated planning naturally is connected with intention refinement. Consider an intention to achieve a goal which is in the simplest case a propositional formula: if we restrict to one step of refinement, the process of refining the intention is relegated to the planning process in classical planning.

Apart from the line of work [Sardina et al., 2006] on the integration of HTN planning into BDI agents, there exist a number of promising contributions aiming at a connection between the automated planning community and BDI agents. Conformant planning has also been considered in BDI agents [Bauters et al., 2014b, Ma et al., 2014].

When it comes to the case of multiple steps, intention refinement is similar to HTN planning. Inspired by [Sardina et al., 2006], we summarize the similar entries between BDI theories and HTN planning in Table 1.1.

TABLE 1.1: BDI Theories and HTN Planning

BDI theories	HTN planning
belief	state
intention	task
basic intention	primitive task
high-level intention	compound task
refinement	decomposition

According to this point of view, it seems to us that time has come to reconsider the link between BDI models and plan generation: the integration of HTN planning into BDI logics that we have mentioned above is a promising first step. However, decomposition methods bring a too rigid solution for defining the refinement relation between intentions: the refinement relation has to be declared explicitly in HTN planning domain. A more general perspective, such as the one offered by hybrid planning [Kambhampati et al., 1998], is to consider that high-level actions also have effects. Characterizing such effects is not trivial, as it raises the question of the main ('primary') effect of an action. As a further work, [de Silva et al., 2009] mix BDI reasoning and hybrid planning but the primary effect of an action is not clearly characterized. On the other hand, the expensive computational cost of HTN planning, which is **EXPTIME**-complete even for the propositional case with tasks being totally-ordered, stops its integration into the BDI community.

To sum up, most BDI agents, except for the line of work on CANPLAN, consider intention as an atomic concept and only consider plan libraries in a predefined way.

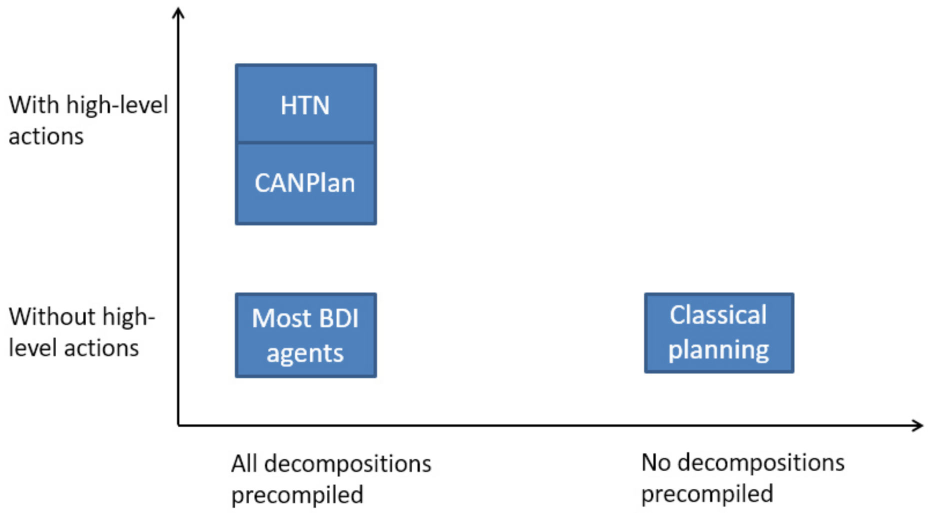

FIGURE 1.2: Comparison between BDI Agents and Automated Planning 1

On the other hand, in classical planning, there are no high-level actions, consequently no refinement. While high-level actions are considered in HTN planning, but the decomposition has to be compiled in advance. The comparison among them is shown in Figure 1.2.

Different from Cohen & Levesque's linear time logic and Rao & Georgeff's branching time logic, [Shoham, 2009] proposed an alternative and simpler account on beliefs and intentions, called *database perspective*, where intentions to do are stored in a database. The database framework actually is closely connected with automated planning where in the extreme case, the planning process is relegated to the database. Thus, we are convinced that it is a promising basis to consider Shoham's database perspective for designing a logical framework which takes intention refinement into account.

1.4 Major Contributions of the Thesis

This thesis provides a comprehensive analysis of intention refinement. We analyze intention refinement from two perspectives: the entailment and the primitive. From the entailment perspective, we consider intentions in a belief-intention database where refinement is addressed via the logical entailment based on the action law; while from the primitive perspective, refinement is defined in an explicit and static way, just as defined in HTN planning. The main contributions of this thesis are summarized as follows:

- *Establish a logical formalism for beliefs and intentions accounting for intention refinement*

  We start with an extension of Shoham's database framework on beliefs and intentions by introducing the notions of high-level intentions and environmental events. In the extended database, an intention is refinable to a set of intentions if the set is minimal and suffices to guarantee the satisfaction of the intention. Furthermore, we investigate the complexity of the decision problems of satisfiability, consequence, refinement, instrumentality in the proposed database framework, which are all **PSPACE**-complete.

- *Propose two approaches for the decision problem of refinement*

  We explore the relations between the database framework and two logics: propositional linear temporal logic and dynamic logic with propositional assignment. The reductions to these logics contribute to deciding whether an intention is refinable to an intention set via invoking the existing automated tools for these two logics.

- *Provide a dynamic logic account of HTN planning*

  As HTN planning only has an operational semantics, it is difficult to evaluate the correctness of HTN planning domains. We examine a restricted version of HTN planning in the framework of propositional dynamic logic (PDL) where refinement is captured by the program inclusion operator. In the dynamic framework, by equipping high-level actions with pre- and postcondition, we define the coherence condition of HTN planning domains to evaluate the correctness of the predefined refinement methods. Furthermore, we explore the soundness and completeness postulates of actions based on the semantics.

- *Extend HTN planning with task insertion by introducing state constraints*

When it comes to the completeness, it is usually a big challenge to define all possible refinement methods in HTN planning. It is promising to relax some restrictions on solutions: Geier & Bercher's HTN planning with task insertion (TIHTN) allows solutions obtained by inserting actions. We further extend their framework by introducing state constraints so that the pre- and postcondition of high-level actions are captured, and we call the extension TIHTNS planning. We also show that the property of acyclic decomposition holds in TIHTNS planning and then we propose an acyclic progression operator for finding a plan. Based on the progression operator, we prove that the additional consideration of state constraints does not increase the complexity of the plan-existence problem, staying 2-**NEXPTIME**-complete.

1.5 Outline of Chapters

In Chapter 2, we recall Shoham's database perspective which accounts beliefs and intentions in the databases and notions of automated planning including classical planning and HTN planning.

In Chapter 3, we follow Shoham's database perspective and extend his belief-intention database framework by introducing high-level actions and exogenous events. In the extended framework, an intentions is considered as a high-level action with a duration. Based on the databases, we define the refinement relation between higher-level and lower-level intentions. A means-end relation among intentions follows the refinement of intentions, alias instrumentality relation, linking higher- and lower-level intentions. We also investigate the complexity of the decision problems of satisfiability, consequence, refinement and instrumentality in belief-intention databases, which are all **PSPACE**-complete. This chapter is based on [Herzig et al., 2016b] and [Xiao et al., 2017b].

In Chapter 4, we embed the decision problems of satisfiability, consequence, refinement and instrumentality in the belief-intention databases into the satisfiability and validity problems of propositional linear temporal logic (PLTL) and dynamic logic with propositional assignment (DL-PA). The part about the translation to PLTL in this chapter is based on [Xiao et al., 2017b].

In Chapter 5, we propose a semantics of propositional dynamic logic for HTN planning. Based on the framework of dynamic logic, we investigate and formalize the coherence condition of HTN planning domains. We also propose three postulates for HTN planning: modularity, soundness and completeness. This chapter is based on [Herzig et al., 2016a].

In Chapter 6, we extend HTN planning with task insertion by introducing state constraints, which is called TIHTNS planning. We show that all solutions of TIHTNS planning can be obtained by decomposing the tasks acyclically and inserting tasks, entailing that it is decidable to find a solution for TIHTNS planning problem. Furthermore, we propose a progression operator of acyclic decomposition and prove that the plan-existence problem for TIHTNS planning is 2-**NEXPTIME**-complete. We also embed hierarchy-relaxed hierarchical goal network planning (HR-HGN) in TIHTNS without introducing fresh operators. This chapter is based on [Xiao et al., 2017a].

In Chapter 7, we conclude the thesis with answering the research questions and discuss the future work.

Chapter 2

Shoham's Database Perspective and Automated Planning

In the previous chapter, we have introduced the two most influential BDI logics, Cohen & Levesque's and Rao & Georgeff's, which however are rather complicated. In this chapter, we start by introducing Shoham's alternative simplified account on beliefs and intentions, called database perspective, which is simpler than the BDI logics rooted in these two logics and at the same time is more suitable for a logical analysis than the existing, heavily implementation-driven BDI agents. We also introduce two logics inspired by Shoham's database perspective.

As mentioned in the previous chapter, automated planning is a central concept in Bratman's BDI theory. In the second part of this chapter, we recall the notions of two planning approaches: classical planning and HTN planning. In particular, HTN planning is strongly connected with the theory of intention refinement where the decomposition process in HTN is actually the process of refinement. In the literature of automated planning, the object "action" is represented by different terminologies, which makes it difficult to integrate the research results of BDI theories and automated planning. To pave the avenue to combine them, we summarize and categorize the terminologies of actions.

This chapter is organized as follows. In Section 2.1, we introduce Shoham's database perspective and two logics inspired by it. In Section 2.2, we recall classical planning and HTN planning. In Section 2.3, we give the uniform terminology of actions and in Section 2.4 we summarize the chapter.

2.1 Shoham's Database Perspective

Cohen & Levesque's commitment-based logic provided a seminal logical modeling of Bratman's BDI theory. While this approach is much cited, it is rather complicated. (it is even called "the road to hell" in [Shoham and Leyton-Brown, 2008]). None of the BDI logics that were introduced subsequently—starting with [Rao and Georgeff, 1991]—fully adopted Cohen & Levesque's definition of intention.

Almost 20 years later, Shoham argued for a simpler approach that he baptized with *database perspective* [Shoham, 2009]. His aim is to define a framework that is simpler than Cohen & Levesque's and Rao & Georgeff's and that thereby provides a more suitable basis for the design and implementation of BDI agents. Instead of expressing achievement goals by means of the temporal 'eventually' modality as Cohen and Levesque do, he assigns propositional variables to time points. The central idea is to organize beliefs and intentions, which are intentions-to-do, in two temporal databases. A belief database B is a set of pairs made up of time points t in the set of non-negative integers $\mathbb{N}^0$ and literals p.[1] They are written p_t and read "p is true at t". Similarly, an intention database I is a set of pairs made up of time points t and (basic) actions a. They are noted a_t and read "the agent intends to do action a at time t".

Generally, each action a has pre- and postcondition. They are described by functions `pre` and `post` mapping each action a to special atomic formulas[2] $\texttt{pre}(a)$ and $\texttt{post}(a)$.

[1] Shoham mentioned that a belief could be any formula indexed by multiple time values, but does not elaborate this further. Such a generalization should come with more complex notation and new semantical and computational problems.

[2] Shoham does not specify the form of precondition and postcondition in [Shoham, 2009]. But they are in the form of atomic formulas in his future work [Icard et al., 2010].

Similarly to Cohen & Levesque's notion of rational balance, Shoham requires the following *coherence* constraints: let B_t be the set of t-indexed literals of B and I_t be the set of t-indexed actions of I,

1. Every B_t is consistent;
2. Every I_t is either empty or a singleton;
3. If $a \in I_t$ then $B_t \not\models \neg\texttt{pre}(a)$;
4. If $a \in I_t$ then $B_{t+1} \models \texttt{post}(a)$.

Shoham's perspective was subsequently worked out in [Icard et al., 2010], which provided a semantics for belief-intention databases in terms of paths and an axiomatization for such belief-intention databases in terms of sets of paths. A *path* ρ associates to every non-negative integer t a set of propositional symbols and an action: the propositional symbols that are true at t and the action that is going to be performed by the agent at t. A set of paths Π is *appropriate* if

(1) on each path, the postcondition of each action performed at time t is true at time $t{+}1$and

(2) once the precondition of an action is satisfied at time t on ρ then it must be performed on some path that is identical to ρ up to time $t{-}1$.[3]

For a belief base B, Icard *et al.* define a set of paths where beliefs in B is true at time point 0, noted $\rho(B)$.

Given an appropriate set Π of paths, a modal operator of historic possibility $\Diamond$ is interpreted by:

$$\Pi, \rho, t \models_\Pi \Diamond\varphi \text{ iff } \exists \rho' \in \Pi \text{ such that } \rho \text{ and } \rho' \text{ agree up to } t \text{ and } \Pi, \rho', t \models \varphi.$$

[3] The time parameter $t{-}1$ is missing in [Icard et al., 2010].

They defined the coherence condition between belief database B and intention database I as follows: the agent considers it possible to do all actions she intends with respect to some appropriate set of paths. Formally,

$$\text{there is a } \rho \in \rho(B) \text{ such that } \rho(B), \rho, 0 \models \Diamond \bigwedge_{a_t \in I} \texttt{pre}(a)_t.$$

Based on this formalization, Icard *et al.* propose AGM-like postulates for the joint revision of beliefs and intentions and provide a representation theorem.

[van Zee et al., 2015a] recently criticized that Icard *et al.*'s logic is unsound because the axiom $\texttt{pre}(a)_t \rightarrow \Diamond do(a)_t$ is not valid. They adapted Icard *et al.*'s logic by moving to a semantics in terms of CTL*-like tree structures, plus a language with time-indexed modalities. They also provided a sound and complete axiomatization of their new logic w.r.t. the class of all models. They moreover gave an example showing that Icard *et al.*'s coherence constraint (which only considers the precondition of actions) is too weak. They proposed a stronger coherence condition where the pre- and postcondition of actions and beliefs are always jointly consistent.

Based on their logic, van Zee *et al.* focused on the AGM-like revision of beliefs about actions and time [van Zee and Doder, 2016, van Zee et al., 2015b]. They adapted the AGM semantics of belief revision by adding a condition saying that infinite models with the same finite prefix have the same priority in the revision preorder. They then proved representation theorems in the style of Katsuno-Mendelzon and Darwiche-Pearl.

According to [Shoham, 2016], the database perspective is at the heart of the Personal Time Assistant (PTA), which is a next-generation calendar helping people to manage time. His *Timeful* application has intentions as its basic concept and was developed within a start-up company that was acquired by Google in 2015.

2.2 Classical Planning & HTN Planning

In this section, we introduce two important approaches in deterministic automated planning: classical planning and Hierarchical task network (HTN) planning.

The representation of planning problems is based on notations derived from first-order logic. States are represented as sets of atoms that are true or false within some interpretation.

2.2.1 Classical Planning

There are different ways to represent classical planning problems. Here we follow the presentation given in Section 2.3 of [Ghallab et al., 2004].

Let us start with a function-free first-order language $\mathscr{L}$ whose vocabulary includes a finite set of predicates and constants and an infinite set of variables. Every term $\mathtt{t}$ of $\mathscr{L}$ consists of variables and constants.

A state is a set of ground atoms of $\mathscr{L}$. Since $\mathscr{L}$ has no functions, the set of ground atoms of $\mathscr{L}$, denoted by $\mathsf{Atm}(\mathscr{L}_0)$, is finite and thus the set S of all possible states is guaranteed to be finite. The truth conditions are defined as usual. More precisely, a ground atom p holds in s iff $p \in s$. If G is a set of literals (i.e., atoms and negated atoms), we say s satisfies G, denoted by $s \models G$, when there is an assignment μ such that every positive literal of $\mu(G)$ is in s and no negated literal of $\mu(G)$ is in s. Here we use the closed-world assumption: an atom that is not explicitly specified in a state does not hold in that state.

In classical planning, actions are described by *operators* that change the truth values of the atoms in $\mathscr{L}$. An operator o is a tuple $(\mathtt{name}(o), \mathtt{pre}(o), \mathtt{eff}^+(o), \mathtt{eff}^-(o))$ where $\mathtt{name}(o)$ is syntactically an atom, called the name of the operator; $\mathtt{pre}(o)$ is a formula in $\mathscr{L}$, called the precondition of the operator; $\mathtt{eff}^+(o), \mathtt{eff}^-(o)$ are two sets of atoms which are totally disjoint after grounding the atoms, called the positive and negative effects of the operator, respectively.

The purpose of an operator name, is to provide an unambiguous way to refer to the operator or to substitution instances of the operator without having to write the whole tuple explicitly. If o is an operator or an operator instance (i.e., a substitution instance of an action), then $\mathtt{name}(o)$ refers unambiguously to o. Thus, when it is clear from the context, we will write action $\mathtt{name}(o)$ to refer to the entire operator o.

An action is any ground instance of an operator. A set of actions O determines a state-transition function $\gamma : 2^{\mathsf{Atm}(\mathscr{L}_0)} \times O \longrightarrow 2^{\mathsf{Atm}(\mathscr{L}_0)}$, where:

- $\gamma(s, o)$ is defined iff $s \models \mathtt{pre}(o)$;
- $\gamma(s, o) = (s \setminus \mathtt{eff}^-(o)) \cup \mathtt{eff}^+(o)$ if $\gamma(s, o)$ is defined.

A sequence of actions $\langle o_1, \dots, o_n \rangle$ is executable in a state s_0 iff there exists a sequence of states $s_1, \dots, s_n$ such that for all $1 \leq i \leq n$, $\gamma(s_{i-1}, o_i) = s_i$.

A planning domain is a tuple $\mathfrak{D} = (O, \gamma)$ where O is a set ground operators and γ is a state-transition function. A ground ("propositional") classical planning problem is a tuple $\mathcal{P} = \langle \mathfrak{D}, s_I, G \rangle$ where s_I is the initial state and G is a conflict-free conjunction of literals. The solution of a classical planning problem $\mathcal{P}$ is a sequence of actions in O which is executable in the initial state s_I. The decision problem for the classical planning problem is called the plan-existence problem, which is to decide whether there is a solution for the classical planning problem.[4] The complexity of the plan-existence problem for the ground case is **PSPACE**-complete [Bylander, 1994]. For the non-ground case where the planning domain consists of non-ground planning operators, the plan-existence problem is **EXPSPACE**-complete [Ghallab et al., 2004]. The reason is that when we restrict all atoms to be ground, the number of operator instances and the size of each state reduce from exponential to polynomial.

The number of actions in a sequence of action is called the length of the sequence. If the length of solutions is restricted to be less than a natural number δ we

[4]If it is clear from the context, we abbreviate "plan-existence problem for classical planning" as "classical planning problem" simply.

call such problem a plan-existence problem with bounded horizon, denoted by $\mathcal{P}_\delta = \langle \mathfrak{D}, s_I, G, \delta \rangle$. Interestingly, the restriction does not reduce the complexity and when the bound is bigger than 1, the complexity of the plan-existence problem with bounded horizon for ground classical planning is still **PSPACE**-complete [Bylander, 1994].

2.2.2 HTN Planning

Different from classical planning, hierarchical task network (HTN) planning [Erol et al., 1994a] is a planning formalism which is based on domain-specific heuristics about the hierarchical decomposition of compound tasks, until primitive tasks are obtained. [Erol et al., 1994a] proposes a model-theoretic semantics for HTN planning which is operational. In this thesis, we adapt the presentation way of operators in HTN planning to conform with that of operators in classical planning, which is used in [Ghallab et al., 2004].

HTN planning is also defined on a function-free first-order language $\mathscr{L}$ whose vocabulary includes a finite set of predicates and constants and an infinite set of variables. In addition, the vocabulary of $\mathscr{L}$ includes a finite set of primitive task symbols, a finite set of compound task symbols and an infinite set of task labeling symbols which are used to identify tasks, so that multiple instances of task symbols are allowed. All these sets are mutually disjoint.

In HTN planning, task symbols are syntactically atoms and fall into two categories: primitive task symbols that can be executed directly and compound task symbols that cannot. Every task syntactically is a pair of $(t : \alpha)$ where t is a task labeling symbol that cannot occur more than once in the planning problem and α is a task symbol.[5] A primitive task is associated with an operator. Just as classical planning, every operator has the form of $o = (\texttt{name}(o), \texttt{pre}(o), \texttt{eff}^+(o), \texttt{eff}^-(o))$ where its name $\texttt{name}(o)$ is syntactically an atom $n(\mathbf{t})$; its precondition $\texttt{pre}(o)$ is a formula in $\mathscr{L}$; its positive effect $\texttt{eff}^+(o)$ and its negative effect $\texttt{eff}^-(o)$ are two

[5] We usually use the task labeling symbol to denote the entire task.

sets of atoms, respectively. We suppose that after grounding the atoms, these two sets are disjoint. Similarly, we usually take the task symbol as $\texttt{name}(o)$ and use it to denote the entire operator o.

A task network is a couple $d = [T, \varphi]$ consisting of a set of tasks T and a boolean formula φ. The boolean formula φ is constructed[6] from variable binding constraints such as $(v = v')$ and $(v = c)$, ordering constrains such as $(t \prec t')$, and state constraints such as $(t, l), (l, t), (t, l, t')$ where v, v' are propositional variables, l is a literal, c is a constant, and n, n' are tasks in T. Intuitively, ordering constraint $(t \prec t')$ means task t has to be performed before task t'; state constraints $(t, l), (l, t)$ and (t, l, t') mean that l must be true in the state immediately after t, immediately before t, and in all states between t and t'. The task network $[T, \varphi]$ is achieved if the set of tasks T is achieved and the boolean formula φ holds.

A task network is called primitive if it only contains primitive tasks. A task network is ground if all variables in all tasks and constraint formula are ground.

A decomposition method (α, d) specifies that the compound task α can be decomposed into the task network d: α is going to be achieved once d is achieved. For example, the action of submitting a paper can be decomposed into a task network $d = [T, \varphi]$ where T consists of the two actions of writing a paper and uploading it and constraint φ expresses that the writing action has to be performed before the uploading action.

An HTN planning domain is a tuple $\mathfrak{D} = (\mathcal{O}, \mathcal{M})$ where $\mathcal{O}$ is a set of operators and $\mathcal{M}$ is a set of decomposition methods. An HTN planning problem is a tuple $\mathcal{P} = (\mathfrak{D}, s_I, d)$ where $\mathfrak{D}$ is an HTN planning domain, s_I is the initial state which is also a set of ground atoms and d is the initial task network to plan for. If all actions and atoms are ground in the HTN planning problem, we call it ground (or propositional) HTN planning problem, Let us introduce the operational semantics of HTN planning.

[6] Logical connectives $\neg, \wedge, \vee$ are allowed.

Just as in classical planning, the change of the world is defined by the state-transition function, which is determined by a set of ground primitive tasks O: $\gamma : 2^{\mathsf{Atm}(\mathscr{L}_0)} \times O \longrightarrow 2^{\mathsf{Atm}(\mathscr{L}_0)}$, where:

- $\gamma(s, o)$ is defined iff $s \models \mathtt{pre}(o)$;
- $\gamma(s, o) = (s \setminus \mathtt{eff}^{-}(o)) \cup \mathtt{eff}^{+}(o)$ if $\gamma(s, o)$ is defined.

A plan is a sequence of ground primitive tasks. As before, a plan $\langle o_1, \ldots, o_n \rangle$ is executable in a state s_0 iff there exists a sequence of states $s_1, \ldots, s_n$ such that for all $1 \leq i \leq n$, $\gamma(s_{i-1}, o_i) = s_i$.

A plan σ is a completion of a primitive task network d in state s_I, denoted by $\sigma \in \mathsf{comp}(\mathfrak{D}, s_I, d)$, if σ is a total ordering of a ground instance of d that satisfies the constraint formula and is executable in s_I.

Definition 2.1. For a ground primitive task network $d = [\{t_1, \ldots, t_n\}, \varphi]$, let $\sigma = \langle f_1, \ldots, f_n \rangle \in \mathsf{comp}(\mathfrak{D}, s_I, d)$ and π be a permutation such that whenever $\pi(i) = j$, $\alpha_i = f_j$. Then the constraint formula in task network d is evaluated as follows:

- $(c_i = c_j)$, if c_i, c_j are the same constants
- $first[t_i, t_j, \ldots]$ is $min\{\pi(i), \pi(j), \ldots\}$
- $last[t_i, t_j, \ldots]$ is $max\{\pi(i), \pi(j), \ldots\}$
- $(t_i \prec t_j)$ is true if $\pi(i) < \pi(j)$
- (l, t_i) is true if $s_{\pi(i)-1} \models l$
- (t_i, l) is true if $s_{\pi(i)} \models l$
- (t_i, l, t_j) is true if $s_k \models l$ for $\pi(i) \leq k < \pi(j)$
- logical connectives $\neg, \wedge, \vee$ are evaluated as in propositional logic

where $min\{.\}$ and $max\{.\}$ are the minimum and maximum number of the natural number set, respectively.

Tasks in a task network are called totally ordered if there is only a total ordering of tasks to satisfy the constraints in the task network.

If d is a primitive task network containing variables, then

$$\mathsf{comp}(\mathfrak{D}, s_I, d) = \{\sigma \mid \sigma \in \mathsf{comp}(\mathfrak{D}, s_I, d'), d' \text{ is a ground instance of } d\}.$$

If a task network d contains non-primitive tasks, then the set of completions for d is the empty set.

Next we introduce how to decompose tasks.

Definition 2.2. For a task network $d = [\{t, t_1, \ldots, t_n\}, \varphi]$ where $t : \alpha$ is a non-primitive task, let $m = (\alpha, [\{t'_1, \ldots, t'_k\}, \varphi_m])$ be a decomposition method.[7] Then we define $\mathsf{reduce}(d, t, m)$ as follows:

$$\mathsf{reduce}(d, t, m) = [\{t'_1, \ldots, t'_k, t_1, \ldots, t_n\}, \varphi_m \wedge \psi]$$

where ψ is obtained from φ with the following modifications:

- replace $(t \prec t_i)$ with $(last[t'_1, \ldots, t'_k] \prec t_i)$, as t_i must be after every subtask of t
- replace $(t_i \prec t)$ with $(t_i \prec first[t'_1, \ldots, t'_k])$, as t_i must be before every subtask of t
- replace (l, t) with $(l, first[t'_1, \ldots, t'_k])$, as l must be true immediately before the first subtask of t
- replace (t, l) with $(last[t'_1, \ldots, t'_k], l)$, as l must be true immediately after the last subtask of t

[7] We suppose the task symbols in the decomposition method m have been renamed with task symbols which do not occur anywhere else.

- replace (n_i, l, t) with $(n_i, l, first[t'_1, \ldots, t'_k])$, as l must be true between t_i and the first subtask of t
- replace (t, l, t_i) with $(last[t'_1, \ldots, t'_k], l, t_i)$, as l must be true between the last subtask of t and t_i
- replace every occurrence of t in the expressions $first[.]$ and $last[.]$ in φ with $t'_1, \ldots, t'_k$

Intuitively, $\mathsf{reduce}(d, t, m)$ is the task network obtained from task network d by replacing t with the subtasks in the decomposition method m and incorporating the constraint formula of the decomposition method into the constraint formula of d.

Then we define the set of reductions of $d = [T, \varphi]$ in state s_I and domain $\mathfrak{D}$, denoted by $\mathsf{red}(\mathfrak{D}, s_I, d)$, as follows:

$$\mathsf{red}(\mathfrak{D}, s_I, d) = \{\mathsf{reduce}(d, t, m) \mid t \in T, m \in \mathfrak{D}\}.$$

The solution for an HTN planning problem $\mathcal{P} = (\mathfrak{D}, s_I, d)$ is a plan such that the task network d will be achieved by decomposing compound tasks iteratively via the decomposition methods in $\mathfrak{D}$, starting from the initial state s_I.

More precisely, A plan σ solves a primitive task network d in the initial state s_I, iff $\sigma \in \mathsf{comp}(\mathfrak{D}, s_I, d)$; a plan σ solves a non-primitive task network d in the initial state s_I iff σ solves some reduction $d' \in \mathsf{red}(\mathfrak{D}, s_I, d)$ in the initial state s_I.

Definition 2.3. We define the set of plans $\mathsf{sol}(\mathfrak{D}, s_I, d)$ which solve an HTN planning problem $\mathcal{P} = (\mathfrak{D}, s_I, d)$:

$$\begin{aligned}
\mathsf{sol}^1(\mathfrak{D}, s_I, d) &= \mathsf{comp}(\mathfrak{D}, s_I, d) \\
\mathsf{sol}^{k+1}(\mathfrak{D}, s_I, d) &= \mathsf{sol}^k(\mathfrak{D}, s_I, d) \cup \bigcup_{d' \in \mathsf{red}(\mathfrak{D}, s_I, d)} \mathsf{sol}^k(\mathfrak{D}, s_I, d') \\
\mathsf{sol}(\mathfrak{D}, s_I, d) &= \bigcup_k \mathsf{sol}^k(\mathfrak{D}, s_I, d)
\end{aligned}$$

Intuitively, $\mathsf{sol}^k(\mathfrak{D}, s_I, d)$ is the set of plans that can be derived in k steps of refinements, while $\mathsf{sol}(\mathfrak{D}, s_I, d)$ is the set of plans that can be derived in any finite number of steps.

The decision problem for HTN planning is called the plan-existence problem for HTN planning which is to decide whether there is a solution of the HTN planning problem. The plan-existence problem for HTN planning, whether variables are allowed or not, is undecidable [Erol et al., 1996]. Alford *et al.* [2015a] summarize the complexity results of the plan-existence problem for HTN planning with various restrictions, which range from **PSPACE** up to undecidable. In particular, if all tasks in task networks are totally ordered by the constraints, the complexity for propositional HTN planning is **EXPTIME**-complete and the complexity for non-ground HTN planning is 2-**EXPTIME**-complete because of the exponential size expansion caused by the grounding procedure. If we additionally restrict the decomposition of tasks not to be recursive, the complexity for propositional HTN planning reduces to **PSPACE**-complete and **EXPSPACE**-complete for the non-ground case.

2.3 Uniform Terminology

In the literature on classical planning and HTN planning, one can find different terminologies on the object "action". For example, to represent an action instance, "operator" is used in classical planning and "task" is used in HTN planning. In this section, we summarize and categorize the terminologies about "action", as shown in Table 2.1. Usually, we define what type of actions can be performed or accomplished in the domain and we call them "action types". Here we use first order atoms to syntactically represent action types, we call them "ground action types" (or "propositional action types") after grounding. In order to capture multiple occurrences of the same (ground) action, we instantiate action types into "action token" which should be identified in the context. Notice that "task symbol" is used to represent "action type" in the formalism of [Erol et al., 1994a] (noted [1] in Table 2.1) while in the formalism of [Geier and Bercher, 2011] (noted [2] in Table 2.1) it is used for "action token".

Table 2.1: The Uniform Terminology of Actions

		action type	ground action type	action token
high-level action		[1] compound task symbol [2] compound task name	[1] ground compound task symbol [2] ground compound task name	[1] labeling symbol [2] task symbol
basic action	classical	operator	ground operator/actions	
	HTN	[1] primitive task symbol [2] operator	[1] ground primitive task symbol [2] ground operator	[1] labeling symbol [2] task symbol

As the concept of actions is fundamental in both automated planning and BDI theories, an uniform terminology of actions contributes to integrating the research results of these two communities. In this thesis, we stipulate that the (ground) action type for the compound task is called "(ground) high-level action" and for the primitive task is called "(ground) basic action" and that the action token in HTN planning is simply called "compound task" and "primitive task".

2.4 Summary

In this chapter, we have recalled Shoham's database framework on beliefs and intentions. Compared with Cohen & Levesque's logic, Shoham's database framework is a much simpler account that is based on a database of time-indexed basic actions and beliefs. Compared with the existing BDI agents, the belief-intention database is more logical and more suitable for revising beliefs and intentions. Although Shoham's database perspective lacks an account of intention refinement, we believe it to be a promising basis for a logic-based comprehensive analysis of beliefs and intentions. In the next chapter, we put the notion of intention refinement into Shoham's database perspective.

Now, we have a better understanding on the difference between existing theories and automated planning. Updating the figure about the comparison between BDI agents and automated planning (1.2), we obtain a new figure (Figure 2.1) about the comparison between BDI theories and automated planning.

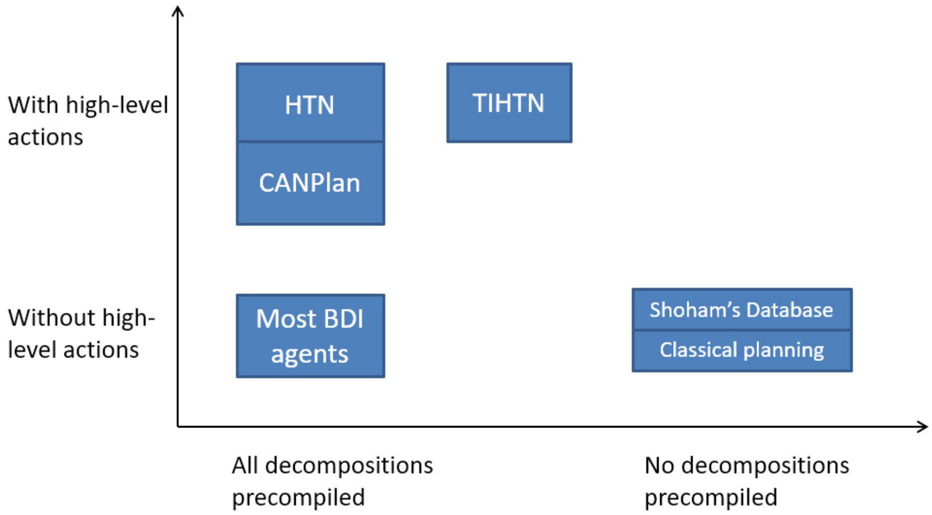

FIGURE 2.1: Comparison between BDI Theories and Automated Planning 2

In the real word, HTN planning is widely used for practical application scenarios [Biundo et al., 2011, de Silva et al., 2015, Dvorak et al., 2014, Lin et al., 2008]. This is partly because the hierarchy of actions provides a convenient way to write problem-solving "recipes" that correspond to how a human domain expert might think about solving a planning problem. However, HTN only has an operational semantics and it lacks semantics for decomposition methods[8] which can help evaluate the correctness of decomposition methods. In Chapter 5, we provide a dynamic logic account on HTN planning via propositional dynamic logic and evaluate the coherence condition of domains.

Another problem is that it is usually a challenge to provide a complete domain which includes all decomposition methods for all compound tasks. Defining only a partially hierarchical domain is not sufficient to produce all desired solutions. Several HTN researchers have investigated how partially hierarchical domain knowledge can be exploited during planning without relying on the standard HTN formalism [Biundo and Schattenberg, 2001, Geier and Bercher, 2011, Kambhampati et al., 1998, Shivashankar et al., 2013]. In the original HTN planning, high-level action has no an explicit "postcondition" as defined for basic actions, which is

[8]As refinement is considered as decomposition in HTN planning, in this thesis the term "refinement method" is also used for "decomposition method".

instead captured by a state constraint expressing that a formula holds immediately after the action. But in different decomposition methods for the high-level action, it has different implicit postconditions because there may be serval state constraints stipulating different formulas to hold immediately after performing the action. Indeed, Kambhmpati [1998] advocates that the high-level action should use a primary effect which is nothing but the postcondition, distinguished with the effects of basic actions. Based on that, we also propose a completeness postulate in Chapter 5, apart from a soundness postulate. On the other hand, hierarchical task network with task insertion (TIHTN) planning [Geier and Bercher, 2011] relaxes the restriction to allow solutions generated by both the decomposition of compound tasks and the insertion of tasks from outside the decomposition hierarchy. Indeed, [Geier and Bercher, 2011] proposes an alternative simplified formalism of HTN planning. Here we omit this formalism and show the details in Chapter 6 which extends TIHTN further by state constraints.

We are convinced that the uniform terminology of actions is supposed to be the first step to combine the HTN planning and BDI agents.

Chapter 3

Refinement of Intentions in the Databases

Inspired by Shoham's database perspective, we view basic and high-level intentions as organized in a belief-intention database that specifies the temporal intervals within which the corresponding actions have to be performed. This database also contain beliefs about the environment and its change. Actions and events are defined in terms of their pre- and postcondition. Higher intentions can be refined by choosing several possible lower-level intentions to implement them. Based on the refinement of intentions, by adding those lower-level intentions, the database may be further refined. A means-end relation on intentions follows the refinement of intentions, alias instrumentality relation, linking higher- and lower-level intentions. We investigate the decision problems in the database which are the satisfiability problem, the consequence problem, the deciding-refinement problem and the deciding-instrumentality problem. We establish the complexity results of these decision problems which are all **PSPACE**-complete.

This chapter is organized as follows. Section 3.1 introduces the notions of events and high-level actions and defines dynamic theories. Section 3.2 presents belief-intention databases and gives their semantics. Section 3.3 defines refinement relation of intentions and the refinement operation of intention database. We define instrumentality relation based on refinement of intentions in Section 3.4. Section 3.5 introduces the complexity results of the decision problems for satisfiability, consequence, refinement and instrumentality. Section 3.6 discuss the two perspectives of considering refinement and summarizes this chapter.

3.1 Dynamic Theories

As mentioned in the last chapter, none of the BDI logics that are rooted in Cohen & Levesque's and Rao & Georgeff's logics fully adapted Cohen & Levesque's definition of intention and they are still rather complicated. Shoham's database perspective on beliefs and intentions is a much simpler account than the rather complex theories of intention due to Cohen & Levesque and to Rao & Georgeff and others, while being more suitable for a logical analysis than the existing, heavily implementation-driven BDI agents. Therefore, we take it to be a starting point for designing a logical framework of beliefs and intentions.

The common point between all these theories is the assumption that intentions are organized in a flat way: there is no notion of high- and low-level intentions. This is clearly a major shortcoming with respect to [Bratman et al., 1988b]. Intentions might be defined at high level and next refined in order to obtain executable actions.

The database approach up to now did not also cater for environment actions, alias events. For that reason, the existing approaches—despite their STRIPS-like action theories—fail to solve the frame problem: contrarily to what one may expect, the agent's beliefs at time point t together with her action a at t do not determine her beliefs at $t+1$ (only belief related to the effects of a can be determined).

Remark 3.1. In Shoham's database perspective, the terms "goal" and "desire" are not used and from some perspective, they are considered as the postcondition of intentions. By considering the frame axiom, beliefs in our framework are also considered as goals or desires in a way that beliefs must be satisfied in the future.

In this chapter, we go beyond the initial Shoham's database framework by not only considering the actions of the agent under concern, but also the environment's actions. Taking the perspective of the planning agent, we call the latter 'events' and the former just 'actions'. We only consider basic events, each of which takes one time unit. We allow for several events to occur simultaneously, allowing thus for environments with multiple agents. In other words, we have a planning agent who is proactive and who has beliefs about a reactive environment: with respect to her beliefs, she believes the environment will react accordingly.

The first thing we have to do is to define *dynamic theories* which describes the behavior of actions and events.

Let $\mathsf{Evt}_0 = \{e, f, \ldots\}$ be a set of basic events and $\mathsf{Act}_0 = \{a, b, \ldots\}$ a set of basic actions. Basic events and basic actions take one time unit. Basic actions can be directly executed by the planning agent. The set Act_0 is contained in the set of all actions $\mathsf{Act} = \{\alpha, \beta, \ldots\}$ which also contains non-basic, high-level actions. The set of propositional variables is $\mathbb{P} = \{p, q, \ldots\}$. The language of propositional formulas built on $\mathbb{P}$ is denoted by $\mathscr{L}_{\mathbb{P}}$. We suppose that the sets $\mathbb{P}$, Evt_0, and Act are all finite. The behavior of actions and events is described by dynamic theories where basic actions and events have the same description.

Definition 3.1. (Dynamic theory) A dynamic theory is a tuple $\mathcal{D} = \langle \mathtt{pre}, \mathtt{post} \rangle$ with $\mathtt{pre}, \mathtt{post} : \mathsf{Act} \cup \mathsf{Evt}_0 \longrightarrow \mathscr{L}_{\mathbb{P}}$, such that the effects of basic actions and events are conjunctions of literals: there are functions $\mathtt{eff}^+, \mathtt{eff}^- : \mathsf{Act}_0 \cup \mathsf{Evt}_0 \longrightarrow 2^{\mathbb{P}}$ such that for every $x \in \mathsf{Act}_0 \cup \mathsf{Evt}_0$,

$$\models \mathtt{post}(x) \leftrightarrow \Big(\bigwedge_{p \in \mathtt{eff}^+(x)} p \Big) \wedge \Big(\bigwedge_{p \in \mathtt{eff}^-(x)} \neg p \Big) \tag{3.1}$$

where $\mathtt{eff}^+(x) \cap \mathtt{eff}^-(x) = \emptyset$.

In other words, basic actions and events are STRIPS-like. For example, for the basic empty action wait we have $\mathtt{pre}(\mathsf{wait}) = \top$ and $\mathtt{eff}^+(\mathsf{wait}) = \mathtt{eff}^-(\mathsf{wait}) = \emptyset$. The functions $\mathtt{pre}, \mathtt{post}, \mathtt{eff}^+$ and $\mathtt{eff}^-$ are naturally extended to sets, e.g. $\mathtt{pre}(X) = \bigwedge_{x \in X} \mathtt{pre}(x)$ for $X \subseteq \mathsf{Act}_0 \cup \mathsf{Evt}_0$.

We use $|S|$ to denote the cardinality of a set S. We use $\mathsf{length}(\varphi)$ to denote the length of a propositional formula φ which is the number of symbols except for parentheses in the formula. We use $\mathsf{length}(\mathcal{D})$ to denote the length of dynamic theory $\mathcal{D}$ which is the sum of the length of all pre- and postcondition formulas in $\mathcal{D}$.

Definition 3.2. (Coherence) A dynamic theory $\mathcal{D}$ is coherent if and only if for every basic action $a \in \mathsf{Act}_0$ and event set $E \subseteq \mathsf{Evt}_0$, if $\mathtt{pre}(\{a\} \cup E)$ is consistent then $\mathtt{post}(\{a\} \cup E)$ is consistent.

Notice that the exponential number of pairs of basic action and set of events entails a potential exponential number of consistency tests. To avoid this issue, we introduce a formula checking coherence with a polynomial length of dynamic theory and propositional variables.

Proposition 3.1. *A dynamic theory $\mathcal{D}$ is coherent iff the following formula, denoted by $\mathtt{Coh}(\mathcal{D})$, is valid:*

$$\bigwedge_{\substack{e \in \mathsf{Evt}_0, x \in \mathsf{Act}_0 \cup \mathsf{Evt}_0, \\ \mathtt{eff}^+(e) \cap \mathtt{eff}^-(x) \neq \emptyset \text{ or } \mathtt{eff}^-(e) \cap \mathtt{eff}^+(x) \neq \emptyset}} (\mathtt{pre}(e) \wedge \mathtt{pre}(x) \to \bot).$$

Proof. "$\Rightarrow$": Suppose dynamic theory $\mathcal{D}$ is coherent and $\mathtt{post}(\{a\} \cup E)$ is inconsistent. Because all basic actions and events have a consistent postcondition in form of a conjunction of literals, only a pair of an action or event $x \in \{a\} \cup E$ and an event $e \in E$ such that one has a positive effect on propositional variable p and the other has a negative effect on p, would make $\mathtt{post}(\{x, e\})$ inconsistent and further $\mathtt{post}(\{a\} \cup E)$ inconsistent. According to the definition of coherence their jointly precondition $\mathtt{pre}(\{x, e\})$ is inconsistent. Thus $\mathtt{Coh}(\mathcal{D})$ is valid.

"$\Leftarrow$": Suppose $\mathtt{Coh}(\mathcal{D})$ is valid and there exists some action a and event set E such that $\mathtt{pre}(\{a\} \cup E)$ is consistent while $\mathtt{post}(\{a\} \cup E)$ is inconsistent. As $\mathtt{post}(a)$ and $\mathtt{post}(E)$ can be rewritten into a conjunction of literals as formula (3.1), there must exist a pair of p and $\neg p$ occurring in $\mathtt{post}(\{a\} \cup E)$. Then there are $x, y \in \{a\} \cup E$ such that $x \neq y$ and $p \in \mathtt{eff}^+(x) \cap \mathtt{eff}^-(y)$. Due to $\mathtt{Coh}(\mathcal{D})$, we have $\mathtt{pre}(x) \wedge \mathtt{pre}(y) \to \bot$, which entails $\mathtt{pre}(a) \wedge \mathtt{pre}(E) \to \bot$, contradicting that $\mathtt{pre}(\{a\} \cup E)$ is consistent. $\square$

Let us now prove that checking coherence is co-**NP**-complete.

Theorem 3.2 (**Complexity of Coherence**). *Given any dynamic theory $\mathcal{D}$, to decide whether $\mathcal{D}$ is coherent is co-**NP**-complete.*

Proof. As the length of formula $\mathtt{Coh}(\mathcal{D})$ is bounded by $O(|\mathsf{Act}_0 \cup \mathsf{Evt}_0|^2 \times \mathsf{length}(\mathcal{D}))$, deciding whether the formula is valid is in co-**NP**, in consequence deciding coherence is in co-**NP**.

To establish hardness, consider a propositional formula φ. Let $\mathsf{Act}_0 = \{a\}$, $\mathsf{Evt}_0 = \{e\}$ and let $\mathcal{D}$ be a dynamic theory with $\mathtt{pre}(a) = \mathtt{pre}(e) = \varphi$, $\mathtt{post}(a) = p$ and $\mathtt{post}(e) = \neg p$. As $\mathtt{post}(a) \wedge \mathtt{post}(e)$ is inconsistent, φ is inconsistent iff $\mathcal{D}$ is coherent. It follows that deciding coherence is co-**NP**-hard. $\square$

Now we use a simple example to show what a coherent dynamic theory is.

Example 3.1. *Alice has a high-level action* **buy** *of buying a movie ticket and the basic actions of buying a ticket online* buyWeb, *going to the cinema* gotoC, *and buying a ticket at the cinema counter* buyC. *Moreover, there is an event of the website delivering the electronic ticket* $\mathsf{deliver}$. *Let the propositional variables* $\mathsf{PaidWeb}$, Ticket *and* InC *respectively stand for "Alice has paid online", "Alice has a ticket" and "Alice is in the cinema". The actions and events obey the following coherent dynamic theory $\mathcal{D}$:*

$\mathtt{pre}(\mathsf{wait}) = \top$	$\mathtt{post}(\mathsf{wait}) = \top$
$\mathtt{pre}(\mathsf{buyWeb}) = \top$	$\mathtt{post}(\mathsf{buyWeb}) = \mathsf{PaidWeb}$
$\mathtt{pre}(\mathsf{gotoC}) = \top$	$\mathtt{post}(\mathsf{gotoC}) = \mathsf{InC}$
$\mathtt{pre}(\mathsf{buyC}) = \mathsf{InC}$	$\mathtt{post}(\mathsf{buyC}) = \mathsf{Ticket}$
$\mathtt{pre}(\mathbf{buy}) = \top$	$\mathtt{post}(\mathbf{buy}) = \mathsf{Ticket}$
$\mathtt{pre}(\mathsf{deliver}) = \mathsf{PaidWeb} \wedge \neg\mathsf{Delivered}$	$\mathtt{post}(\mathsf{deliver}) = \mathsf{Ticket} \wedge \mathsf{Delivered}$

The following is a counterexample of coherent dynamic theory.

Example 3.2. *Alice has a basic action to close the door* $\mathsf{closeDoor}$ *and there is an event* $\mathsf{openDoorAuto}$ *that the door is open automatically if there is someone is in front of the door which is denoted by propositional variable* $\mathsf{FrontDoor}$. *We use propositional variable* $\mathsf{DoorClosed}$ *to denote that the door is closed. We have a dynamic theory* $\mathcal{D}'$ *which includes this basic action and event:*

- $\mathtt{pre}(\mathsf{closeDoor}) = \mathsf{FrontDoor}$, $\mathtt{post}(\mathsf{closeDoor}) = \mathsf{DoorClosed}$
- $\mathtt{pre}(\mathsf{openDoorAuto}) = \mathsf{FrontDoor}$, $\mathtt{post}(\mathsf{openDoorAuto}) = \neg\mathsf{DoorClosed}$

As the postcondition of basic action $\mathsf{closeDoor}$ *and event* $\mathsf{openDoorAuto}$ *are inconsistent and they have the same precondition, the above dynamic theory* $\mathcal{D}'$ *is not coherent.*

3.2 Belief-Intention Databases

We extend the Shoham's belief-intention database with events: an agent's database then contains her intentions plus her beliefs about the states and event occurrences. Her beliefs about the latter two may be incomplete. Occurrence of an event $e \in \mathsf{Evt}_0$ at time point t is noted (t, e). We also want to be able to talk about the non-occurrence of events. To that end we define the set $\overline{\mathsf{Evt}_0} = \{\bar{e} : e \in \mathsf{Evt}_0\}$ of event complements. Non-occurrence of e is noted $(t, \bar{e})$.

An *intention* is a triple $i = (t, \alpha, d) \in \mathbb{N}^0 \times \mathsf{Act} \times \mathbb{N}$ with $t < d$. It represents that the agent wants to perform α in the time interval $[t, d]$: action α should start after t and end before d. When $\alpha \in \mathsf{Act}_0$ then i is a *basic* intention. We use $i, j, \ldots$ to denote intentions and $J, J_1, \ldots$ to denote sets thereof.

Definition 3.3 (Belief-Intention Database). A belief-intention database Δ is a finite set

$$\Delta \subseteq (\mathbb{N}^0 \times \mathscr{L}_{\mathbb{P}}) \cup (\mathbb{N}^0 \times \mathsf{Evt}_0) \cup (\mathbb{N}^0 \times \overline{\mathsf{Evt}_0}) \cup (\mathbb{N}^0 \times \mathsf{Act} \times \mathbb{N}).$$

We often partition the database into belief base, event base and intention base by means of the following functions:

$$\mathcal{B}(\Delta) = \Delta \cap (\mathbb{N}^0 \times \mathscr{L}_{\mathbb{P}})$$
$$\mathcal{E}(\Delta) = \Delta \cap (\mathbb{N}^0 \times (\mathsf{Evt}_0 \cup \overline{\mathsf{Evt}_0}))$$
$$\mathcal{I}(\Delta) = \Delta \cap (\mathbb{N}^0 \times \mathsf{Act} \times \mathbb{N})$$

Given an intention $i = (t, \alpha, d)$, we define $\mathsf{end}(i) = d$. For a database Δ, we let $\mathsf{end}(\Delta)$ be the greatest time point occurring in Δ. This is well defined as database are finite. When $\mathcal{B}(\Delta) = \mathcal{E}(\Delta) = \mathcal{I}(\Delta) = \emptyset$ then we set $\mathsf{end}(\Delta) = 0$.

We reconsider the previous example to show what a database is.

Example 3.3. *[Example 3.1, continued] Alice's initial database only contains* $\Delta_c = \{(0, \mathbf{buy}, 2)\}$*, i.e., Alice intends to buy a movie ticket within the temporal interval* $[0, 2]$*. According to the dynamic theory* $\mathcal{D}$ *there are two ways to achieve this intention: either perform* buyWeb *at* 0 *and then* wait *(believing event* $\mathsf{deliver}$ *will occur at* 1 *because its precondition is* $\mathsf{PaidWeb} \wedge \neg\mathsf{Delivered}$*); or perform* gotoC *at* 0 *and* buyC *at* 1*.*

3.2.1 Semantics

The semantics of dynamic theories and databases is in terms of *paths*. A path defines for each time point which propositional variables are true, which basic actions the agent will perform, and which events will occur. Basic actions and events share the same description in the definition, but they differ in the semantics: basic actions are proactive while events are reactive.

Definition 3.4 (**$\mathcal{D}$-path**)**.** A *$\mathcal{D}$-path* is a triple $\rho = \langle V, H, D \rangle$ with $V : \mathbb{N}^0 \longrightarrow 2^{\mathbb{P}}$, $H : \mathbb{N}^0 \rightarrow 2^{\mathsf{Evt}_0}$, and $D : \mathbb{N}^0 \rightarrow \mathsf{Act}_0$.

A path ρ associates to every time point t a valuation $V(t)$ (alias a state), a set of basic events $H(t)$ happening at t, and a basic action $D(t)$ that the agent does at t. database is interpreted given a background dynamic theory.

Definition 3.5 (**$\mathcal{D}$-model**). A *model of* $\mathcal{D}$, or $\mathcal{D}$*-model*, is a path $\rho = \langle V, H, D\rangle$ such that for every time point $t \in \mathbb{N}^0$,

$$\texttt{eff}^+\big(H(t)\cup\{D(t)\}\big) \cap \texttt{eff}^-\big(H(t)\cup\{D(t)\}\big) = \emptyset \tag{3.2}$$

and

$$\begin{aligned} V(t{+}1) &= \big(V(t) \cup \texttt{eff}^+\big(H(t)\cup\{D(t)\}\big)\big) \setminus \texttt{eff}^-\big(H(t)\cup\{D(t)\}\big) \\ H(t) &= \{e \in \mathsf{Evt}_0 \mid V(t) \models \texttt{pre}(e)\} \\ D(t) &\in \{a \in \mathsf{Act}_0 \mid V(t) \models \texttt{pre}(a)\} \end{aligned}$$

So a path is a $\mathcal{D}$-model when (1) the action $D(t)$ to be performed and the events $H(t)$ to happen are consistent; (2) a minimal change condition is satisfied: the state at t and the basic action and events occurring at t determine the state at $t{+}1$; (3) the environment is reactive: event e occurs *iff* $\texttt{pre}(e)$ is true; (4) the agent is autonomous: if $\texttt{pre}(a)$ is true then the agent *can* perform a, but does not necessarily do so. Note that when $\mathcal{D}$ is coherent then the constraint (3.2) in Definition 3.5 is always satisfied.

We are now ready to define the satisfaction relation $\Vdash_{\mathcal{D}}$ between a path and an intention or a database.

Definition 3.6 (**Satisfaction of an intention**). Intention $i = (t, \alpha, d)$ is satisfied at a path $\rho = \langle V, H, D\rangle$, noted $\rho \Vdash_{\mathcal{D}} i$, if there exist t', d' such that $t{\le}t'{<}d'{\le}d$, $V(t') \models \texttt{pre}(\alpha)$, $V(d') \models \texttt{post}(\alpha)$, and $\alpha \in \mathsf{Act}_0$ implies $D(t') = \alpha$.

An intention $i = (t, \alpha, d)$ is satisfied at ρ if the intended action α can start at some $t' \geq t$ where the precondition of α holds and can end at some $d' \leq d$ where the postcondition of α holds. Moreover, when α is basic then α is indeed performed at the starting point t' according to the 'do'-function D of ρ.

Definition 3.7 (**Satisfaction of a database**). A $\mathcal{D}$-model $\rho = \langle V, H, D\rangle$ is a $\mathcal{D}$*-model* of Δ, noted $\rho \Vdash_{\mathcal{D}} \Delta$, if

- for every $(t, \varphi) \in \mathcal{B}(\Delta)$: $V(t) \models \varphi$;
- for every $(t, e) \in \mathcal{E}(\Delta)$: $e \in H(t)$;
- for every $(t, \bar{e}) \in \mathcal{E}(\Delta)$: $e \notin H(t)$;
- for every $i \in \mathcal{I}(\Delta)$: $\rho \Vdash_{\mathcal{D}} i$.

So when $\rho \Vdash_{\mathcal{D}} \Delta$ then the agent's beliefs about the state $\mathcal{B}(\Delta)$ and about the (non-)occurrence of events $\mathcal{E}(\Delta)$ are correct w.r.t. ρ, and all intentions in $\mathcal{I}(\Delta)$ are satisfied on ρ. A database Δ is $\mathcal{D}$-*satisfiable* when Δ has a $\mathcal{D}$-model.

Let us reconsider the previous example and describe the $\mathcal{D}$-model with a graph.

Example 3.4 (Example 3.3, continued). *Suppose Alice chooses to buy the ticket online. Let $\rho = \langle V, H, D\rangle$ be the $\mathcal{D}$-model of Figure 3.1, where we suppose that*

$$
\begin{aligned}
&V(0) = \emptyset \\
&V(1) = \{\mathsf{PaidWeb}\} \\
&V(2) = \{\mathsf{PaidWeb}, \mathsf{Ticket}\} \\
&H(0) = \emptyset,\ \ H(1) = \{\mathsf{deliver}\} \\
&D(0) = \mathsf{buyWeb},\ \ D(1) = \mathsf{wait}
\end{aligned}
$$

As $V(0) \models \mathtt{pre}(\mathbf{buy})$ and $V(2) \models \mathtt{post}(\mathbf{buy})$, we have $\rho \Vdash_{\mathcal{D}} (0, \mathbf{buy}, 2)$.

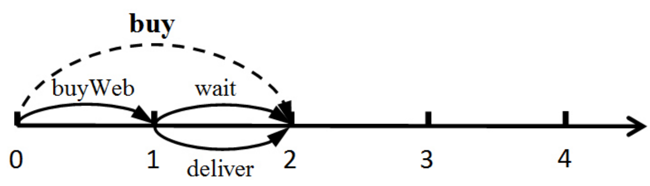

FIGURE 3.1: A $\mathcal{D}$-Model of the Database Δ of Example 3.4

Δ is a $\mathcal{D}$-*consequence of* Δ', noted $\Delta' \models_{\mathcal{D}} \Delta$, if every $\mathcal{D}$-model of Δ' is also a $\mathcal{D}$-model of Δ. When Δ is a singleton $\{i\}$ we write $\Delta' \models_{\mathcal{D}} i$ instead of $\Delta' \models_{\mathcal{D}} \{i\}$.

3.3 Refinement

3.3.1 Refining an Intention

A high-level intention cannot be executed directly by the agent: it can only be refined into lower-level intentions, until basic intentions are produced. For example, my high-level intention i to submit a paper to conference before its deadline (suppose Jun. 30) is decomposed into intention i_1 to log in the paper submission management system before Jun. 30 and then intention i_2 to upload a paper as a PDF file; etc.

Definition 3.8. We say that an intention $i \in \Delta$ is *refined in* Δ when $\Delta \setminus \{i\} \models_{\mathcal{D}} i$; otherwise we say that i has not been refined yet.

Intuitively, to refine an intention i means to add a minimal set of new intentions J to the database which, together with other intentions but i, suffice to guarantee satisfaction of i.

Definition 3.9 (Intention refinement). Let Δ be a database. Let $i \in \mathcal{I}(\Delta)$ and let J be some set of intentions. Then i is refinable to J in Δ, noted $\Delta \models_{\mathcal{D}} i \lhd J$, iff

1. there is no $j \in J$ such that $\Delta \models_{\mathcal{D}} j$;
2. $\Delta \cup J$ has a $\mathcal{D}$-model;
3. $(\Delta \cup J) \setminus \{i\} \models_{\mathcal{D}} i$;
4. $(\Delta \cup J') \setminus \{i\} \not\models_{\mathcal{D}} i$ for every $J' \subset J$;
5. $\mathsf{end}(J) \leq \mathsf{end}(i)$.

Intuitively, Condition 1 states that refinement consists of adding new intentions, Condition 2 enforces consistent refinement, Condition 3 states that new added intentions must satisfy the refined intention i, Condition 4 enforces minimality of refinement. The last condition checks that time constraints are actually satisfied.

Example 3.5 (Example 3.4, continued). *Suppose Alice's current database is* Δ_c *and she decides to buy a ticket online. Let* $i = (0, \mathbf{buy}, 2)$ *and* $j = (0, \mathsf{buyWeb}, 1)$. *We have that* $\Delta_c \not\models_{\mathcal{D}} j$ *and* $(\Delta_c \cup \{j\}) \setminus \{i\} \models_{\mathcal{D}} i$ *and* $\Delta_c \setminus \{i\} \not\models_{\mathcal{D}} i$. *Therefore satisfying* j *guarantees the satisfaction of* i*: we have* $\Delta_c \models_{\mathcal{D}} i \lhd \{j\}$.

The following proposition shows that refinement is minimal:

Proposition 3.3. *Let* $\Delta \models_{\mathcal{D}} i \lhd J$. *Then:*

1. Δ *has a* $\mathcal{D}$*-model;*
2. $J \cap \mathcal{I}(\Delta) = \emptyset$ *and* $i \notin J$*;*
3. *there is no* $J' \subset J$ *such that* $\Delta \models_{\mathcal{D}} i \lhd J'$*;*
4. $J = \emptyset$ *iff* i *is already refined in* Δ.

Proof. Let $\Delta \models_{\mathcal{D}} i \lhd J$.

1. Δ must be $\mathcal{D}$-satisfiable because of Condition 1 of Def. 3.9 (and also because of Condition 2).
2. Suppose $j \in J \cap \mathcal{I}(\Delta)$. Let $J' = J \setminus \{j\}$. Then $\Delta \cup J'$ equals $\Delta \cup J$, and as $(\Delta \cup J) \setminus \{i\} \models_{\mathcal{D}} i$ we also have $(\Delta \cup J') \setminus \{i\} \models_{\mathcal{D}} i$. The latter violates condition 4 of minimality of Def. 3.9.
3. Suppose there is a $J' \subset J$ such that $\Delta \models_{\mathcal{D}} i \lhd J'$. Then by the definition of refinement we would have $(\Delta \cup J') \setminus \{i\} \models_{\mathcal{D}} i$, contradicting $\Delta \models_{\mathcal{D}} i \lhd J$.
4. $J = \emptyset$ implies that i is already refined in Δ, i.e., $\Delta \setminus \{i\} \models_{\mathcal{D}} i$, by condition 3 of Def. 3.9. The other way round, if $\Delta \setminus \{i\} \models_{\mathcal{D}} i$ then all the conditions for $\Delta \models_{\mathcal{D}} i \lhd J$ will be satisfied.

□

It immediately follows from item 2 of Proposition 3.3 that $\Delta \models_{\mathcal{D}} i \lhd J$ implies $i \notin J$.

3.3.2 Refining the Database

Let us show how the refinement relation between intentions contributes to refining a database. Intuitively, a given database Δ can be refined by means of the following procedure:

1. select an intention i of Δ;
2. find a set of intentions J such that $\Delta \models i \lhd J$;
3. add J to Δ.

The refinement relation between intentions can be extended to database:

Definition 3.10 (**Database refinement**)**.** A database Δ is one-step refinable to Δ', noted $\Delta \lhd \Delta'$, iff there is an i in Δ and a nonempty set of intentions J such that $\Delta \models i \lhd J$ and $\Delta' = \Delta \cup J$.

For $n \geq 0$, we write $\Delta \lhd^n \Delta'$ when there exist $\Delta_1, \ldots, \Delta_n$ such that $\Delta = \Delta_1 \lhd \Delta_2, \ldots, \Delta_{n-1} \lhd \Delta_n = \Delta'$. (For $n = 0$ we suppose that $\Delta = \Delta'$.)

Example 3.6 (Example 3.5, continued)**.** *Let Δ'_c be the result of adding to Δ_c intention set $\{(0, \mathsf{buyWeb}, 1)\}$ to refine intention $i = (0, \mathbf{buy}, 2)$. We have $\Delta_c \lhd \Delta'_c$.*

The following proposition shows that a model of a refined database is also a model of the original database, but the converse does not necessarily hold. Moreover, the refinement operator preserves satisfiability.

Proposition 3.4. *Let $\Delta \lhd^n \Delta'$ and $n \geq 1$. Then:*

1. $\Delta' \models \Delta$;
2. $\Delta \not\models \Delta'$;
3. *if Δ is $\mathcal{D}$-satisfiable then Δ' is $\mathcal{D}$-satisfiable;*

Proof. The proof is by induction on n. For $n = 1$ we have $\Delta \lhd \Delta'$. So there is an $i \in \Delta$ and a set of intentions J such that $\Delta \models_{\mathcal{D}} i \lhd J$ and $\Delta' = \Delta \cup J$. The first item is clear because $\Delta \subset \Delta'$. It also follows that there is at least one such $j \in \Delta'$ such that $\Delta \not\models_{\mathcal{D}} j$ by condition 1 in Def. 3.9. So we cannot have $\Delta \models_{\mathcal{D}} \Delta'$. The third item is trivial due to Condition 2 in Def. 3.9.

Let $n = k + 1, k \geq 1$ and $\Delta \lhd^k \Delta^k$, $\Delta^k \lhd \Delta'$. Suppose the three conditions are satisfied for Δ^k. Then $\Delta' \models_{\mathcal{D}} \Delta$ and $\Delta' \models_{\mathcal{D}} \Delta^k$ because $\Delta \subset \Delta^k \subset \Delta'$. From assumption we get $\Delta \not\models_{\mathcal{D}} \Delta^k$, if $\Delta \models_{\mathcal{D}} \Delta'$ then due to $\Delta' \models_{\mathcal{D}} \Delta^k$ we get $\Delta \models_{\mathcal{D}} \Delta^k$, which contradicts the assumption. So $\Delta \not\models_{\mathcal{D}} \Delta'$. If Δ^k is $\mathcal{D}$-satisfiable then Δ' is $\mathcal{D}$-satisfiable. Item 3 is then also proved. □

Our notion of refinement is strict (or proper): the models of the refined database are a strict subset of the models of the original database. We may then *complete* the database: intentions which are entailed by a database but not belonging to it are explicitly added.

Definition 3.11 (Database completion). An intention i *completes* a database Δ if $\Delta \models i$, $i \notin \Delta$ and $\mathsf{end}(i) \leq \mathsf{end}(\Delta)$. Δ' is a completion of Δ if there exists an i completing Δ such that $\Delta' = \Delta \cup \{i\}$.

The completed database is clearly equivalent to the original one. Moreover, as database and actions are finite, only a finite number of completion steps can be made.

Let us now give a sufficient condition for the elaboration of a database.

Proposition 3.5. *If a $\mathcal{D}$-satisfiable Δ contains a non-basic intention that is not refined then Δ can be either completed or refined.*

Proof. Consider a path $\rho = \langle V, H, D \rangle$ such that $\rho \Vdash_{\mathcal{D}} \Delta$. Let $J_0 = \{(s, D(s), s{+}1) \mid s < \mathsf{end}(i)\}$. So the non-basic and non-refined intention i is not in the set of basic intentions J_0 and J_0 is finite. We have $(\Delta \setminus \{i\}) \cup J_0 \models i$. Let $J \subseteq J_0$ be an inclusion-minimal set such that $(\Delta \cup J) \setminus \{i\} \models i$. (When $J = \emptyset$ then i has already been refined by item 4 of Proposition 3.3.) If there is an intention $j \in J$ with $\Delta \models_{\mathcal{D}} j$, then Δ can be completed by j (because $j \notin \Delta$ due to minimality of J). Otherwise Δ has more than one model $\Delta \models_{\mathcal{D}} i \lhd J$ and $\Delta \lhd \Delta \cup J$. □

As we require refinements to be proper, when all $\mathcal{D}$ models of a database share the same fragment from 0 to $\mathtt{end}(\Delta)$, the database cannot be refined further. So every database can be refined in a finite number of steps.

Proposition 3.6. *For every satisfiable database Δ there is an $n \leq |\mathcal{I}(\Delta)|$ and a satisfiable database Δ' such that $\Delta \lhd^n \Delta'$ and Δ' cannot be refined.*

Proof. We follow the reasoning in the proof of Proposition 3.5, refining one by one all those intentions in Δ that are non-basic and have not been refined yet. Each of these refinement steps only adds basic intentions, therefore we terminate after at most $|\mathcal{I}(\Delta)|$ steps. □

3.4 Instrumentality from Refinement

A high-level intention cannot be executed directly by the agent: it can only be refined into lower-level actions, until a database containing one basic intention per time point is obtained. In an online planning view—that also conforms to Bratman's idea that 'the further the future is away, the more uncertain it is'—one may be less demanding and only require a basic intention to start with: the agent can then perform the associated action and later refine the rest. So I start by refining the paper writing intention (down to the intention to start by, say, typing the central proof of the paper) and postpone how I am going to deal precisely with Easychair. A high-level intention cannot be executed directly by the agent: it can only be refined into lower-level actions until an executable plan obtains. In the previous section we have described how higher-level intention could be refined by lower-level intentions. The higher- and lower-level intentions should stand in naturally a kind of means-end relation: the lower-level means contributes to the higher-level end. Indeed this relation is also called *instrumentality relation* which is a notion used frequently, such as [Audi, 1982, Bratman, 2009, Dignum and Conte, 1997, Lorini and Herzig, 2008].

Instrumentality cannot be defined from an action theory alone, for several reasons. First, the time point of action execution matters. For example, let us take up our intention of attending the conference which supposed to be held in August. Suppose I also have to go to the conference host city (suppose Melbourne) in February, for some other reason. The postcondition of that action—to be in Melbourne—entails one of the preconditions of the attend-conference action. However, as I am going to come back from Melbourne by the end of February, my February intention does not contribute to my August intention. So the former is not necessarily instrumental for the latter. It may actually happen that j is instrumental for i although the postcondition of j are inconsistent with the precondition of i. For example, suppose a robot needs to recover a ball from a locked box and unlocking the box uses up all its energy; however, it needs energy to pick up the ball: while unlocking the box is a way to pick up the ball, its postcondition is inconsistent with the precondition of the ball-taking action.

Second, the precondition of the means are typically more demanding than the preconditions of the end; similarly, the postcondition of the means are more detailed than the effects of the end. For example, buying a movie ticket requires cash in your pocket while buying a ticket online may require an account. The precondition of former action should *a priori* not involve the online account because I can choose another way to buy a ticket.

Formally, the instrumentality relation relates a refined high-level intention to a set of lower-level intentions, given a background database.

Definition 3.12 (Instrumentality). Let Δ be a $\mathcal{D}$-satisfiable database. Let $i \in \mathcal{I}(\Delta)$ and let $J \subseteq \mathcal{I}(\Delta)$. Then J is *instrumental* for i in Δ, noted $\Delta \models_{\mathcal{D}} J \gtrdot i$, iff

1. $\Delta \setminus J \not\models_{\mathcal{D}} i$;
2. $(\Delta \setminus J) \cup \{j\} \models_{\mathcal{D}} i$ for every $j \in J$;

3. $\mathsf{end}(J) \leq \mathsf{end}(i)$.

When $\Delta \models_{\mathcal{D}} J \gtrdot i$ then J is a minimal set of intentions satisfying the counterfactual "if J was not in Δ then i would no longer be guaranteed by Δ" (Condition 1). Next, the presence of all intentions of J in Δ is mandatory for satisfying i (Condition 2). Moreover, the intentions of J have to be terminated before or together with i (Condition 3).

For example, consider a database Δ where I have refined my intention of submitting a paper to a conference to the intention of writing a paper and the intention of uploading it to the paper submission management system (and where these two intentions are not refined further). Then under an appropriate action theory $\mathcal{D}$, both the writing intention and the uploading intention are instrumental for the submitting intention.

Let us use a simple example with a graph to show the instrumentality works in the hierarchy of actions.

Example 3.7. *Consider the database Δ of Figure 3.2. Let $i = (t_0, \alpha, t_5)$, $i_1 = (t_0, \alpha_1, t_2)$, etc. If α_{11}, α_{12} are not the unique way to achieve α_1, which means $\Delta \setminus \{\alpha_{11}\} \not\models_{\mathcal{D}} \alpha_{11}$ and $\Delta \setminus \{\alpha_{12}\} \not\models_{\mathcal{D}} \alpha_{12}$, and α_{21}, α_{22} together are the unique way for α_2, then the sets of intentions that are instrumental for i in Δ are $\{i, i_1, i_{11}\}$, $\{i, i_1, i_{12}\}$, $\{i, i_2, i_{21}, i_{22}\}$, $\{i, i_3\}$, while $\{i, i_1, i_2\}$ and $\{i, i_1, i_2\}$ are not.*

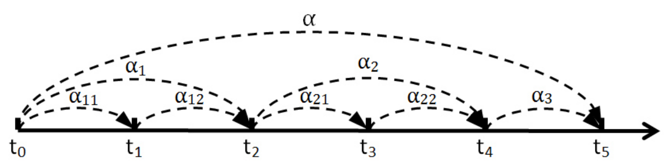

FIGURE 3.2: The database Δ of Example 3.7.

Example 3.8 (Example 3.6, continued). *The only set of intentions that is instrumental for $(0, \mathbf{buy}, 2)$ in Δ'_c is $\{(0, \mathbf{buy}, 2), (0, \mathsf{buyWeb}, 1)\}$. That is, $\Delta'_c \models_{\mathcal{D}} \{(0, \mathbf{buy}, 2), (0, \mathsf{buyWeb}, 1)\} \gtrdot (0, \mathbf{buy}, 2)$.*

Note that it is decidable whether $\Delta \models_{\mathcal{D}} J \gtrdot i$, because instrumentality results from checking satisfiability and checking consequence of databases. When $\Delta \models_{\mathcal{D}} J \gtrdot i$ then clearly $\Delta \models_{\mathcal{D}} i$ (because $i \in \Delta$). The following are some properties of instrumentality.

Proposition 3.7. *Let $\Delta \models_{\mathcal{D}} J \gtrdot i$. Then $i \in J$ and:*

- *$J = \{i\}$ iff $\Delta \setminus \{i\} \not\models_{\mathcal{D}} i$;*
- *$J = \{i, j\}$ iff $\Delta \setminus \{i\} \models_{\mathcal{D}} i$, $\Delta \setminus \{i, j\} \not\models_{\mathcal{D}} i$, and $\mathsf{end}(j) \leq \mathsf{end}(i)$.*

Proof. It is easy to check by the definition of instrumentality (Definition 3.12). □

Consequently when $\Delta \models_{\mathcal{D}} J \gtrdot i$ then J cannot be empty and $i \in J$. We now relate intention refinement to instrumentality: when $\Delta \models_{\mathcal{D}} i \lhd J$ then every element of J is instrumental for i in the new database $\Delta \cup J$.

Theorem 3.8. *If $\Delta \models_{\mathcal{D}} i \lhd J$ then $\Delta \cup J \models_{\mathcal{D}} \{i, j\} \gtrdot i$ for every $j \in J$.*

Proof. Let $\Delta \models_{\mathcal{D}} i \lhd J$ and $j \in J$. We show that $\Delta \cup J \models_{\mathcal{D}} \{i, j\} \gtrdot i$:

1. $(\Delta \cup J) \setminus \{i, j\} \not\models_{\mathcal{D}} i$ holds because $\Delta \models_{\mathcal{D}} i \lhd J$ implies $(\Delta \cup (J \setminus \{j\})) \setminus \{i\} \not\models_{\mathcal{D}} i$.
2. $(\Delta \cup J) \setminus J' \models_{\mathcal{D}} i$ holds for every $J' \subset \{i, j\}$:
   - $(\Delta \cup J) \setminus \{i\} \models_{\mathcal{D}} i$ follows from $\Delta \models_{\mathcal{D}} i \lhd J$;
   - $(\Delta \cup J) \setminus \{j\} \models_{\mathcal{D}} i$ holds because Δ contains i and $i \neq j$ by Proposition 3.3.
3. $\mathsf{end}(\{i, j\}) \leq \mathsf{end}(i)$ holds as $\Delta \models_{\mathcal{D}} i \lhd J$ implies $\mathsf{end}(J) \leq \mathsf{end}(i)$.

□

The converse does not hold: instrumentality can not guarantee that the added intentions are new, contradicting item 1 of in the definition of intention refinement (Definition 3.9).

3.5 Complexity

In this section, we will show the complexity results of the decision problems of satisfiability, consequence, refinement and instrumentality in the belief-intention database.

3.5.1 Complexity of Satisfiability

The coherence condition guarantees that there is no conflict on the effect of the action and the events occurring simultaneously at every time points, entailing the constraint formula (3.2) in the definition of $\mathcal{D}$-models (Definition 3.5):

$$\texttt{eff}^{+}\big(H(t)\cup\{D(t)\}\big)\cap\texttt{eff}^{-}\big(H(t)\cup\{D(t)\}\big)=\emptyset$$

In other words, if dynamic theory $\mathcal{D}$ is coherent, then the empty database is $\mathcal{D}$-satisfiable. Therefore, given a coherent dynamic theory, it is not necessary to check whether the infinite path is a $\mathcal{D}$-model of a database and we only need to check the former part bounded by the greatest time point occurring in the database.

We define the restriction of natural number set $\mathbb{N}$ by a natural number δ as a set of sequential natural numbers $[0,\ldots,\delta]$, denoted by $\mathbb{N}_\delta$. We define the restriction of a function $f:\mathbb{N}\longrightarrow S$ such that the domain is natural number set to a natural number δ as $f|_\delta=\{(n,s)\mid n\in\mathbb{N}_\delta\}$.

Next we introduce the notion of bounded paths.

Definition 3.13. For a path $\rho=\langle V,H,D\rangle$ and a natural number δ we call the tuple $\overline{\rho}=\langle V|_{\delta+1},H|_\delta,D|_\delta,\delta\rangle$ a bounded path of ρ and call δ the bound of $\overline{\rho}$.

Then we define bounded models by bounded paths.

Definition 3.14 (Bounded $\mathcal{D}$-model). Given a coherent dynamic theory $\mathcal{D}$, a $\mathcal{D}$-model ρ and a database Δ such that $\rho\Vdash_{\mathcal{D}}\Delta$, we call the bounded path $\overline{\rho}$ of ρ a bounded model of Δ, denoted by $\overline{\rho}\Vdash_{\mathcal{D}}\Delta$, if the bound of $\overline{\rho}$ is greater than $\mathsf{end}(\Delta)$.

The coherence condition of the dynamic theory allows us to decide if a database has a model by checking a finite path, stated as follows.

Proposition 3.9. *Given a coherent dynamic theory $\mathcal{D}$, a database Δ is $\mathcal{D}$-satisfiable iff Δ has a bounded model.*

Proof. "$\Rightarrow$": Straightforward.

"$\Leftarrow$": When the dynamic theory $\mathcal{D}$ is coherent, for every time point t, the sets $\texttt{eff}^+\big(H(t)\cup\{D(t)\}\big)$ and $\texttt{eff}^-\big(H(t)\cup\{D(t)\}\big)$ are totally disjoint. So, there is no state in which the performed action has an effect conflicting with the effect of the events which are happening. Thus, we can construct an infinite $\mathcal{D}$-model starting from the bounded model according to the definition of $\mathcal{D}$-models. $\square$

The lower bound of the complexity comes from the reduction to a plan-existence problem. Different from that shown in Section 2.2, here we rewrite the plan-existence problem with bounded horizon as a tuple $\mathcal{P} = \langle \mathcal{D}_{\mathsf{Act}_0}, \mathcal{I}, \mathcal{G}, \delta\rangle$ where δ is a natural number, $\mathcal{I}$ is a subset of propositional variable and $\mathcal{G}$ is a conflict-free conjunction of literals and $\mathcal{D}_{\mathsf{Act}_0}$ is a dynamic theory only for basic actions. The plan-existence problem with bounded horizon is to decide whether there exists a sequence of basic actions, with a length less than δ from a initial state $\mathcal{I}$ to a goal state satisfying $\mathcal{G}$. Recall that the plan-existence problem with bounded horizon is **PSPACE**-complete [Bylander, 1994].

Theorem 3.10. *Given a coherent dynamic theory $\mathcal{D}$, the $\mathcal{D}$-satisfiability problem of a belief-intention database is* ***PSPACE****-complete.*

Proof. First, we prove the problem is in **PSPACE**. Suppose the number of intentions in Δ is m. Consider a memory space with the size of $|\mathbb{P}| + 2m$. When the $|\mathbb{P}|$ cells can denote a state, the $2m$ cells can indicate the satisfaction of pre- and postcondition of the action in the corresponding intentions.

Guess a path $\overline{\rho} = \langle V, H, D, \mathsf{end}(\Delta)\rangle$, we can change the $|\mathbb{P}|$ cells according to the valuation of each time point defined by $\overline{\rho}$. Because the basic actions and events are finite, it can be checked whether $\overline{\rho}$ is a $\mathcal{D}$-model in polynomial time. The $2m$ cells are initially set to 0 and with time point changing, we change the $2m$ cells according to the satisfaction of intentions. To be specific, consider an intention $i = (t, \alpha, d)$, from time point t if $\texttt{pre}(\alpha)$ is satisfied then the cell corresponding to the precondition of i is set to 1. Then in the following time points once $\texttt{post}(\alpha)$ is satisfied the cell corresponding to postcondition of i is set to 1. Unless this cell is 1 at time point d, we stop and conclude that the path guessed does not satisfy i and further does not satisfy Δ. Therefore, we can check whether the path is a $\mathcal{D}$-model of Δ in polynomial time, because every effect, pre- and postcondition is defined as a propositional formula which can be decided to be satisfied by the state in polynomial time. As only finite sets of basic actions, events, and propositional variables can be nondeterministically chosen, deciding whether the path guessed is a $\mathcal{D}$-model of Δ is in **NPSPACE**. Because **NPSPACE** = **PSPACE**, the $\mathcal{D}$-satisfiability problem is in **PSPACE**.

Next we prove the problem is **PSPACE**-hard by reducing the plan-existence problem with bounded horizon. For a plan-existence problem $\mathcal{P}_\delta = \langle \mathcal{D}_{\mathsf{Act}_0}, \mathcal{I}, \mathcal{G}, \delta \rangle$, we construct a dynamic theory $\mathcal{D}$ by extending $\mathcal{D}_{\mathsf{Act}_0}$ with a high-level action $\mathsf{Goal} \in \mathsf{Act} \setminus \mathsf{Act}_0$ such that $\texttt{pre}(\mathsf{Goal}) = \top$ and $\texttt{post}(\mathsf{Goal}) = \mathcal{G}$. We also suppose the event set Evt_0 is empty, entailing that $\mathcal{D}$ is coherent. Suppose $\varphi_{\mathcal{I}} = \bigwedge_{p \in \mathcal{I}} p \wedge \bigwedge_{q \in \mathbb{P} \setminus \mathcal{I}} \neg q$. Then the database $\Delta = \{(0, \varphi_{\mathcal{I}}), (0, \mathsf{Goal}, \delta)\}$ has a $\mathcal{D}$-model iff there exists a plan in $\mathcal{P}$. So the $\mathcal{D}$-satisfiability problem is **PSPACE**-hard.

Hence the $\mathcal{D}$-satisfiability problem is **PSPACE**-complete. $\square$

So, we have the decidability of the satisfiability problem as the following corollary states.

Corollary 3.11. *Given a coherent dynamic theory $\mathcal{D}$, the $\mathcal{D}$-satisfiability problem of a belief-intention database is decidable.*

3.5.2 Complexity of Consequence

Next we will show the complexity of the consequence problem in belief-intention databases which is also **PSPACE**-complete.

Theorem 3.12. *Given a coherent dynamic theory $\mathcal{D}$, the $\mathcal{D}$-consequence problem deciding whether $\Delta' \models_{\mathcal{D}} \Delta$ is **PSPACE**-complete.*

Proof. Consider the special case that $\Delta' \models_{\mathcal{D}} \perp$ which means Δ' is $\mathcal{D}$-unsatisfiable. Because the $\mathcal{D}$-satisfiable problem is **PSPACE**-complete by Theorem 3.10, the problem deciding whether Δ' is $\mathcal{D}$-unsatisfiable is co-**PSPACE**-complete. As co-**PSPACE** = **PSPACE**, the $\mathcal{D}$-consequence problem **PSPACE**-hard.

Then we prove the $\mathcal{D}$-consequence problem is in **PSPACE**. Suppose the number of intentions in Δ and Δ' is m and m' respectively. As $\mathcal{D}$ is coherent, we can consider the complementary problem deciding whether there exists a path which $\mathcal{D}$-satisfies Δ' but not Δ. Consider a memory space with a size of $|\mathbb{P}| + 2(m + m')$ where the $|\mathbb{P}|$ cells can denote a state and the $2(m + m')$ cells can indicate the satisfaction of pre- and postcondition of the action in the corresponding intentions in Δ and Δ'.

Suppose $k = \mathbf{max}(\mathsf{end}(\Delta), \mathsf{end}(\Delta')$. Guess a path $\overline{\rho} = \langle V, H, D, k\rangle$, we can change the $|\mathbb{P}|$ cells according to the valuation of each time point defined by $\overline{\rho}$. Because the basic actions and events are finite, it can be check if $\overline{\rho}$ is a $\mathcal{D}$-model in polynomial time. The $2(m + m')$ cells are initially set to 0 and with time point changing, we change the $2(m + m')$ cells according to the satisfaction of intentions. Consider an intention $i = (t, \alpha, d)$, in the time points after t if $\mathtt{pre}(\alpha)$ is satisfied then the cell corresponding to the precondition of i is set to 1. Then in the following time points once $\mathtt{post}(\alpha)$ is satisfied the cell corresponding to the postcondition of i is set to 1. Unless this cell is 1 at time point d, we stop and conclude that the path guessed does not satisfy i and further does not satisfy the database including i. Therefore, we can check whether the path is a $\mathcal{D}$-model of Δ or Δ' in polynomial time, because every effect, pre- and postcondition is defined as a propositional formula which can be checked to be satisfied by the state in polynomial time. As only finite sets of basic actions, events, and propositional variables can be nondeterministically chosen, deciding whether the path guessed is a $\mathcal{D}$-model of Δ' but not Δ is in **NPSPACE**. Because **NPSPACE** = **PSPACE**, the $\mathcal{D}$-consequence problem is in **PSPACE**.

Hence, the $\mathcal{D}$-consequence problem is **PSPACE**-complete. □

So, we have a corollary for the consequence problem as follows.

Corollary 3.13. *Given a coherent dynamic theory $\mathcal{D}$, the $\mathcal{D}$-consequence problem deciding whether $\Delta' \models_{\mathcal{D}} \Delta$ is decidable.*

3.5.3 Complexity of Refinement and Instrumentality

From the definition of refinement and instrumentality, we know that satisfiability and consequence are subproblems of deciding refinement and instrumentality. So, deciding refinement and instrumentality are both **PSPACE**-hard. Next we show that these two problems are also **PSPACE**-complete by translating them into several satisfiability and consequence problems.

Theorem 3.14. *Given a coherent dynamic theory $\mathcal{D}$ and a database Δ, to decide whether an intention $i \in \Delta$ is refinable to an intention set J in Δ is **PSPACE**-complete.*

Proof. Condition 1, 2 and 4 are $\mathcal{D}$-satisfiability problems and condition 3 is a $\mathcal{D}$-consequence problem and it is easy to check condition 5 in polynomial time. As the refinement checking problem can be reduced to several $\mathcal{D}$-satisfiability problems and a $\mathcal{D}$-consequence problem which are all in **PSPACE**, the refinement checking problem is also in **PSPACE**. As the $\mathcal{D}$-satisfiability problem is its subproblem, deciding refinement is **PSPACE**-complete. □

Theorem 3.15. *Given a coherent dynamic theory $\mathcal{D}$ and a database Δ, to decide whether an intention set is instrumental for an intention in Δ is **PSPACE**-complete.*

Proof. (1) condition 1 is a $\mathcal{D}$-satisfiability problem; (2) condition 2 is a set of $\mathcal{D}$-consequence problems with a number of $|J|$; (3) it is easy to check condition 3 in polynomial time. As the instrumentality checking problem can be reduced polynomially to a $\mathcal{D}$-satisfiability problem and a $\mathcal{D}$-consequence problem which are both in **PSPACE**, the instrumentality checking problem is also in **PSPACE**. As the $\mathcal{D}$-satisfiability problem is its subproblem, deciding instrumentality is **PSPACE**-complete. □

So, we have the decidability result for the two decision problems, as the following corollary shows.

Corollary 3.16. *Given a coherent dynamic theory $\mathcal{D}$ and a database Δ, the checking refinement problem and the instrumentality problem are decidable.*

3.6 Discussion and Summary

Compared with Shoham's database framework, our contribution differs in two key points. First, we introduce high-level actions and consider high-level intentions in a flexible way: in our running example, Alice intends to buy a ticket (**buy**) during some time interval. While in Shoham's database framework, Alice can only intend at a specific time point to buy a ticket online (buyWeb) or in the cinema (buyC). Second, by considering in an explicit way the environment we solve the frame problem: our semantics enables Alice to predict that the online system will give a ticket while she waits. We stress that this last issue is also unsolved in other existing frameworks related to Shoham's database perspective. For instance, [van Zee and Doder, 2016] which proposes an AGM-like revision of beliefs and intentions in a temporal logic does not bring any solution for the frame problem.

Let us come back to the two perspectives of considering refinement: the entailment and the primitive. When refinement is primitive, just as HTN planning does, all refinement methods have to be defined by hand by the designer. In our running example, it means that Alice should already set before buying a ticket the refinement methods: either buying a ticket online (and wait) or in the cinema. This contrasts with our belief-intention database, which does not require such work: given the action theory and the current database, the refinement relation is defined based on logically entailment.

In this chapter, we extend Shoham's database framework by the fundamental concept of high-level intentions with a flexible duration. The integration of STRIPS-like environment events has allowed us to solve the frame problem. Also, the additional introduction of high-level actions does not cause undecidability for checking satisfiability and consequence relation. Our database framework is about high- and low-level intentions that are related by the refinement operator, which is, in some way, a well-founded database expansion. More general expansion may lead to unsatisfiable database and raises the issue about withdrawal or revision of intentions.

In the next chapter, we will solve the problems of deciding refinement and instrumentality by translating into the satisfiability and validity problems in propositional linear temporal logic and dynamic logic with propositional assignment.

Chapter 4

Deciding Refinement via Translating to PLTL and DL-PA

In the previous chapter, we have introduced the refinement relation among intentions in the database perspective which plays a fundamental role in BDI theories. The decision problem to check whether an intention can be refined into some intentions would be one of the most important problems to solve. In this chapter, we will introduce reductions from the decision problems in our framework to the satisfiability and validity problems in Propositional Linear Temporal Logic (PLTL) and Dynamic Logic of Propositional Assignment (DL-PA).

Linear temporal logics are widely used to describe infinite behaviors of discrete systems. Taking advantage of the efficient theorem provers of PLTL, we can solve the decision problems of satisfiability, consequence, refinement and instrumentality of belief-intention databases by translating them into the satisfiability and validity problems in PLTL.

On the other hand, Dynamic Logic of Propositional Assignment (DL-PA) was proposed in [Herzig et al., 2011] which is an instantiation of Propositional Dynamic Logic (PDL) [Harel, 1984, Harel et al., 2000]. As DL-PA only considers two assignment programs which change the truth value of a propositional variable every time, it allows us to represent the frame axiom of the dynamic theory models naturally and easily. It is shown that [Balbiani et al., 2013, Herzig et al., 2011] that every DL-PA formula can be reduced to an equivalent propositional formula. We thereby translate the decision problems in the belief-intention databases into the satisfiability problem and the validity problem in classical propositional logic and then can utilize the state of the art in efficient SAT solvers.

We will organize this chapter as follows: Section 4.1 first introduces the syntax and semantics of PLTL and then translates the satisfiability and consequence problems, under a coherent dynamic theory and a general dynamic theory, into the satisfiability and validity problems in PLTL, and proves the translations. Section 4.2 recalls briefly the syntax and semantics of DL-PA and translates the satisfiability and consequence problems under a coherent dynamic theory into the satisfiability and validity problems in DL-PA, with proving the translations. Section 4.3 summarizes this chapter.

4.1 Translating to PLTL

Over the last few decades, propositional linear temporal logic (PLTL) has obtained a lot of attentions from researchers, both theocratically and practically. The decision procedure for the satisfiability problem and the validity problem in (PLTL) has been studied [Shilov, 2005, Wolper, 1985] and the automated tools are quite mature. Various of model checking techniques for PLTL have been developed, such as approaches based on binary decision diagrams (BDD-based) [Burch et al., 1992] and based on propositional satisfiability problems (SAT-based) [Biere et al., 1999]. Besides, by translating to Büchi automata, several model checkers for PLTL have been developed, such as LTL3BA [Babiak et al., 2012] and SPOT [Duret-Lutz and Poitrenaud, 2004]. Taking advantage of the automated tools of PLTL[1], we can solve the decision problems of satisfiability, consequence, refinement and instrumentality in belief-intention databases by translating them into the satisfiability and validity problems in PLTL.

[1]Several theorem provers of PLTL can be found on http://users.cecs.anu.edu.au/~rpg/PLTLProvers/ (accessed on 27 Sep. 2017).

In this section we translate the satisfiability and consequence problems, either under a coherent dynamic theory or not, in the database into those problems of PLTL. The translations are not in polynomial time, but nevertheless we believe that the translations can help us to solve the decision problems of databases due to the existence of the existing theorem provers of PLTL.

4.1.1 Syntax and Semantics of PLTL

Following the notations in [Emerson, 1990], PLTL is defined on a countably infinite set $\mathbb{P}_L$ of propositional variables, classical propositional connectives and temporal operators $\mathbf{X}$ (next) and $\mathbf{F}$ (sometimes). PLTL formulas are defined in the standard way with abbreviating $\mathbf{G}$ as $\neg\mathbf{F}\neg$ and $\mathbf{X}^n$ as n contiguous operator $\mathbf{X}$. In particular, $\mathbf{X}^0$ is empty.

Next, we introduce the semantics of PLTL which is based on a linear-time structures. A linear-time structure is a pair of $M = (S, \varepsilon)$ where S is a set of states and $\varepsilon : S \to 2^{\mathbb{P}}_L$ is a function mapping each state s_i to a set of propositional variables which hold in s_i. Let M be a linear-time structure, $i \in \mathbb{N}^0$ a position, and φ, ψ are PLTL formulas. We define the satisfiable relation $\models$ as follows:

$$M, i \models p \text{ iff } p \in \varepsilon(s_i), \text{ where } p \in \mathbb{P}_L$$

$$M, i \models \neg\varphi \text{ iff } M, i \not\models \varphi$$

$$M, i \models \varphi \land \psi \text{ iff } M, i \models \varphi \text{ and } M, i \models \psi$$

$$M, i \models \mathbf{F}\varphi \text{ iff for some } j \geq i, M, j \models \varphi$$

$$M, i \models \mathbf{X}\varphi \text{ iff } M, i + 1 \models \varphi$$

If there exists a linear-time structure M such that $M, 0 \models \varphi$, we say φ is satisfiable. If for all linear-time structure M we have $M, 0 \models \varphi$, we say φ is valid.

4.1.2 Translation to PLTL

We first start by defining some auxiliary propositional variables. To capture the occurrence of events and the execution of basic actions, for every event e we introduce an auxiliary propositional variable h_e, defining the set $\mathbb{P}_h = \{h_e \mid e \in \mathsf{Evt}_0\}$ and for every basic action a we introduce an auxiliary propositional variable do_a, defining the set $\mathbb{P}_d = \{do_a \mid a \in \mathsf{Act}_0\}$. Moreover, we introduce a set $\mathbb{P}_c$ auxiliary propositional variables $pre_\alpha, post_\alpha, pre_e$ and $post_e$ for every action and event to denote their pre- and postcondition. Formally, $\mathbb{P}_c = \{pre_\alpha, post_\alpha, pre_e, post_e \mid \alpha \in \mathsf{Act}, e \in \mathsf{Evt}_0\}$. With the variables in $\mathbb{P}_c$, we can reduce the size of the resulting PLTL formula by avoiding multiply quoting the pre- and postcondition of actions.

Definition 4.1. Given a coherent dynamic theory $\mathcal{D}$, we define a conjunction of formulas $\Gamma^{\mathsf{L}}(\mathcal{D})$ as:

$$\bigwedge_{a \in \mathsf{Act}_0} \big(do_a \to \mathtt{pre}_a\big) \land \bigwedge_{e \in \mathsf{Evt}_0} \big(h_e \leftrightarrow \mathtt{pre}_e\big) \tag{4.1}$$

$$\land \bigvee_{a \in \mathsf{Act}_0} do_a \land \bigwedge_{a,b \in \mathsf{Act}_0, a \neq b} \neg\big(do_a \land do_b\big) \tag{4.2}$$

$$\land \bigwedge_{p \in \mathbb{P}} \Big(\mathbf{X}p \leftrightarrow \bigvee_{\substack{e \in \mathsf{Evt}_0 \\ p \in \mathtt{eff}^+(e)}} h_e \lor \bigvee_{\substack{a \in \mathsf{Act}_0 \\ p \in \mathtt{eff}^+(a)}} do_a \lor \big(p \land \bigwedge_{\substack{e \in \mathsf{Evt}_0 \\ p \in \mathtt{eff}^-(e)}} \neg h_e \land \bigwedge_{\substack{a \in \mathsf{Act}_0 \\ p \in \mathtt{eff}^-(a)}} \neg do_a\big)\Big) \tag{4.3}$$

$$\land \bigwedge_{\alpha \in \mathsf{Act}} \big((pre_\alpha \leftrightarrow \mathtt{pre}(\alpha)) \land (post_\alpha \leftrightarrow \mathtt{post}(\alpha))\big) \tag{4.4}$$

$$\land \bigwedge_{e \in \mathsf{Evt}_0} \big((pre_e \leftrightarrow \mathtt{pre}(e)) \land (post_e \leftrightarrow \mathtt{post}(e))\big) \tag{4.5}$$

Intuitively, formula (4.1) means that basic action a is executable if its precondition is satisfied and that events are reactive: when their precondition is satisfied they will happen. Formula (4.2) says that exactly one basic action is allowed at one time point. Formula (4.3) means propositional variable p is true in the next state if and only if either there is a basic action or event making it true or it is true currently and there is no basic action or event making it false. Formula (4.4) and (4.5) link the formulas of pre- and postcondition of actions and events with propositional variables. Finally, $\Gamma^{\mathsf{L}}(\mathcal{D})$ captures the definition and progression of valuations, which must be satisfied at every time point. In other words, the linear-time structure should satisfy $\mathbf{G}\Gamma^{\mathsf{L}}(\mathcal{D})$. But when it comes to deciding the satisfiability of the database with respect to a coherent dynamic theory, we only need to consider the initial fragment of the path instead of the whole infinite path, because the path is able to progress infinitely under the coherent condition. So, to denote $\mathcal{D}$ is satisfied in the beginning n valuations from $V(0)$, Then we define $\Gamma^{\mathsf{L}}(n, \mathcal{D})$ as $\Gamma^{\mathsf{L}}(\mathcal{D}) \wedge \mathbf{X}\Gamma^{\mathsf{L}}(\mathcal{D}) \wedge ... \wedge \mathbf{X}^n\Gamma^{\mathsf{L}}(\mathcal{D})$ to capture the first n time points of a $\mathcal{D}$-model.

Definition 4.2. We translate a database Δ into a conjunction of formulas $\Gamma^{\mathsf{L}}(\Delta)$ as follows:

$$\bigwedge_{(t,\varphi)\in\Delta} \mathbf{X}^t\varphi \wedge \bigwedge_{(t,e)\in\Delta} \mathbf{X}^t h_e \wedge \bigwedge_{(t,\bar{e})\in\Delta} \mathbf{X}^t \neg h_e \tag{4.6}$$

$$\wedge \bigwedge_{\substack{\alpha\notin\mathsf{Act}_0 \\ (t,\alpha,d)\in\Delta}} \bigvee_{t\leq t'<d'\leq d} (\mathbf{X}^{t'} pre_\alpha \wedge \mathbf{X}^{d'} post_\alpha) \tag{4.7}$$

$$\wedge \bigwedge_{\substack{a\in\mathsf{Act}_0 \\ (t,a,d)\in\Delta}} \bigvee_{t\leq t'<d} \mathbf{X}^{t'} do_a \tag{4.8}$$

The above definition actually formalizes the satisfaction of database in a path. For a coherent dynamic theory, we only consider the fragment of $\mathcal{D}$-model from time point 0 to $\mathsf{end}(\Delta)$. The following theorem shows the satisfiability problem in the database is connected to the satisfiability problem of PLTL.

Theorem 4.1. *Given a coherent dynamic theory $\mathcal{D}$, a database Δ is $\mathcal{D}$-satisfiable iff $\Gamma^{\mathsf{L}}(\mathsf{end}(\Delta), \mathcal{D}) \wedge \Gamma^{\mathsf{L}}(\Delta)$ is satisfiable.*

Proof. Let $M = (S, \varepsilon)$ be a linear-time structure where $S = \{s_0, s_1, ...\}$ and $\varepsilon : S \to 2^{\mathbb{P}_L}$ such that $\mathbb{P}_L = \mathbb{P} \cup \mathbb{P}_d \cup \mathbb{P}_h \cup \mathbb{P}_c$.

" $\Rightarrow$ " : Suppose there exists a $\mathcal{D}$-model $\rho = \langle V, H, D\rangle$ of Δ. Let us build a linear-time structure M as follows: for every time point ω,

- $\varepsilon(s_\omega) \cap \mathbb{P} = V(\omega)$;
- $\varepsilon(s_\omega) \cap \mathbb{P}_h = \{h_e \mid e \in H(\omega)\}$;

- if $D(\omega) = a$ then $\varepsilon(s_\omega) \cap \mathbb{P}_d = do_a$;
- $\varepsilon(s_\omega) \cap \mathbb{P}_c = \{pre_x, post_y \mid V(\omega) \models \mathtt{pre}(x), V(\omega) \models \mathtt{post}(y), x, y \in \mathsf{Act} \cup \mathsf{Evt}_0\}$.

Now we first prove $M, 0 \models \Gamma^{\mathsf{L}}(\mathsf{end}(\Delta), \mathcal{D})$. According to the definition of $\mathcal{D}$-models (Definition 3.5), if $D(\omega) = a$ and $e \in H(\omega)$ then $V(\omega) \models \mathtt{pre}(a) \wedge \mathtt{pre}(e)$. If $V(\omega) \not\models \mathtt{pre}(e)$ then e must not be in $H(\omega)$. Also, because ρ is a $\mathcal{D}$-model of Δ, we have $\mathtt{eff}^{+}(H(\omega) \cup \{D(\omega)\}) \subseteq V(\omega{+}1)$ and $\mathtt{eff}^{-}(H(\omega) \cup \{D(\omega)\}) \cap V(\omega{+}1) = \emptyset$, then $M, \omega{+}1 \models \mathtt{post}(D(\omega)) \wedge \mathtt{post}(H(\omega))$ follows. Immediately we have $M, \omega \models$ (4.1) for each time point ω. As only one action will be done at every time point, $M, \omega \models$ (4.2). For every propositional variable p, $M, \omega{+}1 \models p$ iff either there is a basic action or event to make it true or $M, \omega \models p$ and there is no action or event to make it false. Thus, $\forall \omega, M, \omega \models$ (4.3). According to the assignment of $\mathbb{P}_c$, we know that $M, \omega \models$ (4.4) $\wedge$ (4.5). Thus, for every time point ω, we obtain $M, \omega \models \Gamma^{\mathsf{L}}(\mathcal{D})$ and then $M, 0 \models \Gamma^{\mathsf{L}}(\mathsf{end}(\Delta), \mathcal{D})$.

Next we need to prove $M, 0 \models \Gamma^{\mathsf{L}}(\Delta)$. For every $(t, \varphi) \in \Delta$, we obtain $M, t \models \varphi$. For every $(t, e) \in \mathcal{E}(\Delta)$, $M, 0 \models \mathbf{X}^t h_e$ because $e \in H(t)$ and $h_e \in \varepsilon(s_t)$. So, we have $M, 0 \models$ 4.6. Similarly, for every $(t, \bar{e}) \in \mathcal{E}(\Delta)$, we have $M, 0 \models \mathbf{X}^t \neg h_e$. By the definition of satisfaction of an intention (Definition 3.6), if $\rho \Vdash_{\mathcal{D}} i$ we have $M, 0 \models$ (4.7) $\wedge$ (4.8). So we have $M, 0 \models \Gamma^{\mathsf{L}}(\Delta)$.

Thus, we can conclude that $M, 0 \models \Gamma^{\mathsf{L}}(\mathsf{end}(\Delta), \mathcal{D}) \wedge \Gamma^{\mathsf{L}}(\Delta)$.

" $\Leftarrow$ " : Suppose there exists a linear-time structure M which is a model of the resulting formula. In other words, $M, 0 \models \Gamma^{\mathsf{L}}(\mathsf{end}(\Delta), \mathcal{D}) \wedge \Gamma^{\mathsf{L}}(\Delta)$. Now we build a path $\rho = \langle V, H, D \rangle$ as follows: for every time point $\omega \leq \mathsf{end}(\Delta)$,

- $V(\omega) = \varepsilon(s_\omega) \cap \mathbb{P}$;
- $H(\omega) = \{e \mid h_e \in \varepsilon(s_\omega) \cap \mathbb{P}_h\}$;
- $D(\omega) = a$ if $do_a \in \varepsilon(s_\omega)$.

For those time points $\omega > \mathtt{end}(\Delta)$, we must be able to construct an infinite ρ according to the definition of $\mathcal{D}$-models (Definition 3.5), because $\mathcal{D}$ is coherent.

Next we show ρ is a $\mathcal{D}$-model. For every time point $\omega \leq \mathtt{end}(\Delta)$, due to (4.2), there must be an action a such that $M, t \models do_a$. So $M, \omega \models \mathtt{pre}(a)$ then we have $V(\omega) \models \mathtt{pre}(a)$ and $D(\omega) = a$. Because $M, \omega \models h_e$ iff $M, \omega \models \mathtt{pre}(e)$, we have $H(\omega) = \{e \mid V(\omega) \models \mathtt{pre}(e)\}$.

For propositional variable p and time point ω, let us consider the case that there exists an event e or action a which make p be true such that $M, \omega \models h_e \vee do_a$ (Case 1). It entails that $p \in \mathtt{eff}^+(H(\omega) \cup \{D(\omega)\})$ and $M, \omega \models \mathbf{X}p$ then $p \in V(\omega+1)$. As $\mathcal{D}$ is coherent, there is no event e' or action a' which make p false happening at ω. So, we have $p \notin \mathtt{eff}^-(H(\omega) \cup \{D(\omega)\})$ and then $p \in V(\omega+1)$. Thus, Case 1 satisfies the progression criterion of Function V in $\mathcal{D}$-model. Let us consider the negation of Case 1, that is, there is no event or action which make p be true happening at ω. Then there are two cases: either there is an event e' or action a' which make p false happening ar ω (Case 2) or not (Case 3). For Case 2, it entails that there is no disjunct in right part in formula (4.3) holding. So, we have $p \in \mathtt{eff}^-(H(\omega) \cup \{D(\omega)\})$ and $M, \omega \not\models \mathbf{X}p$ then $p \notin V(\omega+1)$. By the negation of Case 1, $p \notin \mathtt{eff}^+(H(\omega) \cup \{D(\omega)\})$, which entails that $p \notin V(\omega+1)$. So, Case 2 satisfies the progression criterion of Function V. For Case 3, it entails $p \in \mathtt{eff}^+(H(\omega) \cup \{D(\omega)\}) \cup \mathtt{eff}^-(H(\omega) \cup \{D(\omega)\})$. In this case, the formula (4.3) reduces to $\mathbf{X}p \leftrightarrow p$ and we have $p \in V(\omega+1)$ iff $p \in V(\omega)$. So, Case 3 satisfies the progression criterion of Function V. So we can conclude that ρ is a $\mathcal{D}$-model.

Now we need to prove $\rho \Vdash_{\mathcal{D}} \Delta$. Due to (4.6), we have $\rho \Vdash_{\mathcal{D}} \mathcal{B}(\Delta) \cup \mathcal{E}(\Delta)$. For $\alpha \notin \mathsf{Act}_0$, if $M, 0 \models \mathbf{X}^{t'}\mathtt{pre}(\alpha) \wedge \mathbf{X}^{d'}\mathtt{post}(\alpha)$, then $\rho \Vdash_{\mathcal{D}} (t', \alpha, d')$. Because of $t \leq t' < d' \leq d$, we get $\rho \Vdash_{\mathcal{D}} (t, \alpha, d)$. When it comes to basic action (t, a, d), there exists a time point t' such that $D(t') = a$ since (4.8). By the definition of satisfaction of databases (Definition 3.7), we prove $\rho \Vdash_{\mathcal{D}} \Delta$. $\square$

The next theorem states the equivalence between the consequence problem in belief-intention database and the validity problem in PLTL.

Theorem 4.2. *Given a coherent dynamic theory $\mathcal{D}$, Δ' is a $\mathcal{D}$-consequence of Δ iff $\Gamma^{\mathsf{L}}(\mathbf{max}(\mathsf{end}(\Delta), \mathsf{end}(\Delta')), \mathcal{D}) \to (\Gamma^{\mathsf{L}}(\Delta) \to \Gamma^{\mathsf{L}}(\Delta'))$ is valid where $\mathbf{max}(m, n)$ is the greater one for natural number m and n.*

Proof. Suppose $k = \mathbf{max}(\mathsf{end}(\Delta), \mathsf{end}(\Delta'))$.

"$\Rightarrow$": for all $\mathcal{D}$-models of Δ, we have $\Gamma^{\mathsf{L}}(\mathsf{end}(\Delta), \mathcal{D}) \wedge \Gamma^{\mathsf{L}}(\Delta)$. Then if these models are also models of Δ', then $(\Gamma^{\mathsf{L}}(\mathsf{end}(\Delta), \mathcal{D}) \wedge \Gamma^{\mathsf{L}}(\Delta)) \to (\Gamma^{\mathsf{L}}(\mathsf{end}(\Delta'), \mathcal{D}) \wedge \Gamma^{\mathsf{L}}(\Delta'))$. Because $\mathcal{D}$-model is infinite on time, no matter whether $\mathsf{end}(\Delta) \geq \mathsf{end}(\Delta')$ or not, $\Gamma^{\mathsf{L}}(k, \mathcal{D})$ is satisfied. Thus, we have $\Gamma^{\mathsf{L}}((k, \mathcal{D}) \to (\Gamma^{\mathsf{L}}(\Delta) \to \Gamma^{\mathsf{L}}(\Delta'))$.

"$\Leftarrow$": from the proof of Theorem 4.1, if $\Gamma^{\mathsf{L}}(k, \mathcal{D})$ is satisfied we can construct a $\mathcal{D}$-model ρ. Further if $\Gamma^{\mathsf{L}}(\Delta)$ is satisfied then $\rho \Vdash_{\mathcal{D}} \Delta$. Thus, if $\Gamma^{\mathsf{L}}(\Delta) \to \Gamma^{\mathsf{L}}(\Delta')$, then we have $\Delta \models_{\mathcal{D}} \Delta'$. $\square$

The translation Γ^{L} is based on the assumption of coherent dynamic theory. It will be wrong when the dynamic theory is not coherent, because the formula (4.3) cannot avoid the events or actions whose negative effect conflict with the positive effect of other events and actions, as shown in the following example.

Example 4.1. *[Example 3.2 continued] Let's recall the incoherent dynamic theory $\mathcal{D}'$:*

- $\mathtt{pre}(\mathsf{closeDoor}) = \mathsf{FrontDoor}$, $\mathtt{post}(\mathsf{closeDoor}) = \mathsf{DoorClosed}$
- $\mathtt{pre}(\mathsf{openDoorAuto}) = \mathsf{FrontDoor}$, $\mathtt{post}(\mathsf{openDoorAuto}) = \neg\mathsf{DoorClosed}$

Consider the case that Alice is in front of the door and intends to close the door at time point 0. Then we have the path ρ such that $V(0) = \{\mathsf{FrontDoor}\}, V(1) = \{\mathsf{FrontDoor}, \mathsf{DoorClosed}\}$ and $H(0) = \{\mathsf{openDoorAuto}\}$, $D(0) = \{\mathsf{closeDoor}\}$. It is clear that it is not a $\mathcal{D}'$-model because it violates the constraint formula 3.2 in the definition of $\mathcal{D}'$-model (Definition 3.5). However, the linear-time structure M built from ρ where $\varepsilon(s_0) = \{\mathsf{FrontDoor}, do_{\mathsf{closeDoor}}, h_{\mathsf{openDoorAuto}}\}$ and $\varepsilon(s_1) = \{\mathsf{FrontDoor}, \mathsf{DoorClosed}, h_{\mathsf{openDoorAuto}}\}$ satisfies the formula 4.3 where $\boldsymbol{X}\mathsf{DoorClosed}$

and $do_{\mathsf{closeDoor}}$ hold. It entails that there exists a linear-time structure satisfying the translating formula.

Indeed the coherence assumption guarantees that once h_e or do_a holds, there is no $h_{e'}$ or $do_{a'}$, whose effect contradicts the effect of e or a, holding.

Now we relax the coherence assumption from the dynamic theory. While formula (4.3) captures how propositional variables change positively, we still need to capture the negative change for propositional variables. That is, a propositional variable is false in the next state if and only if there either is a basic action or event which make it false or it is false in the current state and there is no basic action or event to make it false. To satisfy the constraint of $\mathcal{D}$-model, we define $R^{\neg}(\mathbb{P})$ as

$$\bigwedge_{p\in\mathbb{P}} \Big(\mathbf{X}\neg p \leftrightarrow \bigvee_{\substack{e\in\mathsf{Evt}_0\\ p\in\mathtt{eff}^-(e)}} h_e \vee \bigvee_{\substack{a\in\mathsf{Act}_0\\ p\in\mathtt{eff}^-(a)}} do_a \vee (\neg p \wedge \bigwedge_{\substack{e\in\mathsf{Evt}_0\\ p\in\mathtt{eff}^+(e)}} \neg h_e \wedge \bigwedge_{\substack{a\in\mathsf{Act}_0\\ p\in\mathtt{eff}^+(a)}} \neg do_a)\Big) \quad (4.9)$$

So, if there exists a $\mathcal{D}$-model of Δ, its corresponding linear-time structure will satisfy $R^{\neg}(\mathbb{P})\wedge$(4.3). The other way round, $R^{\neg}(\mathbb{P})\wedge$(4.3) guarantees that the path must satisfy the constraint formula (3.2) in the definition of $\mathcal{D}$-models (Definition 3.5).

Let us come back Example 4.1, by formula (4.3), we have $M, 0 \models \mathbf{X}\mathsf{DoorClosed}$ because of $M, 0 \models do_{\mathsf{closeDoor}}$; while by formula $R^{\neg}(\mathbb{P})$, we have $M, 0 \models \mathbf{X}\neg\mathsf{DoorClosed}$ because of $M, 0 \models h_{\mathsf{openDoorAuto}}$. So, there is a conflict on DoorClosed.

We next use operator $\mathbf{G}$ in PLTL to guarantee that an infinite path conforms to the constraint of $\mathcal{D}$-model.

Theorem 4.3. *Given any dynamic theory $\mathcal{D}$, a database Δ is $\mathcal{D}$-satisfiable iff $\boldsymbol{G}(\Gamma^{\mathsf{L}}(\mathcal{D}) \wedge R^{\neg}(\mathbb{P})) \wedge \Gamma^{\mathsf{L}}(\Delta)$ is satisfiable.*

Proof. Similar with the proof of Theorem 4.1. □

Theorem 4.4. *Given any dynamic theory $\mathcal{D}$, $\Delta \models_{\mathcal{D}} \Delta'$ iff $\boldsymbol{G}(\Gamma^{\mathsf{L}}(\mathcal{D}) \wedge R^{\neg}(\mathbb{P})) \rightarrow (\Gamma^{\mathsf{L}}(\Delta) \rightarrow \Gamma^{\mathsf{L}}(\Delta')$ is valid.*

Proof. Similar with the proof of Theorem 4.2. □

The problems deciding the refinement relation and the instrumentality relation are based on the satisfiability and consequence problems. According to the definition of refinement (Definition 3.9), the deciding-refinement problem can be decomposed into $\mathcal{D}$-satisfiability problems (Condition 1,2 and 4) and a $\mathcal{D}$-consequence problem (Condition 3). The following theorem states the correctness of the translation.

Theorem 4.5. *Given any dynamic theory $\mathcal{D}$, $\Delta \models_{\mathcal{D}} i \lhd J$, iff the following hold:*

- *for all $j \in J$, every $\boldsymbol{G}(\Gamma^{\mathsf{L}}(\mathcal{D}) \wedge R^{\neg}(\mathbb{P})) \wedge (\Gamma^{\mathsf{L}}(\Delta) \wedge \neg\Gamma^{\mathsf{L}}(\{j\})$ is satisfiable*
- $\boldsymbol{G}(\Gamma^{\mathsf{L}}(\mathcal{D}) \wedge R^{\neg}(\mathbb{P})) \wedge \Gamma^{\mathsf{L}}(\Delta \cup J)$ *is satisfiable*
- $\boldsymbol{G}(\Gamma^{\mathsf{L}}(\mathcal{D}) \wedge R^{\neg}(\mathbb{P})) \to \big(\Gamma^{\mathsf{L}}((\Delta \cup J) \setminus \{i\}) \to \Gamma^{\mathsf{L}}(\{j\})\big)$ *is valid*
- *for all $j \in J$, every $\boldsymbol{G}(\Gamma^{\mathsf{L}}(\mathcal{D}) \wedge R^{\neg}(\mathbb{P})) \wedge \Gamma^{\mathsf{L}}((\Delta \cup J) \setminus \{i, j\}) \wedge \neg\Gamma^{\mathsf{L}}(\{i\})$ is satisfiable*
- $\mathsf{end}(J) \leq \mathsf{end}(i)$

Proof. It is straightforward by the definition of intention refinement and the above theorems. □

Furthermore, by the definition of instrumentality (Definition 3.12), the deciding-instrumentality problem can be decomposed into a $\mathcal{D}$-satisfiability problem and a $\mathcal{D}$-consequence problem. By the following theorem, we show the translation is sound and complete.

Theorem 4.6. *Given any dynamic theory $\mathcal{D}$, $\Delta \models_{\mathcal{D}} J \gtrdot i$, iff the following hold:*

- $\boldsymbol{G}(\Gamma^{\mathsf{L}}(\mathcal{D}) \wedge R^{\neg}(\mathbb{P})) \wedge \Gamma^{\mathsf{L}}(\Delta \setminus J) \wedge \neg\Gamma^{\mathsf{L}}(\{i\})$ *is satisfiable*
- $\boldsymbol{G}(\Gamma^{\mathsf{L}}(\mathcal{D}) \wedge R^{\neg}(\mathbb{P})) \to (\Gamma^{\mathsf{L}}(\Delta \setminus J) \wedge (\bigvee_{j \in J} \Gamma^{\mathsf{L}}(\{j\})) \to \Gamma^{\mathsf{L}}(\{i\}))$ *is valid*
- $\mathsf{end}(J) \leq \mathsf{end}(i)$

Proof. It is straightforward by the definition of instrumentality and the above propositions. □

By Theorem 4.5 and Theorem 4.6, we can solve the decision problems of refinement and instrumentality by taking advantage of the efficient theorem provers of PLTL.

Remark 4.1. The complexity of the fragments of PLTL with different restrictions has been summarized in [Demri and Schnoebelen, 2002]. The satisfiability problem in PLTL without any operators but **X** is NP-complete while its validity problem is co-NP-complete.

For the database, time points as natural numbers are encoded in a binary way while they are considered as decimal in the size of the resulted PLTL formula. For example, if $(e, 10) \in \Delta$ then we have $\mathbf{X}^{10}h_e \in \Gamma^{\mathsf{L}}(\Delta)$. In this case, the time point 10 in the binary encoding occupies $\log 10$ on size in belief-intention database, while $\mathbf{X}^{10}$ which is a sequence of ten continuous operators **X** occupies 10 on size in PLTL. Therefore, the size of the resulting formula is not polynomial with respect to the size of the database. Notice that for the database, time points are encoded in a binary way and in consequence the translations are exponential. Although the satisfiability of PLTL is **NP**-complete, we cannot conclude that the satisfiability of database is NP-complete. Actually it is **PSPACE**-complete as shown in the previous chapter.

4.2 Translating to DL-PA

In this section, we introduce the translation from the decision problems of the databases to Dynamic Logic of Propositional Assignment (DL-PA). [Herzig et al., 2011] and [Balbiani et al., 2013] show that every DL-PA formula can be reduced to an equivalent propositional formula. We thereby further translate the decision problems into the satisfiability problem and the validity problem in classical propositional logic and then invoke efficient SAT solvers to solve the problem of deciding refinement.[2]

4.2.1 Syntax and Semantics of DL-PA

The first studies of assignments in the context of dynamic logic are due to [Tiomkin and Makowsky, 1985]. Dynamic Logic of Propositional Assignment (DL-PA) was proposed in [Herzig et al., 2011] which is an instantiation of Propositional Dynamic Logic (PDL) [Harel, 1984, Harel et al., 2000]. Let us first briefly recall the syntax and semantics of DL-PA.

Just as in PDL, DL-PA programs describe the evolution of the world, while DL-PA formulas describe the state of the world. Different from PDL, only two kinds of atomic programs are considered in DL-PA: for a propositional variable p, it can be assigned to true or false, written by $p \leftarrow \top$ and $p \leftarrow \bot$. Just as in PDL, these two kinds of atomic programs can be combined via program operators: sequential (';'), nondeterministic composition ('$\sqcup$'), finite iteration ('$*$'), and test ('?').

The models of DL-PA is simpler than PDL's Kripke models: we use valuations of classical propositional logic. A *valuation*[3] associates a truth value to each propositional variable in $\mathbb{P}$ and we identify valuations with subsets of $\mathbb{P}$ and use v, v_1, v_2, etc. to denote them. Note that the set of all valuations is $\mathbb{V} = 2^{\mathbb{P}}$ We use $v(p) = 1$ to denote $p \in v$ and $v(p) = 0$ to denote $p \notin v$. The assignment program $p \leftarrow \top$ adds the current valuation by p, while the assignment program $p \leftarrow \bot$ removes p from the valuation.

Propositional formulas are built from propositional variables by means of the standard boolean connectives in classical propositional logic. A DL-PA formula consists of propositional formulas and programs, denoted by φ, ψ, etc. For a given propositional formula φ, we use $\mathbb{P}_\varphi$ to denote the set of propositional variables occurring

[2] Several efficient SAT solvers can be found in the SAT Competition 2017 `http://www.satcompetition.org/` (accessed on 27 Sep. 2017).

[3] We use the same term "valuation" to denote the state in DL-PA and the belief-intention database. To distinguish them, we use v for the valuations of DL-PA and use V for those of the database.

in φ. For example, $\mathbb{P}_{p \wedge q} = \{p, q\}$. A given valuation determines the truth values of every propositional formula and we call a valuation where formula φ is true as $\varphi - valuation$. The language of DL-PA is defined as follows:

$$\varphi ::= p \mid \top \mid \bot \mid \varphi \vee \varphi \mid \langle \pi \rangle \varphi$$

$$\pi ::= p \leftarrow \top \mid p \leftarrow \bot \mid \pi ; \pi \mid \pi \sqcup \pi \mid \pi^* \mid \varphi ?$$

where p ranges over $\mathbb{P}$. The atomic programs of DL-PA is of the form $p \leftarrow \top$ and $p \leftarrow \bot$. The operators of DL-PA are sequential composition (';'), nondeterministic composition ('$\sqcup$'), finite iteration ('$*$'), and test ('?'). Particularly, we use $\mathop{;}$ and $\bigsqcup$ for the operators ; and $\sqcup$ applied on the sets, respectively. Formally, for a set A of propositional variables and an expression $f(p)$ related with propositional variable p, the formulas $\mathop{;}_{p \in A} f(p)$ and $\bigsqcup_{p \in A} f(p)$ respectively represent $f(p_1); f(p_2); \ldots; f(p_n)$ and $f(p_1) \sqcup f(p_2) \sqcup \ldots \sqcup f(p_n)$ where $p_1, p_2, \ldots, p_n \in A$. In particular, we use $\mathop{;}_k f$ to represent the sequence $f; f; \ldots; f$ with the number of k.

We abbreviate the logical connectives $\wedge, \rightarrow$ and $\leftrightarrow$ in the usual way. Moreover, we abbreviate $\neg\langle \pi \rangle \neg \varphi$ as $[\pi]\varphi$. Intuitively, formulas of the form $\langle \pi \rangle \varphi$ means that formula φ is true after *some* possible execution of π while $[\pi]\varphi$ means that formula φ is true after *every* possible execution of π.

The *length* of a formula φ, denoted by $|\varphi|$, is the number of symbols used to write down φ without "$\langle$","$\rangle$" and parentheses. The length of a program π, denoted by $|\pi|$, is defined in the same way. For example $|\langle p \leftarrow \top \rangle (p \rightarrow q)| = 3 + 3 = 6$.

DL-PA programs are interpreted by means of relations between valuations. The atomic programs $p \leftarrow \top$ and $p \leftarrow \bot$ update valuations just as update actions do (see the preceding section), and complex programs are interpreted just as in PDL by mutual recursion. Next we give the interpretation of formulas and programs where $\circ$ is relation composition.

$$||p|| = \{v \mid p \in v\}$$

$$
\begin{aligned}
||\top|| &= \mathbb{V} \;=\; 2^{\mathbb{P}} \\
||\bot|| &= \emptyset \\
||\neg\varphi|| &= 2^{\mathbb{P}} \setminus ||\varphi|| \\
||\varphi \vee \psi|| &= ||\varphi|| \cup ||\psi|| \\
||\langle\pi\rangle\varphi|| &= \big\{v \mid \exists v_1 \text{ s.t. } \langle v, v_1\rangle \in ||\pi|| \text{ and } v_1 \in ||\varphi||\big\} \\
||\alpha|| &= \big\{\langle v_1, v_2\rangle \mid v_2 = v_1 \diamond \{\alpha\}\big\} \\
||\pi;\pi'|| &= ||\pi|| \circ ||\pi'|| \\
||\pi \sqcup \pi'|| &= ||\pi|| \cup ||\pi'|| \\
||\pi^*|| &= \bigcap_{k\in\mathbb{N}_0} ||\pi||^k \\
||\varphi?|| &= \big\{\langle v, v\rangle \mid v \in ||\varphi||\big\}
\end{aligned}
$$

A formula φ is DL-PA *valid* iff $||\varphi|| = 2^{\mathbb{P}} = \mathbb{V}$. It is DL-PA *satisfiable* iff $||\varphi|| \neq \emptyset$. For example, the formulas $\langle p \leftarrow \bot\rangle\top$ and $\langle p \leftarrow \bot\rangle\neg p$ are all DL-PA valid, entailing that they are all DL-PA satisfiable.

In particular, the truth test $\top?$ means doing nothing. Moreover, the conditional program "if φ then π_1 else π_2" is expressed by $(\varphi?;\pi_1) \sqcup (\neg\varphi?;\pi_2)$. Specially, the conditional program without negative branch "if φ then π_1" is represented by $(\varphi?;\pi_1) \sqcup \neg\varphi?$ instead of $\varphi?;\pi_1$. Furthermore, the loop program "while φ do π" is expressed by $(\varphi?\pi)^*;\neg\varphi?$.

We abbreviate the assignments of literals to variables as follows:

$$
\begin{aligned}
p \leftarrow q &= \; \textsf{if } q \textsf{ then } p \leftarrow \top \textsf{ else } p \leftarrow \bot \\
&= (q?; p \leftarrow \top) \sqcup (\neg q?; p \leftarrow \bot) \\
p \leftarrow \neg q &= \; \textsf{if } q \textsf{ then } p \leftarrow \bot \textsf{ else } p \leftarrow \top \\
&= (q?; p \leftarrow \bot) \sqcup (\neg q?; p \leftarrow \top)
\end{aligned}
$$

Intuitively, $p \leftarrow q$ assigns to p the truth the value of q: if q is true then p will be true otherwise p will be false. While the program $p \leftarrow \neg q$ assigns to p the truth value of $\neg q$.

To change the truth value of some propositional variables, [Herzig, 2014] define a program: for a set of propositional variables P,

$$\mathsf{Flip}(P) = \mathop{;}_{p \in P} \big(p \leftarrow \top \sqcup p \leftarrow \bot\big).$$

The program $\mathsf{Flip}(P); \varphi?$ means to nondeterministically change the truth value of some variables in the set P to satisfy the formula φ.

Observe that if p does not occur in φ then formulas such as $\varphi \to \langle p \leftarrow \top \rangle \varphi$ and $\varphi \to \langle p \leftarrow \bot \rangle \varphi$ are valid.

A distinguishing feature of DL-PA is that its dynamic operators can be eliminated (which is not possible in PDL). Just as for QBF, the resulting formula may be exponentially longer than the original formula.

Theorem 4.7 ([Balbiani et al., 2013]). *Every* DL-PA *formula has an equivalent boolean formula.*

For example, the DL-PA formula $\langle p \leftarrow \bot \rangle (\neg p \wedge \neg q)$ is equivalent to the formula $\langle p \leftarrow \bot \rangle \neg p \wedge \langle p \leftarrow \bot \rangle \neg q$, which is by itself equivalent to $\top \wedge \neg q$. Therefore, the DL-PA formula $\langle p \leftarrow \bot \rangle (\neg p \wedge \neg q)$ is reduced to the boolean formula $\neg q$.

Every sequence of assignment programs $\pi_1; \ldots; \pi_n$ is a deterministic program that is always executable: for a given v, there is exactly one v' such that $\langle v, v' \rangle \in ||\pi_1; \ldots; \pi_n||$. Moreover, the order of the π_i in a sequential composition is irrelevant when a set of assignment programs $\{\pi_1, \ldots, \pi_n\}$ is consistent.

4.2.2 Translation to DL-PA

Before defining the translation from the belief-intention database to DL-PA, we use the following abbreviations for programs. Just as for the reduction to PLTL in Section 4.1, we use the same auxiliary propositional variables for basic actions, events and their pre- and postcondition: $\mathbb{P}_d = \{do_a \mid a \in \mathsf{Act}_0\}$, $\mathbb{P}_h = \{h_e \mid e \in \mathsf{Evt}_0\}$ and $\mathbb{P}_c = \{pre_\alpha, post_\alpha, pre_e, post_e \mid \alpha \in \mathsf{Act}, e \in \mathsf{Evt}_0\}$. Similar to the translation $\Gamma^{\mathsf{L}}(\mathcal{D})$ (Definition 4.1), at every time point the following formulas should hold:

$$
\begin{aligned}
\Phi_d &= \bigwedge_{a \in \mathsf{Act}_0} \big(do_a \to pre_a)\big) \wedge \bigvee_{a \in \mathsf{Act}_0} do_a \wedge \bigwedge_{a,b \in \mathsf{Act}_0, a \neq b} \neg(do_a \wedge do_b) \\
\Phi_h &= \bigwedge_{e \in \mathsf{Evt}_0} \big((h_e \leftrightarrow pre_e)\big) \\
\Phi_c &= \bigwedge_{\alpha \in \mathsf{Act}} \big((pre_\alpha \leftrightarrow \mathtt{pre}(\alpha)) \wedge (post_\alpha \leftrightarrow \mathtt{post}(\alpha))\big) \wedge \\
&\quad \bigwedge_{e \in \mathsf{Act}} \big((pre_e \leftrightarrow \mathtt{pre}(e)) \wedge (post_e \leftrightarrow \mathtt{post}(e))\big)
\end{aligned}
$$

The formula Φ_d means that at every time point (or valuation), there is one and only one basic action chosen to perform and if a basic action is chosen then its precondition must be satisfied. The formula Φ_h means that events happen if and only if their precondition are satisfied and the formula Φ_c links the auxiliary variables in $\mathbb{P}_c$ with the pre- and postcondition of basic actions and events.

We now define a function, $\mathbb{P}_c : 2^{\mathbb{P}} \longrightarrow 2^{\mathbb{P}_c}$, to extract the auxiliary propositional variables of pre- and postcondition from valuations. Formally, for a time point ω and a $\mathcal{D}$-model $\langle V, H, D \rangle$,

$$\mathbb{P}_c(V(\omega)) = \{pre_x, post_y \mid V(\omega) \models \mathtt{pre}(x), V(\omega) \models \mathtt{post}(y) \text{ and } x, y \in \mathsf{Act}_0 \cup \mathsf{Evt}_0\}.$$

Next we introduce a program to capture the effect of basic actions and events. For every $x \in \mathsf{Act}_0 \cup \mathsf{Evt}_0$, we define the program Effect to capture the effect of x as follows:

$$\mathsf{Effect}(x) = \mathop{;}_{p \in \mathtt{eff}^+(x)} (p \leftarrow \top); \mathop{;}_{q \in \mathtt{eff}^-(x)} (q \leftarrow \bot)$$

For the basic action of buying ticket online buyWeb, the program $\mathsf{Effect}(buyweb)$ is $\mathsf{PaidWeb} \leftarrow \top$.

For a coherent dynamic theory, the current valuation and the action being performed determine the next valuation. So, the effect of basic actions and events is achieved by a sequence of assignment programs on propositional variables: the variables in the positive effect will be assigned to true while those variables in the negative effect will be assigned to false.

$$\begin{aligned}
\mathsf{Actions} &= \mathsf{Flip}(\mathbb{P}_d); \Phi_d?; \bigsqcup_{a \in \mathsf{Act}_0} \big(do_a?; \mathsf{Effect}(a)\big) \\
\mathsf{Events} &= \mathop{;}_{e \in \mathsf{Evt}_0} \big((h_e?; \mathsf{Effect}(e)) \sqcup \neg h_e?\big) \\
\mathsf{Update} &= \mathsf{Flip}(\mathbb{P}_h \cup \mathbb{P}_c); (\Phi_h \wedge \Phi_c)? \\
\mathsf{Dyn} &= \mathsf{Actions}; \mathsf{Events}; \mathsf{Update}
\end{aligned}$$

As there is no operator for concurrence in DL-PA, we use sequential programs to capture the concurrence of basic actions and events. Basic actions and events are different: at every time point there is one and only one basic action performed while events can occur simultaneously if their preconditions are satisfied. We model them in different ways: for basic actions, the agent is proactive in choosing actions by means of the program $\mathsf{Flip}(\mathbb{P}_d)$ and the chosen action has to satisfy the formula Φ_d; whereas events occur reactively which is captured by the program "if h_e then $\mathsf{Effect}(e)$". After the program Actions, the valuation actually changes because of $\mathsf{Effect}(a)$ and the formula $\mathtt{pre}(e)$ may become unsatisfied even though it is satisfied before performing action a. So, we use h_e, which is equivalent to $\mathtt{pre}(e)$ before performing action a, as the condition to determine whether the event will happen. Intuitively, the program Actions chooses a basic action to perform. For every event with a satisfied precondition, the program Events actives it. Finally, the program Update changes auxiliary variables in $\mathbb{P}_h \cup \mathbb{P}_c$ to satisfy the formula $\Phi_h \wedge \Phi_c$ so that these auxiliary variables can correctly capture the pre- and postcondition of actions and events. Intuitively, the program Dyn describes the minimal change of valuations which satisfying the frame axiom according to the definition of the dynamic theory model (Definition 3.5).

Note that if the dynamic theory is coherent, there is no conflict between the effects of actions and events, so $||\mathsf{Actions};\mathsf{Events}||$ is equivalent to $||\mathsf{Events};\mathsf{Actions}||$.

The next proposition states that Dyn captures the progression of valuations in the $\mathcal{D}$-model.

Proposition 4.8. *Given a coherent dynamic theory $\mathcal{D}$ and a $\mathcal{D}$-model $\langle V, H, D\rangle$, for every time point ω, there exist $(v, v') \in ||\mathsf{Dyn}||$ where $v \cap \mathbb{P} = V(\omega)$ and $v' \cap \mathbb{P} = V(\omega + 1)$, $v' \cap \mathbb{P}_h = H(\omega + 1)$, $v' \cap \mathbb{P}_d = D(\omega)$, $v' \cap \mathbb{P}_c = \mathbb{P}_c(V(\omega + 1))$.*

Proof. Given a $\mathcal{D}$-model $\langle V, H, D\rangle$, we will construct a pair $(v, v') \in ||\mathsf{Dyn}||$ by induction. According to the semantics of DL-PA, the following holds: for every formula φ,

$$\big\langle \mathsf{Flip}(\mathbb{P}_d); \Phi_d? \big\rangle \big\langle \bigsqcup_{a \in \mathsf{Act}_0} \big(do_a?; \mathsf{Effect}(a)\big); \mathsf{Events} \big\rangle \big\langle \mathsf{Update} \big\rangle \varphi \leftrightarrow \big\langle \mathsf{Dyn} \big\rangle \varphi.$$

For the purpose of simplicity, we use π_1, π_2, π_3 to denote these three programs, respectively. Then we will construct a sequence of valuations v, v_1, v_2, v' according to the sequential partition π_1, π_2, π_3 of the program Dyn.

Base case. When $\omega = 0$, let valuation $v = V(0) \cup \{h_e \mid e \in H(0)\} \cup \mathbb{P}_c(V(0))$. Then we construct $v_1 = v \cup \{do_a \mid a = D(0)\}$. By the definition of $\mathcal{D}$-models (Definition 3.5), we have $V(0) \models \mathtt{pre}(D(0))$. Because $\mathtt{pre}(a) \leftrightarrow pre_a$ and $do_a \to pre_a$ are in the formula Φ_d, we have $(v, v_1) \in ||\pi_1||$. Next we construct

$$v_2 = (v_1 \setminus \mathtt{eff}^-(H(0) \cup \{D(0)\})) \cup \mathtt{eff}^+(H(0) \cup \{D(0)\}).$$

As h_e and do_a depends on $H(0)$ and $D(0)$ respectively and $\mathsf{Effect}(x)$ makes p in $\mathtt{eff}^+(x)$ be true and q in $\mathtt{eff}^-(x)$ be false, we have $(v_1, v_2) \in ||\pi_2||$. Now we construct

$$v' = (v_2 \setminus (\mathbb{P}_h \cup \mathbb{P}_c)) \cup \{h_e \mid e \in H(1)\} \cup \mathbb{P}_c(V(1)).$$

Because $\Phi_h \wedge \Phi_c$ is satisfied, $(v_2, v') \in ||\pi_3||$. So, we have $(v, v') \in ||\pi_1; \pi_2; \pi_3||$ which entails that $(v, v') \in ||\mathsf{Dyn}||$.

Inductive step. Suppose when $t = k$ the proposition holds. When $\omega = k + 1$, let valuation

$$v = V(k+1) \cup \{h_e \mid e \in H(k+1)\} \cup \mathbb{P}_c(V(k+1)) \cup \{do_a \mid a = D(k)\}.$$

Next we construct $v_1 = (v \setminus \mathbb{P}_d) \cup \{do_a \mid a = D(k+1)$. By the definition of $\mathcal{D}$-models (Definition 3.5), we have $V(k+1) \models \mathtt{pre}(D(k+1))$ and the formula Φ_d is satisfied, entailing that $(v, v_1) \in ||\pi_1||$. Then we construct

$$v_2 = (v_1 \setminus \mathtt{eff}^{-}(H(k+1) \cup \{D(k+1)\})) \cup \mathtt{eff}^{+}(H(k+1) \cup \{D(k+1)\}).$$

Due to $\mathsf{Effect}(x)$, we have $(v_1, v_2) \in ||\pi_2||$. Now we construct

$$v' = (v_2 \setminus (\mathbb{P}_h \cup \mathbb{P}_c)) \cup \{h_e \mid e \in H(k+2)\} \cup \mathbb{P}_c(V(k+2)).$$

It is not difficult to conclude that $(v_2, v') \in ||\pi_3||$. So, we have $(v, v') \in ||\mathsf{Dyn}||$. □

The next proposition states that a sequence of program Dyn can generate a bounded dynamic theory model.

Proposition 4.9. *Given a coherent dynamic theory $\mathcal{D}$ and a natural number k, the formula $\Phi_h \wedge \Phi_c \wedge \langle \,;_k \mathsf{Dyn} \rangle \top$ is* DL-PA *satisfiable iff there exists a bounded $\mathcal{D}$-model $\overline{\rho} = \langle V, H, D, k \rangle$.*

Proof. Proposition 4.8 states that two time points in a $\mathcal{D}$-model are linked by the program Dyn. Then the sequence $;_k \mathsf{Dyn}$ generates a sequence of time points in a bounded $\mathcal{D}$-model with bound k. In particular, the initial valuation should satisfy the formula $\Phi_h \wedge \Phi_c$ to guarantee that every event is reactive. □

As intentions are defined in a flexible way, we need to represent whether the high-level actions actually start and finish. To do so, for every intention i in the database, we introduce pairs of auxiliary propositional variables, b_i and f_i, where b_i means the corresponding action of intention i has been already started; f_i means the action has been finished.

$$\begin{aligned}
\mathsf{BegInt}(t,\alpha,d) &= \textsf{if } pre_\alpha \textsf{ then } b_{(t,\alpha,d)} \leftarrow \top \\
&= (pre_\alpha?; b_{(t,\alpha,d)} \leftarrow \top) \sqcup \neg pre_\alpha? \\
\mathsf{FinHInt}(t,\alpha,d) &= \textsf{if } b_{(t,\alpha,d)} \wedge post_\alpha \textsf{ then } f_{(t,\alpha,d)} \leftarrow \top \\
&= ((b_{(t,\alpha,d)} \wedge post_\alpha)?; f_{(t,\alpha,d)} \leftarrow \top) \sqcup \neg(b_{(t,\alpha,d)} \wedge post_\alpha)? \\
\mathsf{FinBInt}(t,a,d) &= \textsf{if } do_a \textsf{ then } f_{(t,a,d)} \leftarrow \top \\
&= (do_a?; f_{(t,a,d)} \leftarrow \top) \cup \neg do_a?
\end{aligned}$$

Intuitively, the program $\mathsf{BegInt}(t,\alpha,d)$ means that if the precondition of α is satisfied, then intention (t,α,d) will be started while the program $\mathsf{FinHInt}(t,\alpha,d)$ means that if intention (t,α,d) has already been started and the postcondition of high-level action α has been satisfied, then the intention will be finished. For basic intentions, the program $\mathsf{FinBInt}(t,a,d)$ means that if basic action a has been done then intention (t,α,d) will be finished.

Based on the above programs, we can introduce the programs of starting and finishing intentions for every time point. For all time points $\omega \in \mathbb{N}^0$ and all intentions $i = (t,\alpha,d) \in \Delta$:

$$\begin{aligned}
\mathsf{Begin}(\omega) &= \mathop{;}_{\substack{i\in\Delta \\ t\leq\omega<d}} \mathsf{BegInt}(i) \\
\mathsf{Finish}(\omega) &= \mathop{;}_{\substack{i\in\Delta \\ \alpha\notin\mathsf{Act}_0 \\ t<\omega\leq d}} \mathsf{FinHInt}(i); \mathop{;}_{\substack{i^0\in\Delta \\ \alpha\in\mathsf{Act}_0 \\ t<\omega\leq d}} \mathsf{FinBInt}(i^0)
\end{aligned}$$

Intuitively, the program $\mathsf{Begin}(\omega)$ starts the intentions which can be started in time point ω and the program $\mathsf{Finish}(\omega)$ finishes the intentions which can be finished in time point ω. At time point ω, only those intentions such that $t \leq \omega < d$ are possible to start and only those intention with $t < \omega \leq d$ are possible to finish.

As we stated before, DL-PA programs describe the evolution of the world while DL-PA formulas describe the state of the world. We have already defined the programs which capture how the basic actions and events change the world and which decide if an intention is started and finished, then we describe how the valuation should satisfy the database by means of DL-PA formulas:

$$\tau_{\mathsf{S}}^{\mathsf{D}}(\Delta,\omega) \;=\; \bigwedge_{(\omega,\varphi)\in\Delta} \varphi \wedge \bigwedge_{(\omega,e)\in\Delta} h_e \wedge \bigwedge_{(\omega,\bar{e})\in\Delta} \neg h_e \wedge \bigwedge_{i=(t,\alpha,\omega)\in\Delta} f_i$$

Intuitively, the formula $\Gamma_{\mathsf{S}}^{\mathsf{D}}(\Delta,\omega)$ describes that at time point ω, both the beliefs and the (non-)occurrence of events labeled by ω should be satisfied and that all intentions whose deadline is ω should be accomplished.

$$\mathrm{Init} \;= \bigwedge_{i\in\Delta} \big(\neg b_i \wedge \neg f_i\big) \wedge \Phi_h \wedge \Phi_c$$

The formula Init enforces that no intention is started or finished initially.

Given a dynamic theory $\mathcal{D}$ and database Δ, we can define the translation for the satisfiability inductively:

$$\Gamma_{\mathsf{S}}^{\mathsf{D}}(\mathcal{D},\Delta,\omega) = \begin{cases} \big\langle \mathsf{Dyn} \big\rangle \top, & \omega = \mathsf{end}(\Delta)+1 \\ \big\langle \mathsf{Progress}(\omega) \big\rangle \big(\tau_{\mathsf{S}}^{\mathsf{D}}(\Delta,\omega) \wedge \Gamma_{\mathsf{S}}^{\mathsf{D}}(\mathcal{D},\Delta,\omega{+}1)\big), & 1 \le \omega \le \mathsf{end}(\Delta) \\ \mathrm{Init} \wedge \tau_{\mathsf{S}}^{\mathsf{D}}(\Delta,0) \wedge \big\langle \mathsf{Begin}(0) \big\rangle \Gamma_{\mathsf{S}}^{\mathsf{D}}(\mathcal{D},\Delta,1), & \omega = 0 \end{cases}$$

where $\mathsf{Progress}(\omega) = \mathsf{Dyn}; \mathsf{Finish}(\omega); \mathsf{Begin}(\omega)$.

When $\omega = 0$, it is the initial case that the beliefs on valuation and environmental change are true and only those intentions whose corresponding precondition is satisfied will be started. With the time running, some action is performed and some events occur and then their effect will change the valuation. Meanwhile, some intentions are finished while some intentions are started. Because the finishing-check of intentions depends on the starting-check of intentions, the program $\mathsf{Finish}(\omega)$ has to be ahead of $\mathsf{Begin}(\omega)$. For a coherent dynamic theory, we only need to consider a bounded $\mathcal{D}$-model with a bound $\mathsf{end}(\Delta)$.

Theorem 4.10. *Given a coherent dynamic theory $\mathcal{D}$, a database Δ is $\mathcal{D}$-satisfiable iff the formula $\Gamma_{\mathsf{S}}^{\mathsf{D}}(\mathcal{D},\Delta,0)$ is* DL-PA *satisfiable.*

Proof. "$\Rightarrow$:" Suppose there is a $\mathcal{D}$-model $\rho = \langle V, H, D \rangle$. Let $v_0 = V(0) \cup \{h_e \mid e \in H(0)\} \cup \mathbb{P}_c(V(0))$. Then we will prove $v_0 \in ||\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D}, \Delta, 0)||$.

By the definition of satisfaction of a database (Definition 3.7), we have $V(0) \models \bigwedge_{(0,\varphi)\in\Delta} \varphi \wedge \bigwedge_{(0,e)\in\Delta} h_e \wedge \bigwedge_{(0,\bar{e})\in\Delta} \neg h_e$. As there is no intention such that $\langle t, \alpha, 0\rangle \in \Delta$, we have $v_0 \in ||\text{Init} \wedge \tau^{\mathsf{D}}_{\mathsf{S}}(\Delta, 0)||$. Next we construct $v'_0 = v_0 \cup \{b_{0,\alpha,t} \mid V(0) \models \mathtt{pre}(\alpha), \langle 0, \alpha, t\rangle \in \Delta\}$. Due to Φ_c, we have $(v_0, v'_0) \in ||\mathsf{Begin}(0)||$. Then we construct

$$v''_0 = V(1) \cup \{h_e \mid e \in H(1)\} \cup \mathbb{P}_c(V(1)) \cup \{do_a \mid a = D(0)\}.$$

By Proposition 4.8, we have $(v'_0, v''_0) \in ||\mathsf{Dyn}||$. Now we construct

$$\begin{aligned} v_1 = v''_0 &\cup \{f_{(0,\alpha,1)} \mid V(1) \models \mathtt{post}(\alpha), b_{(0,\alpha,1)} \in v''_0, (0,\alpha,1) \in \Delta, \alpha \in \mathsf{Act} \setminus \mathsf{Act}_0\} \\ &\cup \{f_{(0,a,1)} \mid D(0) = a, (0, a, 1) \in \Delta\} \\ &\cup \{b_{(1,\alpha,d)} \mid V(1) \models \mathtt{pre}(\alpha), (1, \alpha, d) \in \Delta, \alpha \in \mathsf{Act} \setminus \mathsf{Act}_0\} \end{aligned}$$

It is easy to check that $(v''_0, v_1) \in ||\mathsf{Finish}(1); \mathsf{Begin}(1)||$.

For $(0, \alpha, 1) \in \Delta$ where $\alpha \in \mathsf{Act} \setminus \mathsf{Act}_0$, because of $\rho \Vdash_{\mathcal{D}} (0, \alpha, 1)$, we have $V(1) \models \mathtt{post}(\alpha)$ and $V(0) \models \mathtt{pre}(\alpha)$ which entails $b_{(0,\alpha,1)} \in v''_0$. So, we have $f_{(0,\alpha,1)} \in v_1$. For $(0, a, 1) \in \Delta$ where $a \in \mathsf{Act}_0$, we have $D(0) = a$ and then have $f_{(0,a,1)} \in v_1$. So we have $(v_0, v_1) \in ||\mathsf{Begin}(0); \mathsf{Progress}(1)||$. As $v_1 \cap \mathbb{P} = V(1)$, it is easy to check that $v_1 \in ||\tau^{\mathsf{D}}_{\mathsf{S}}(\Delta, 1)||$.

Next, for the case $1 \leq \omega \leq \mathsf{end}(\Delta)$, we will show that there is a sequence of valuations $v_1, \ldots, v_{\mathsf{end}(\Delta)}$ where for $k = 1, \ldots, \mathsf{end}(\Delta) - 1$, $v_k \in ||\tau^{\mathsf{D}}_{\mathsf{S}}(\Delta, k)||$ and $(v_k, v_{k+1}) \in ||\mathsf{Progress}(k + 1)||$ by induction.

Base case. We construct

$$v'_1 = V(2) \cup \{h_e \mid e \in H(2)\} \cup \mathbb{P}_c(V(2)) \cup \{do_a \mid a = D(1)\}.$$

By Proposition 4.8, we have $(v_1, v'_1) \in ||\mathsf{Dyn}||$. Now we construct

$$v_2 = v'_1 \cup \{f_{(t,\alpha,d)} \mid V(2) \models \mathtt{post}(\alpha), b_{(t,\alpha,d)} \in v'_1, t < 2 \leq d, \alpha \notin \mathsf{Act}_0\}$$

$$\cup \{f_{(t,a,d)} \mid D(1) = a, t < 2 \leq d, (t,a,d) \in \Delta\}$$
$$\cup \{b_{(t,\alpha,d)} \mid V(2) \models \texttt{pre}(\alpha), t \leq 2 < d, (t,\alpha,d) \in \Delta\}$$

It is easy to check that $(v_1', v_2) \in ||\mathsf{Finish}(2); \mathsf{Begin}(2)||$.

For $(t,\alpha,d) \in \Delta$ where $\alpha \in \mathsf{Act} \setminus \mathsf{Act}_0$, if $d = 2$, because of $\rho \Vdash_{\mathcal{D}} (t,\alpha,2)$, we have $V(2) \models \texttt{post}(\alpha)$ and there exists $t \leq t' < 2$ such that $V(t') \models \texttt{pre}(\alpha)$ which entails that either $b_{(t,\alpha,2)} \in v_0'$ or $b_{(t,\alpha,2)} \in v_1$. As there is no program to make $b_{(t,\alpha,2)}$ be false, if $b_{(t,\alpha,2)} \in v_0'$ then $b_{(t,\alpha,2)} \in v_1$. So, we have $f_{(t,\alpha,2)} \in v_2$.

For $(t,a,d) \in \Delta$ where $a \in \mathsf{Act}_0$ and $t < 2 \leq d$, because $\rho \Vdash_{\mathcal{D}} (t,a,d)$, there exists $0 \leq t' \leq 1$ such that $D(t') = a$. As there is no program to make $f_{(t,a,d)}$ be false, if $t' < 2$, we have $f_{(t,a,d)} \in v_2$. So, $v_2 \in ||\bigwedge_{i=(t,\alpha,2)\in\Delta} f_i||$. As $v_2 \cap \mathbb{P} = V(2)$, it is easy to check that $v_2 \in ||\tau_{\mathsf{S}}^{\mathsf{D}}(\Delta, 2)||$. Therefore, we have $(v_1, v_2) \in ||\mathsf{Progress}(2)||$.

Inductive step. As an inductive hypothesis, we suppose a valuation sequence $v_1, \ldots, v_n$ where $2 \leq n \leq \mathsf{end}(\Delta) - 1$ and for $k = 1, \ldots, n-1$, $v_k \in ||\tau_{\mathsf{S}}^{\mathsf{D}}(\Delta, k)||$ and $(v_k, v_{k+1}) \in ||\mathsf{Progress}(k+1)||$.

Now we will construct a valuation v_{n+1} such that $v_{n+1} \in ||\tau_{\mathsf{S}}^{\mathsf{D}}(\Delta, n+1)||$ and $(v_n, v_{n+1}) \in ||\mathsf{Progress}(n+1)||$.

We first construct

$$v_n' = V(n+1) \cup \{h_e \mid e \in H(n+1)\} \cup \mathbb{P}_c(V(n+1)) \cup \{do_a \mid a = D(n)\}.$$

By Proposition 4.8, we have $(v_n, v_n') \in ||\mathsf{Dyn}||$. Now we construct

$$v_{n+1} = v_n' \cup \{f_{(t,\alpha,d)} \mid V(n+1) \models \texttt{post}(\alpha), b_{(t,\alpha,d)} \in v_n', t < n+1 \leq d, \alpha \notin \mathsf{Act}_0\}$$
$$\cup \{f_{(t,a,d)} \mid D(n) = a, t < n+1 \leq d, (t,a,d) \in \Delta\}$$
$$\cup \{b_{(t,\alpha,d)} \mid V(n+1) \models \texttt{pre}(\alpha), t \leq n+1 < d, (t,\alpha,d) \in \Delta\}$$

It is easy to check that $(v_n', v_{n+1}) \in ||\mathsf{Finish}(n+1); \mathsf{Begin}(n+1)||$.

For $i = (t, \alpha, d) \in \Delta$ where $\alpha \in \mathsf{Act} \setminus \mathsf{Act}_0$ and $t < n+1 \leq d$, because of $\rho \Vdash_{\mathcal{D}} i$, there exist t', d' such that $t \leq t' < d' \leq d$ and such that $V(d') \models \mathtt{post}(\alpha)$ and $V(t') \models \mathtt{pre}(\alpha)$. Suppose t' and d' is the smallest time points which start and finish the intention i. That is, there is no $t \leq t'' < t'$ and $t \leq d'' < d'$ such that $V(t'') \models \mathtt{pre}(\alpha)$ and $V(d'') \models \mathtt{post}(\alpha)$. As $\mathsf{BegInt}(i)$ is a subprogram of $\mathsf{Begin}(t')$, we have if $t' = 0$ $b_i \in v'_0$ otherwise $b_i \in v_{t'}$. As there is no program to make b_i be false, if $t' < n+1$, we have $b_i \in v_n$. If $t' = n+1$, then $b_i \in v_{n+1}$. On the other hand, because $\mathsf{FinHInt}(i)$ is a subprogram of $\mathsf{Finish}(d')$ and there is no program to make f_i be false, if $d' < n+1$, then $f_i \in v_n$ and $f_i \in v_{n+1}$. If $d' = n+1$, then $f_i \in v_{n+1}$. So, if $d = n+1$, we have $f_{(t,\alpha,n+1)} \in v_{n+1}$.

For $i^0 = (t, a, d) \in \Delta$ where $a \in \mathsf{Act}_0$ and $t < n+1 \leq d$, because of $\rho \Vdash_{\mathcal{D}} i^0$, there exists t' such that$t \leq t' < d$ and $D(t') = a$. Suppose t' is the smallest time point in which basic action a is performed. In other words, there is no $t \leq t'' < t'$ such that $D(t'') = a$. As $\mathsf{FinBInt}(i^0)$ is a subprogram of $\mathsf{Finish}(t')$, if $t' < n$, then $f_{i^0} \in v_{n-1}$, $f_{i^0} \in v_n$ and $f_{i^0} \in v_{n+1}$. If $t' = n$, then $f_i \in v_{n+1}$. So, if $d = n+1$, we have $f_{(t,a,n+1)} \in v_{n+1}$.

As $v_{n+1} \cap \mathbb{P} = V(n+1)$, it is easy to check that $v_{n+1} \in ||\tau^{\mathsf{D}}_{\mathsf{S}}(\Delta, n+1)||$. Thus, we have $(v_n, v_{n+1}) \in ||\mathsf{Progress}(n+1)||$.

Therefore, there is a sequence of valuations $v_1, \ldots, v_{\mathsf{end}(\Delta)}$ such that for $k = 1, \ldots, \mathsf{end}(\Delta){-}1$, $v_k \in ||\tau^{\mathsf{D}}_{\mathsf{S}}(\Delta, k)||$ and $(v_k, v_{k+1}) \in ||\mathsf{Progress}(k+1)||$.

Next we construct

$$\begin{aligned} v_{\mathsf{end}(\Delta)+1} = v_{\mathsf{end}(\Delta)} &\cup \{h_e \mid e \in H(\mathsf{end}(\Delta)+1)\} \cup \mathbb{P}_c(V(\mathsf{end}(\Delta)+1)) \\ &\cup \{do_a \mid a = D(\mathsf{end}(\Delta))\}. \end{aligned}$$

By Proposition 4.8, we have $(v_{\mathsf{end}(\Delta)}, v_{\mathsf{end}(\Delta)+1}) \in ||\mathsf{Dyn}||$. So, $v_0 \in ||\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D}, \Delta, 0)||$ and the DL-PA formula $\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D}, \Delta, 0)$ is satisfiable.

"$\Leftarrow$": Suppose $v_0 \in ||\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D}, \Delta, 0)||$ and $n = \mathsf{end}(\Delta)$. Then there exists a valuation sequence $v_0, v_1, \ldots, v_n, v_{n+1}$ such that:

(i) $(v_0, v_1) \in ||\mathsf{Begin}(0); \mathsf{Progress}(1)||$ and $v_0 \in ||\tau^{\mathsf{D}}_{\mathsf{S}}(\Delta, 0)||$;

(ii) for $k = 1, \ldots, n-1$, $(v_k, v_{k+1}) \in ||\mathsf{Progress}(k+1)||$ and $v_k \in ||\tau^{\mathsf{D}}_{\mathsf{S}}(\Delta, k)||$;

(iii) $(v_n, v_{n+1}) \in ||\mathsf{Dyn}||$.

Now we construct a bounded path $\overline{\rho} = \langle V, H, D, n\rangle$ where for $\omega \in \mathbb{N}_n$, $V(\omega) = v_\omega \cap \mathbb{P}$, $H(\omega) = v_\omega \cap \mathbb{P}_h$, $D(\omega) \in v_{\omega+1} \cap \mathbb{P}_d$ and $V(n+1) = v_{n+1} \cap \mathbb{P}$. Then we will first show that $\overline{\rho}$ is a bounded model and then prove that $\overline{\rho} \Vdash_{\mathcal{D}} \Delta$.

The program $\mathsf{Flip}(\mathbb{P}_d); \Phi_d?$ decides that only one do_a is true and that $pre_a \in v_\omega$. As the program $\mathsf{Flip}(\mathbb{P}_d); \Phi_d?$ is a subprogram of the program Dyn, we have $do_a \in v_{\omega+1}$ and $v_{\omega+1} \cap \mathbb{P}_d = \{do_a\}$. So, it satisfies the definition of $\mathcal{D}$-models (Definition 3.5) on D-function: $D(\omega) \in \{a \in \mathsf{Act}_0 \mid V(\omega) \models \mathtt{pre}(a)\}$. Due to the subprogram Update of the program Dyn, by the formula $\Phi_h \wedge \Phi_c$, we have $H(\omega) = \{e \in \mathsf{Evt}_0 \mid V(\omega) \models \mathtt{pre}(e)\}$, which is the definition of $\mathcal{D}$-models on H-function. Because only programs $do_a?; \mathsf{Effect}(a)$ and $h_e?; \mathsf{Effect}(e)$ can change the truth value of propositional variables in $\mathbb{P}$ and dynamic theory $\mathcal{D}$ is coherent, it satisfies the definition of $\mathcal{D}$-models (Definition 3.5) on valuations: for every time point $\omega \in \mathbb{N}_n$,

$$V(\omega{+}1) = \big(V(\omega) \cup \mathtt{eff}^{+}\big(H(\omega){\cup}\{D(\omega)\}\big)\big) \setminus \mathtt{eff}^{-}\big(H(\omega){\cup}\{D(\omega)\}\big).$$

Therefore, $\overline{\rho}$ is a bounded $\mathcal{D}$-model.

Next we will prove that $\overline{\rho} \Vdash_{\mathcal{D}} \Delta$. For every time point ω, $v_\omega \in ||\tau^{\mathsf{D}}_{\mathsf{S}}(\Delta, \omega)||$, so $\overline{\rho}$ satisfies all beliefs and (non-)occurrence of events in Δ which are the first three criteria in the definition of database satisfaction (Definition 3.7).

Next we will show that for all intentions $i \in \Delta$, $\overline{\rho} \Vdash_{\mathcal{D}} i$. For an intention $i = (t, \alpha, d) \in \Delta$, because $v_d \in ||\tau^{\mathsf{D}}_{\mathsf{S}}(\Delta, d)||$, we have $f_i \in v_d$. In the case that $\alpha \notin \mathsf{Act}_0$, the variable f_i only can be assigned to be true by the program $\mathsf{FinHInt}(i)$ where the prerequisite of the assignment is $b_i \wedge post_\alpha$. As the program $\mathsf{FinHInt}(i)$ only included in the programs $\mathsf{Finish}(\omega)$ such that $t < \omega \leq d$, we can conclude that f_i is assigned in some program $\mathsf{Finish}(d')$ such that $t < d' \leq d$. It entails that

$\mathtt{post}_\alpha \in v_{d'}$ and then $V(d') \models \mathtt{post}_\alpha$. For one of the prerequisites b_i, because it only can be assigned to be true by the program $\mathsf{BegInt}(i)$ and for every time point ω, the program $\mathsf{Finish}(\omega)$ is ahead of the program $\mathsf{Begin}(\omega)$, there exists t' such that $t \leq t' < d'$ and the assignment program $b_i \leftarrow \top$ happens. As the prerequisite of the assignment of b_i is pre_α, we have $pre_\alpha \in v_{t'}$ and then $V(t') \models \mathtt{pre}_\alpha$. Therefore, by the definition of intention satisfaction (Definition 3.6), we have $\overline{\rho} \Vdash_{\mathcal{D}} i$. In the case that $\alpha \in \mathsf{Act}_0$, the variable f_i only can be assigned to be true by the program $\mathsf{FinBInt}(i)$ where the prerequisite of the assignment is do_α. Because the program $\mathsf{FinBInt}(i)$ is only included in the programs $\mathsf{Finish}(\omega)$ such that $t < \omega \leq d$ and $f_i \in v_d$, we can conclude that f_i is assigned in some program $\mathsf{Finish}(d')$ such that $t < d' \leq d$. It entails that $do_\alpha \in v_{d'}$ and then $D(d'-1) = a$. Thus, by the definition of intention satisfaction (Definition 3.6), we have $\overline{\rho} \Vdash_{\mathcal{D}} i$.

Therefore, the bounded $\mathcal{D}$-model $\overline{\rho}$ constructed from the sequence of valuations $v_0, v_1, \ldots, v_{n+1}$ is a bounded model of Δ. As the dynamic theory $\mathcal{D}$ is coherent, the database Δ is satisfiable. $\square$

Here is a simple example:

Example 4.2. *Consider the database* $\Delta = \{(0,p),(1,e),(0,\alpha,3),(1,b,2)\}$. *We can get formula* $\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D},\Delta,0) =$

$$\begin{aligned} & \neg b_{(0,\alpha,3)} \wedge \neg b_{(1,b,2)} \wedge \neg f_{(0,\alpha,3)} \wedge \neg f_{(1,b,2)} \wedge p \\ \wedge \quad & \langle \pi_1 \rangle \Big(h_e \wedge \langle \pi_2 \rangle \big(f_{(1,b,2)} \wedge \langle \pi_3 \rangle (f_{(0,\alpha,3)} \wedge \langle \mathit{Dyn} \rangle \top) \big) \Big) \end{aligned}$$

where

$$\begin{aligned} \pi_1 &= \mathit{Begin}(0); \mathit{Dyn}; \mathit{Finish}(1); \mathit{Begin}(1) \\ &= \mathit{BegInt}(0,\alpha,3); \mathit{Dyn}; \mathit{FinHInt}(0,\alpha,3); \mathit{BegInt}(0,\alpha,3); \mathit{BegInt}(1,b,2) \\ \pi_2 &= \mathit{Dyn}; \mathit{Finish}(2); \mathit{Begin}(2) \\ &= \mathit{Dyn}; \mathit{FinHInt}(0,\alpha,3); \mathit{FinBInt}(1,b,2); \mathit{BegInt}(0,\alpha,3) \\ \pi_3 &= \mathit{Dyn}; \mathit{Finish}(3); \mathit{Begin}(3) \\ &= \mathit{Dyn}; \mathit{FinHInt}(0,\alpha,3) \end{aligned}$$

Next we translate the consequence problem in the database into the validity problem in DL-PA. Given a dynamic theory $\mathcal{D}$ and two databases Δ_1 and Δ_2, suppose $n = \mathbf{max}(\mathsf{end}(\Delta_1), \mathsf{end}(\Delta_2))$, we can define the translation for the consequence inductively:

$$\Gamma^{\mathsf{D}}_{\mathsf{C}}(\mathcal{D}, \Delta_1, \Delta_2, \omega) = \begin{cases} [\mathsf{Dyn}]\top, & \omega = n+1 \\ [\mathsf{Progress}'(\omega)]\big(\tau^{\mathsf{D}}_{\mathsf{C}}(\Delta_1, \Delta_2, \omega) \wedge \Gamma^{\mathsf{D}}_{\mathsf{C}}(\mathcal{D}, \Delta_1, \Delta_2, \omega{+}1)\big), & 1 \leq \omega \leq n \\ \mathrm{Init} \wedge \tau^{\mathsf{D}}_{\mathsf{C}}(\Delta_1, \Delta_2, 0) \wedge [\mathsf{Begin}'(0)]\Gamma^{\mathsf{D}}_{\mathsf{C}}(\mathcal{D}, \Delta_1, \Delta_2, 1), & \omega = 0 \end{cases}$$

where

$$\begin{aligned} \mathsf{Begin}'(\omega) &= \mathop{;}_{\substack{i \in \Delta_1 \cup \Delta_2 \\ t \leq \omega < d}} \mathsf{BegInt}(i) \\ \mathsf{Finish}'(\omega) &= \mathop{;}_{\substack{i \in \Delta_1 \cup \Delta_2 \\ \alpha \notin \mathsf{Act}_0 \\ t < \omega \leq d}} \mathsf{FinHInt}(i); \mathop{;}_{\substack{i^0 \in \Delta_1 \cup \Delta_2 \\ \alpha \in \mathsf{Act}_0 \\ t < \omega \leq d}} \mathsf{FinBInt}(i^0) \\ \mathsf{Progress}'(\omega) &= \mathsf{Dyn}; \mathsf{Finish}'(\omega); \mathsf{Begin}'(\omega) \\ \tau^{\mathsf{D}}_{\mathsf{C}}(\Delta_1, \Delta_2, \omega) &= \tau^{\mathsf{D}}_{\mathsf{S}}(\Delta_1, \omega) \to \tau^{\mathsf{D}}_{\mathsf{S}}(\Delta_2, \omega) \end{aligned}$$

While the translation for the satisfiability guarantees there exists a sequence of valuations satisfying the criterion of $\mathcal{D}$-models, the translation $\Gamma^{\mathsf{D}}_{\mathsf{C}}$ requires that for all such sequences of valuations, they satisfy Δ_1, entailing the they also satisfy Δ_2. The following theorem shows the correctness of the translation for the consequence problem.

Theorem 4.11. *Given a coherent dynamic theory $\mathcal{D}$, $\Delta_1 \models_{\mathcal{D}} \Delta_2$ iff the formula $\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D}, \Delta_1, 0) \to \Gamma^{\mathsf{D}}_{\mathsf{C}}(\mathcal{D}, \Delta_1, \Delta_2, 0)$ is* DL-PA *valid.*

Proof. "$\Rightarrow$:" Suppose $\Delta_1 \models_{\mathcal{D}} \Delta_2$ and we need to prove the translating formula is DL-PA valid. Let us consider two cases: Δ_1 is $\mathcal{D}$-satisfiable and Δ_1 is not.

For the case that Δ_1 is $\mathcal{D}$-satisfiable. By Theorem 4.10, the formula $\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D}, \Delta_1, 0)$ is DL-PA satisfiable. Note that the program $\mathsf{Progress}'$ differs from $\mathsf{Progress}$ in Begin'

and Finish' which does not influence propositional variables in $\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D}, \Delta_1, 0)$. Then, by the proof of Theorem 4.10, the program $\mathsf{Begin}'(0); \;;_{1\leq\omega\leq n} \mathsf{Progress}'(\omega); \mathsf{Dyn}$ actually generates sequences of DL-PA valuations $v_0, v_1, \ldots, v_{n+1}$ such that

(i) $(v_0, v_1) \in ||\mathsf{Begin}(0); \mathsf{Progress}(1)||$ and $v_0 \in ||\tau^{\mathsf{D}}_{\mathsf{S}}(\Delta, 0)||$;

(ii) for $k = 1, \ldots, n-1$, $(v_k, v_{k+1}) \in ||\mathsf{Progress}(k+1)||$ and $v_k \in ||\tau^{\mathsf{D}}_{\mathsf{S}}(\Delta, k)||$;

(iii) $(v_n, v_{n+1}) \in ||\mathsf{Dyn}||$.

Every such sequence of valuations corresponds to a bounded $\mathcal{D}$-model of Δ_1 and $v_k \models \tau^{\mathsf{D}}_{\mathsf{S}}(\Delta_1, k)$ for $0 \leq k \leq n+1$. By the assumption $\Delta_1 \models_{\mathcal{D}} \Delta_2$, those bounded $\mathcal{D}$-models also satisfy Δ_2 and then $v_k \models \tau^{\mathsf{D}}_{\mathsf{S}}(\Delta_2, k)$ for $0 \leq k \leq n+1$. Thus, we have $v_k \models \tau^{\mathsf{D}}_{\mathsf{S}}(\Delta_1, k) \wedge \tau^{\mathsf{D}}_{\mathsf{S}}(\Delta_2, k)$ for $0 \leq k \leq n+1$, entailing that the formula $\Gamma^{\mathsf{D}}_{\mathsf{C}}(\mathcal{D}, \Delta_1, \Delta_2, 0)$ holds.

For the case that Δ_1 is not $\mathcal{D}$-satisfiable, we have the DL-PA formula $\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D}, \Delta_1, 0)$ is false.

So, the translating formula $\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D}, \Delta_1, 0) \to \Gamma^{\mathsf{D}}_{\mathsf{C}}(\mathcal{D}, \Delta_1, \Delta_2, 0)$ is DL-PA valid.

"$\Leftarrow$:" Suppose the translated formula is DL-PA valid and we show that $\Delta_1 \models_{\mathcal{D}} \Delta_2$. Let us consider two cases: the formula $\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D}, \Delta_1, 0)$ is true or not.

For the case that $\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D}, \Delta_1, 0)$ is false, by Theorem 4.10, we have Δ_1 is $\mathcal{D}$-unsatisfiable, and in consequence $\Delta_1 \models_{\mathcal{D}} \Delta_2$.

For the case that $\Gamma^{\mathsf{D}}_{\mathsf{S}}(\mathcal{D}, \Delta_1, 0)$ is true, we have Δ_1 is $\mathcal{D}$-satisfiable and the formula $\Gamma^{\mathsf{D}}_{\mathsf{C}}(\mathcal{D}, \Delta_1, \Delta_2, 0)$ is true. Similarly, by the proof of Theorem 4.10, The valuation sequence of all bounded models of Δ_1 with a bound n can be introduced by the program $\mathsf{Begin}'(0); \;;_{1\leq\omega\leq n} \mathsf{Progress}'(\omega); \mathsf{Dyn}$ from the valuation v_0 such that $v_0 \models \mathsf{Init} \wedge \tau^{\mathsf{D}}_{\mathsf{S}}(\Delta_1, 0)$. Therefore, for a valuation sequence $V(0), V(1), \ldots, V(n), V(n+1)$ of a bounded model $\overline{\rho}$ of Δ_1, there exists a valuation sequence $v_0, v_1, \ldots, v_n, v_{n+1}$ where

(i) for every time point $0 \leq \omega \leq n+1$, $v_\omega \cap \mathbb{P} = V(\omega)$;

(ii) $(v_0, v_1) \in ||\mathsf{Begin}(0); \mathsf{Progress}(1)||$ and $v_0 \in ||\tau_{\mathsf{S}}^{\mathsf{D}}(\Delta, 0)||$;

(iii) for $k = 1, \ldots, n-1$, $(v_k, v_{k+1}) \in ||\mathsf{Progress}(k+1)||$ and $v_k \in ||\tau_{\mathsf{S}}^{\mathsf{D}}(\Delta, k)||$;

(iv) $(v_n, v_{n+1}) \in ||\mathsf{Dyn}||$.

As $\Gamma_{\mathsf{C}}^{\mathsf{D}}(\mathcal{D}, \Delta_1, \Delta_2, 0)$ is true, for every time point $0 \leq \omega \leq n+1$, we have $v_\omega \in ||\tau_{\mathsf{C}}^{\mathsf{D}}(\Delta_1, \Delta_2, \omega)||$, and in consequence $v_\omega \in ||\tau_{\mathsf{S}}^{\mathsf{D}}(\Delta_2, \omega)||$. So, the bounded model $\overline{\rho}$ is also a model of Δ_2, entailing $\Delta_1 \models_{\mathcal{D}} \Delta_2$. □

Note that the above translations are based on the coherence assumption of dynamic theories. Just as for the translation to the satisfiability and consequence problems in PLTL, the problems deciding refinement and instrumentality can also be embedded in DL-PA. The following theorem states the correctness of the translation for the deciding-refinement problem.

Theorem 4.12. *Given a coherent dynamic theory $\mathcal{D}$, $\Delta \models_{\mathcal{D}} i \lhd J$, iff the following hold:*

- *for all $j \in J$, every $\Gamma_{\mathsf{C}}^{\mathsf{D}}(\mathcal{D}, \Delta, \{j\}, 0)$ is not* DL-PA *valid*
- $\Gamma_{\mathsf{S}}^{\mathsf{D}}(\mathcal{D}, \Delta \cup J, 0)$ *is* DL-PA *satisfiable*
- $\Gamma_{\mathsf{C}}^{\mathsf{D}}(\mathcal{D}, (\Delta \cup J) \setminus \{i\}, \{i\}, 0)$ *is* DL-PA *valid*
- *for all $j \in J$, every $\Gamma_{\mathsf{C}}^{\mathsf{D}}(\mathcal{D}, (\Delta \cup J) \setminus \{i, j\}, \{i\}, 0)$ is not* DL-PA *valid*
- $\mathsf{end}(J) \leq \mathsf{end}(i)$

Proof. It is straightforward by the definition of intention refinement and the above theorems. □

Furthermore, the deciding-instrumentality problem can be embedded in the satisfiability and validity problems in DL-PA, as shown in the following theorem.

Theorem 4.13. *Given a coherent dynamic theory $\mathcal{D}$, $\Delta \models_{\mathcal{D}} J \gtrdot i$, iff the following hold:*

- $\Gamma_{\mathsf{S}}^{\mathsf{D}}(\mathcal{D}, \Delta \setminus J, 0) \rightarrow \Gamma_{\mathsf{C}}^{\mathsf{D}}(\mathcal{D}, \Delta \setminus J, \{i\}, 0)$ *is not* DL-PA *valid*
- $\Gamma_{\mathsf{S}}^{\mathsf{D}}(\mathcal{D}, (\Delta \setminus J) \cup \{j\}, 0) \rightarrow \bigwedge_{j \in J} \Gamma_{\mathsf{C}}^{\mathsf{D}}(\mathcal{D}, (\Delta \setminus J) \cup \{j\}, \{i\}, 0)$ *is* DL-PA *valid*
- $\mathsf{end}(J) \leq \mathsf{end}(i)$

Proof. It is straightforward by the definition of instrumentality and the above theorems. □

4.3 Summary

In this chapter, we translate the decision problems of satisfiability, consequence, refinement and instrumentality in the belief-intention database into the satisfiability and validity problems in PLTL, in the cases whether the given dynamic theory is coherent or not. The state of the art in the automated tools of PLTL contributes to develop a solver for refining high-level intentions.

Furthermore, we encode the satisfiability, consequence, deciding-refinement and deciding-instrumentality problems via DL-PA, under the coherence assumption of the dynamic theory. The reduction from DL-PA formulas to propositional formulas allows us to decide refinement between higher and lower intentions via invoking a SAT solver.[4]

[4]The research team I am currently involved is working on developing a theorem prover for DL-PA via translating into QBF formulas whose satisfiability and validity problems are **PSPACE**-complete.

Chapter 5

HTN Planning in PDL

In Chapter 2, we have shown the semantics of hierarchical task network (HTN) planning. Unfortunately, HTN only has an operational semantics and it lacks a more logical semantics. As we can see, decomposition is performed in an incremental way and no global perspective on the HTN specification is considered. In this chapter we examine restricted HTN planning in the framework of propositional dynamic logic (PDL) where actions are viewed as programs and decomposition methods are captured by the program inclusion operator. We restrict all tasks to be totally ordered by the constraints and that the "maintenance" state constraints are not considered. Kambhampati *et al.* [1998] advocate that high-level actions ("compound tasks") are supposed to have a primary effect which is nothing but the postcondition, in order to distinguish them with the effects of basic actions. For every high-level action, its postcondition is required to be satisfied in its each execution, while for every basic action, it is supposed that the environment beyond its effect should not change. Thus, we introduce pre- and postcondition into high-level actions in HTN planning. Under the dynamic framework, we first investigate the modularity of HTN planning domains. Based on the introduction of pre- and postcondition, we give a coherence condition for HTN planning domain, which requires that once an action is performed, its postcondition holds; once an action is refined, its refinement also satisfy its postcondition. Next, we propose the soundness postulate for actions: when the precondition of a high-level action holds then every refinement (decomposition) of it guarantees its postcondition. Furthermore, we also discuss the completeness of actions under the dynamic view: when the precondition of a high- level action is true then it can be accomplished, entailing that there is a way to refine it into an executable plan.

This chapter is organized as follows. Section 5.1 addresses the coherence issue of HTN domains. Section 5.2 introduces the language and semantics of PDL and briefly recall how classical planning can be defined in PDL. In Section 5.3 we introduce a PDL-based presentation of HTN planning domains. In Section 5.4 we recast the standard, operational definition of solutions of a HTN planning problem in PDL. In Section 5.5 we propose and discuss rationality principles for HTN planning domains: criteria of modularity and of soundness and completeness of action refinement. Section 5.6 discusses related work and summarizes this chapter.

5.1 Soundness and Completeness of Decomposition Methods

Compared with classical planning, hierarchical task network (HTN) planning captures domain-specific heuristics for search plans. HTN planning is strictly more expressive than classical planning [Erol et al., 1994a]. That is, solutions of HTN problems may be structured in a way that are more complex than solutions of classical planning problems [Höller et al., 2014]. It also means that it is more difficult to find a more "logical" semantics for HTN planning than for classical planning.

The semantics of HTN planning offered by [Erol et al., 1994a] is operational and parallels the planning algorithm UMCP proposed in [Erol et al., 1994b]. In such an operational semantics, by using the decomposition methods in HTN domains, the domain design is simplified via avoiding the need for the complete causal models which require the complete precondition and postcondition, and allow the domain designer to stipulate that certain actions will be performed "just because he says so.” In other words, the way to refine high-level actions extremely depends on the knowledge of the domain designer on the scenario. However, as mentioned in Chapter 1, in most real-world domains, it is difficult for a domain designer, as a human being, to consider all possible decomposition methods. Furthermore, an alternative question may arise: are these decomposition methods sound? Unfortunately, there is no a clear semantics for HTN decomposition methods to evaluate the coherence of domains.

Let us illustrate the coherence issue by an abstract example. Suppose the only decomposition method for high-level action α is $(\alpha, [\{(t : \beta)\}, (t, p)])$. The pair $[\{(t : \beta)\}, (t, p)]$ is a task network which only contains a task $(t : \beta)$ a constraint (t, p) stipulating that p should be true immediately after t. So the only way to perform α is by performing β, with a postcondition p. Suppose moreover that β is also a high-level action and that its only decomposition method is $(\beta, [\{(\gamma, t')\}, \emptyset])$. So the only way to perform β is to apply γ and now suppose γ has a postcondition of $\neg p$. No task involving α can ever be solved. We call such a HTN planning domain description *unsound*. It is reasonable to expect HTN planning domain descriptions not to contain unsound decomposition methods. This is a simple example, and more complex unsound decomposition methods can be designed. In order to evaluate the coherence of domains, we need to go beyond the operational semantics.

In HTN planning, high-level actions do not have preconditions and effects. This contrasts with basic actions, which are described by their preconditions and effects just as in classical planning. One may bring forward philosophical reasons against such a heterogeneous representation of knowledge. We do not enter that debate here, but rather observe that several authors gave more practical arguments for equipping high-level actions with preconditions and effects, leading to hybrid planning [Kambhampati et al., 1998] which is a combination of HTN planning and classical planning. Kambhampati *et al.* advocate high-level actions to have pre- and postcondition, just as for basic actions. The postcondition of high-level actions is called *primary effect* which holds after performing the high-level action and is an arbitrary formula while the postcondition of basic actions is in form of a conjunction of literals. The pre- and postcondition of actions can be captured by state constraints: (l, t) with meaning that the literal l holds immediately before the action t and (t, l) with meaning that l holds immediately after t. Whereas, a high-level action may have multiple ways to accomplish, it may have different after-state constraints. For example, the action "gotoMelbourne" has at least two decomposition methods: by train and by plane. The by-train decomposition method includes after-state constraints $(\mathsf{AtTrainStation}, t)$ and $(\mathsf{AtMelbourne}, t)$ while the by-plane decomposition method includes after-state constraint $(\mathsf{AtAirport}, t)$ and $(\mathsf{AtMelbourne}, t)$ where t is the last subtask in the two decomposition methods respectively. In this example, AtMelbourne is the primary effect of the high-level action gotoMelbourne. The postcondition of high-level actions indeed is considered to capture their intentional goals.

Indeed, it is not obvious to describe all effects of a high-level action α, one of the reasons being that these effects may in particular be conditional on the refinement of α to be chosen. For example, the primary effect of the high-level action of building a house is that I have a house. Further effects may obtain, depending on whether I build the house myself or hire a builder: I either have a bad back, or an empty bank account. These two effects are non-deterministic and do not dominate in all effects of the action of building a house. We therefore consider that high-level actions are not described by their effect but only by their postcondition, i.e., primary effects.

In this chapter we introduce the precondition and postcondition of high-level actions which are both in form of arbitrary formulas[1]. Based on the pre- and postcondition of actions, we illustrate the coherence condition of the HTN domains. Next we address the soundness postulate of high-level actions: when the precondition of the action α holds then all refinements of α should guarantee the postcondition of α. We also illustrate the completeness postulate of high-level actions: when the precondition of a high-level action is true then it can be accomplished, entailing that there is a way to refine it into an executable plan. Our completeness postulate

[1] In some literature, "effect" is particularly for basic actions which is usually in form of two sets of variables with meaning positive and negative effects.

is similar the "planner completeness" in [Kambhampati et al., 1998]: the planner is able to return every solution that can be generated by applying decomposition methods. The completeness postulate can be weakened by requiring refinability unless there is no executable plan achieving the postcondition of the high-level action. In this chapter we model HTN planning in the framework of propositional dynamic logic and examine the soundness and completeness under the dynamic framework.

5.2 Propositional Dynamic Logic PDL

In the present section we provide the necessary syntactic and semantic definitions for an extension of propositional dynamic logic [Harel, 1984, Harel et al., 2000] with intersection and inclusion of programs and with only boolean test programs. This allows us to define classical planning problems and their solutions in PDL.

5.2.1 Language

Let $\mathbb{P}$ be a finite set of propositional variables, with typical elements p, q,... Let `Act` be a finite set of actions, with typical elements α, β,... In examples we use capital letters for propositional variables (such as HasHouse) and small letters for actions (such as buildHouse).

Boolean formulas are defined as usual. The set of boolean formulas is noted $\mathscr{L}_{\mathbb{P}}$.

Programs are defined by the following grammar:

$$\pi ::= \alpha \mid \pi;\pi \mid \pi \sqcup \pi \mid \pi \sqcap \pi \mid \pi^* \mid \varphi_0?$$

where α ranges over `Act` and φ_0 over $\mathscr{L}_{\mathbb{P}}$. The programs $\pi;\pi'$, $\pi \sqcup \pi'$ and $\pi \sqcap \pi'$ are respectively the sequential, nondeterministic and parallel composition of the programs π and π'; π^* is bounded iteration of π and $\varphi_0?$ is the test of φ_0.

Then formulas are defined by:

$$\varphi ::= p \mid \bot \mid \varphi \to \varphi \mid \langle \pi \rangle \varphi \mid \pi \sqsubseteq \pi$$

The formula $\langle \pi \rangle \varphi$ reads "there is a possible execution of π after which φ is true". The formula $\pi' \sqsubseteq \pi$ reads "every execution of π' is also an execution of π", or "the effects of π' are implied by the effects of π".

Other connectives can be defined in the standard way as abbreviations, e.g. the formula $[\pi]\varphi$ abbreviates $\neg\langle \pi \rangle \neg\varphi$, and the program $\mathsf{while}\ \varphi_0\ \mathsf{do}\ \pi$ abbreviates $(\varphi_0?;\pi)^*;\neg\varphi_0?$.

The set of all PDL programs is noted Ξ. A *theory* is a set of formulas.

5.2.2 Semantics

A model is a triple $M = \langle W, R, V \rangle$ where W is a non-empty set of possible worlds, $R : \Xi \longrightarrow 2^{W \times W}$ associates an accessibility relation R_π to every program in Ξ, and $V : \mathbb{P} \longrightarrow 2^W$ associates a set $V(p) \subseteq W$ to every propositional variable p. The function R must satisfy some constraints:

$$\begin{aligned}
R_{\pi_1;\pi_2} &= R_{\pi_1} \circ R_{\pi_2} \\
R_{\pi_1 \sqcup \pi_2} &= R_{\pi_1} \cup R_{\pi_2} \\
R_{\pi_1 \sqcap \pi_2} &= R_{\pi_1} \cap R_{\pi_2} \\
R_{\pi^*} &= (R_\pi)^* \\
R_{\varphi?} &= \{\langle w, w \rangle \mid M, w \Vdash \varphi\}
\end{aligned}$$

The pair (M, w) where $w \in W$ is called *pointed model*.

Due to the last clause these constraints have to be defined by mutual recursion with the truth conditions for formulas:

$$
\begin{array}{lll}
M, w \Vdash p & \text{iff} & w \in V(p) \\
M, w \nVdash \bot & & \\
M, w \Vdash \varphi \to \varphi' & \text{iff} & M, w \nVdash \varphi \text{ or } M, w \Vdash \varphi' \\
M, w \Vdash \langle \pi \rangle \varphi & \text{iff} & M, v \Vdash \varphi \text{ for some } v \in R_\pi(w) \\
M, w \Vdash \pi \sqsubseteq \pi' & \text{iff} & R_\pi(w) \subseteq R_{\pi'}(w)
\end{array}
$$

where $R_\pi(w) = \{v \mid \langle w, v \rangle \in R_\pi\}$.

A set of formulas Γ is *true in a model* M, written $M \Vdash \Gamma$, if $M, w \Vdash \varphi$ for every $w \in W$ and $\varphi \in \Gamma$. A formula φ is a *consequence of* Γ, written $\Gamma \models \varphi$, if $M \Vdash \Gamma$ implies $M \Vdash \varphi$ for every model M. A PDL formula φ is *satisfiable* if it has a model and φ is *valid* if it is a consequence of $\emptyset$. The following proposition will be useful.

Proposition 5.1. *The following formulas are valid:*

$$
\begin{array}{rr}
\pi' \sqsubseteq \pi \to (\langle \pi' \rangle \varphi \to \langle \pi \rangle \varphi) & \pi' \sqsubseteq \pi \to (\pi' \sqcup \pi_2) \sqsubseteq (\pi \sqcup \pi_2) \\
(\varphi_0?; \pi) \sqsubseteq \pi' \leftrightarrow (\varphi_0 \to \pi \sqsubseteq \pi') & \pi' \sqsubseteq \pi \to (\pi_1 \sqcup \pi') \sqsubseteq (\pi_1 \sqcup \pi) \\
\pi' \sqsubseteq \pi \to (\pi'; \pi_2) \sqsubseteq (\pi; \pi_2) & \pi' \sqsubseteq \pi \to (\pi' \sqcap \pi_2) \sqsubseteq (\pi \sqcap \pi_2) \\
[\pi_1](\pi' \sqsubseteq \pi) \to (\pi_1; \pi') \sqsubseteq (\pi_1; \pi) & \pi' \sqsubseteq \pi \to (\pi_1 \sqcap \pi') \sqsubseteq (\pi_1 \sqcap \pi) \\
\pi' \sqsubseteq \pi \to \big(\langle \pi' \rangle \top \to \langle \pi' \sqcap \pi \rangle \top\big) &
\end{array}
$$

Proof. We only prove the first item. For every model M and every $w \in W$, if $M, w \Vdash \pi \sqsubseteq \pi'$, then $R_\pi(w) \subseteq R_{\pi'}(w)$. Further, if $M, w \Vdash [\pi']\varphi$ then $M, v \Vdash \varphi$ for every $v \in R_{\pi'}(w)$. So $M, w \Vdash [\pi]\varphi$ due to $R_\pi(w) \subseteq R_{\pi'}(w)$. $\square$

5.2.3 Classical Planning in PDL

Dynamic logic constructors have been widely used for reasoning about actions, starting from [de Giacomo and Lenzerini, 1995]. Now we show how to model propositional classical planning in PDL.

Let us recall the notions of classical planning which are shown in Section 2.2. A propositional planning domain is a tuple $\mathfrak{D} = (O, \gamma)$ where O is a set of basic actions and γ is a state-transition function on O. A propositional classical planning problem is a tuple $\mathcal{P} = \langle \mathfrak{D}, s_I, G \rangle$ where s_I is the initial state and G is a conflict-free conjunction of literals. The solution of a classical planning problem $\mathcal{P}$ is a sequence of actions in O which is executable in the initial state s_I.

Just as for the belief-intention databases in Chapter 3, we rewrite the domain of classical planing as a dynamic theory. In propositional classical planning, only ground basic actions are considered and we use $\mathtt{Act}_0$ to denote the set of basic actions, which is the name of actions in O. The behavior of basic actions is described by a dynamic theory $\mathcal{D} = \langle \mathtt{pre}, \mathtt{post} \rangle$: the precondition and postcondition of basic actions are described by mappings $\mathtt{pre}, \mathtt{post} : \mathtt{Act}_0 \longrightarrow \mathscr{L}_{\mathbb{P}}$ associating to each action a boolean formula, and Formally, the classical planning domain $\mathfrak{D} = (O, \gamma)$ is written as the dynamic theory $\mathcal{D}_{\mathsf{class}} = \langle \mathtt{pre}, \mathtt{post} \rangle$ such that:
for every $(\alpha, \mathtt{pre}(o), \mathtt{eff}^+(\alpha), \mathtt{eff}^-(\alpha)) \in O$, $\mathtt{pre}(\alpha) = \bigwedge_{p \in \mathtt{pre}(\alpha)}$ and $\mathtt{post}(\alpha) = \big(\bigwedge_{p \in \mathtt{eff}^+(\alpha)} p\big) \wedge \big(\bigwedge_{p \in \mathtt{eff}^-(\alpha)} \neg p\big)$.

The intended behavior of actions can be captured in PDL by the following theories:

$$\begin{aligned}
\mathtt{Fml}(\mathtt{pre}) &= \{\mathtt{pre}(\alpha) \leftrightarrow \langle \alpha \rangle \top \mid \alpha \in \mathtt{Act}_0\} \\
\mathtt{Fml}(\mathtt{post}) &= \{[\alpha]\mathtt{post}(\alpha) \mid \alpha \in \mathtt{Act}_0\} \cup \\
&\quad \{p \to [\alpha]p \mid p \notin \mathtt{eff}^-(\alpha)\} \cup \\
&\quad \{\neg p \to [\alpha]\neg p \mid p \notin \mathtt{eff}^+(\alpha)\}
\end{aligned}$$

The formulas in $\mathtt{Fml}(\mathtt{pre})$ say that each action α is executable exactly when its precondition $\mathtt{pre}(\alpha)$ is true. The formulas in $\mathtt{Fml}(\mathtt{post})$ say that the effects of each action α obtain, that the variables that are not negatively impacted by α remain true and that the variables that are not positively impacted by α remain false. Note that $\mathtt{Fml}(\mathtt{post})$ is finite because $\mathbb{P}$ is so.

For a classical planning domain $\mathfrak{D}$, we define its theory by

$$\mathtt{Fml}(\mathcal{D}_{\mathsf{class}}) = \mathtt{Fml}(\mathtt{pre}) \cup \mathtt{Fml}(\mathtt{post})$$

When $M \Vdash \texttt{Fml}(\mathcal{D}_{\mathsf{class}})$ then we say that M is a *model of* $\mathcal{D}_{\mathsf{class}}$.

For a classical planning problem $\mathcal{P} = \langle \mathfrak{D}, s_I, \mathsf{Goal} \rangle$, we define a formula for the initial state s_I as $\mathsf{Init} = \bigwedge_{p \in s_I} p$ and suppose $\mathcal{D}_{\mathsf{class}}$ is the dynamic theory for the classical planning domain.[2] A *solution* to $\mathcal{P}$ is a sequence $\alpha_1; \cdots; \alpha_n$ of actions such that $\texttt{Fml}(\mathcal{D}_{\mathsf{class}}) \models \mathsf{Init} \to \langle \alpha_1; \cdots; \alpha_n \rangle \mathsf{Goal}$. The following proposition rephrases solution in terms of more general consequence:

Proposition 5.2. *There exists a solution to* $\mathcal{P}$ *if and only if*

$$\texttt{Fml}(\mathcal{D}_{\mathsf{class}}) \models \mathsf{Init} \to \langle (\bigsqcup_{\alpha \in \texttt{Act}} \alpha)^* \rangle \mathsf{Goal}.$$

5.3 HTN Planning Domains in PDL

In Chapter 2, we have clarified the terminologies about actions in classical planning and HTN planning. In this chapter, we stipulate that the ground action type in HTN planning, for "compound task" is called *high-level action* and for "primitive task" is called *basic actions*. Here we omit the terminology for the action token. In propositional HTN planning, it is presupposed that the set of actions $\texttt{Act}$ is partitioned into two sets:the set of basic actions $\texttt{Act}_0$ and the set of high-level actions $\texttt{Act} \setminus \texttt{Act}_0$. We use α, β for arbitrary elements of $\texttt{Act}$ as before and use $a, b, \ldots$ for typical elements of $\texttt{Act}_0$. A *plan* is a sequence of basic actions and a *primitive program* is a program which does not include any high-level actions. The formula consists of primitive programs and boolean formulas is called *primitive formula*.

As we have said in Section 5.1, standard presentations of HTN planning contain explicit descriptions of preconditions and effects only for the basic actions. We here suppose that *all* actions have preconditions and effects. While we require the effects to be STRIPS-like for the basic actions, we allow high-level actions to have

[2]It is usually supposed that Init completely describes a classical valuation and Goal is a conjunction of literals, but we allow Init to be incomplete and Goal to be any boolean formula here.

any boolean formula as an effect. For example, the effect of the high-level action of travelling abroad could be a formula such as $\neg\mathsf{InFrance}\wedge(\mathsf{InGermany}\vee\mathsf{InChina}\vee\ldots)$.

5.3.1 HTN Planning Domains with Pre- and Postcondition

Recall that in the propositional HTN planning domain $\mathfrak{D} = (\mathcal{O}, \mathcal{M})$, the set $\mathcal{O}$ is the set of operators which are basic action types and the set $\mathcal{M}$ consists of decomposition methods (α, d) which means that the high-level action α can be decomposed into the task network d. Similar for the domain of classical planning, we take all the action names in $\mathcal{O}$ into a set of basic actions $\mathtt{Act}_0$. Here we restrict the HTN planning domain to only include task networks in which all tasks in d are totally ordered by the constraints and in which the "maintenance" state constraints (t, l, t') are not considered. Formally, for the subtask network $d = [\{(t_1 : \alpha_1), \ldots, (t_n : \alpha_n)\}, \varphi]$, we define a sequential program $\pi^d = \langle \varphi^{i_1}?; \alpha_{i_1}; \varphi_{i_1}?; \varphi^{i_2}?; \alpha_{i_2}; \varphi_{i_2}?; \ldots; \varphi^{i_n}?; \alpha_{i_n}; \varphi_{i_n}? \rangle$ where

- the total order $\langle t_{i_1}, t_{i_2}, \ldots, t_{i_n} \rangle$ satisfies the constraint formula φ by the definition of satisfaction of constraints (Definition 2.1)
- $\varphi^{i_k} = \bigwedge_{(\psi, t_{i_k}) \in \varphi} \psi$
- $\varphi_{i_k} = \bigwedge_{(t_{i_k}, \psi) \in \varphi} \psi$

Intuitively, φ^{i_k} captures all the state constraints (ψ, t_{i_k}) which hold immediately before t_{i_k} and φ_{i_k} captures all the state constraints (t_{i_k}, ψ) which hold immediately after t_{i_k}. Note that the formula ψ is restricted as a literal in standard HTN planning but here it is allowed to be any boolean formula.

Remark 5.1. To relax the restriction of totally ordered tasks, we can enumerate all possible sequences on tasks which satisfy the partial orders among tasks and then use the non-deterministic operator to combine these sequences. Formally, suppose tasks in a task network $d = [\{(t_1 : \alpha_1), \ldots, (t_n : \alpha_n)\}, \varphi]$ are not totally ordered.

Then let $d_1, \ldots, d_l$ be all sequences of all tasks in d which satisfy the constraints in d. The PDL program of task network d is represented by $d_1 \sqcup \ldots \sqcup d_l$, which means that only one order of tasks needs to be chosen to do. However, as the number of the possible ordering, in the worst case, is factorial in the number of tasks, the length[3] of the PDL programs is also factorial. For the purpose of simplicity, in this chapter we suppose all tasks are totally ordered.

An HTN planning domain $\mathfrak{D} = (\mathcal{O}, \mathcal{M})$ is rewritten as a dynamic theory $\mathcal{D}_{\mathsf{htn}} = \langle \mathtt{pre}, \mathtt{post}, \mathrm{ref} \rangle$ where

$$\mathtt{pre} : \mathtt{Act} \longrightarrow \mathscr{L}_{\mathbb{P}}$$

$$\mathtt{post} : \mathtt{Act} \longrightarrow \mathscr{L}_{\mathbb{P}}$$

$$\mathrm{ref} : \mathtt{Act} \longrightarrow 2^{\Xi}$$

and where the effect function $\mathtt{post}$ is STRIPS-like for basic actions: for every $a \in \mathtt{Act}_0$, $\mathtt{post}(a)$ is of the form $\left(\bigwedge_{p \in \mathtt{eff}^+(a)} p\right) \wedge \left(\bigwedge_{p \in \mathtt{eff}^-(a)} \neg p\right)$ for some sets $\mathtt{eff}^+(a)$ and $\mathtt{eff}^-(a)$ such that $\mathtt{eff}^+(a) \cap \mathtt{eff}^-(a) = \emptyset$ and the refinement function ref associates to each high-level action α the set of sequential programs π^d corresponding to the subtask network d for every $(\alpha, d) \in \mathcal{M}$. We also suppose that there is no basic action a such that $\mathtt{post}(a)$ is equivalent to $\bot$ and the pre- and postcondition of high-level actions are arbitrary boolean formulas. The refinement function ref must be such that basic actions cannot be refined: $\mathrm{ref}(a) = \emptyset$ for every $a \in \mathtt{Act}_0$.

For the unsound abstract example in Section 5.1, we have $\mathrm{ref}(\alpha) = \{(\beta; p?)\}$ and $\mathrm{ref}(\beta) = \{\gamma\}$. Let us formalize this example and put the test program $p?$ into the postcondition of α. Then we get the following dynamic theory.

Example 5.1. *Suppose the high-level action α has postcondition p and its subtask β has a valid postcondition for the purpose of simplicity. Suppose furthermore that β has only one refinement γ which is a basic action with an effect $\neg p$. Let us*

[3] The length of PDL programs is defined as the length of DL-PA programs, i.e., the number of symbols used to write down the program without "⟨","⟩" and parentheses.

TABLE 5.1: An HTN Planning Domain $\mathcal{D}^{\mathsf{AB}}_{\mathsf{htn}}$: Traveling from A to B

$$
\begin{array}{rlrlrl}
\mathtt{pre}(\mathsf{goAB}) &= \mathsf{AtA} & \mathtt{post}(\mathsf{goAB}) &= \mathsf{AtB} & \mathrm{ref}(\mathsf{goAB}) &= \{\mathsf{TaxiAB}, \mathsf{walkAB}\} \\
\mathtt{pre}(\mathsf{TaxiAB}) &= \mathsf{AtA} & \mathtt{post}(\mathsf{TaxiAB}) &= \mathsf{AtB} & \mathrm{ref}(\mathsf{TaxiAB}) &= \{(\mathsf{rideAB}; \mathsf{pay})\} \\
\mathtt{pre}(\mathsf{walkAB}) &= \mathsf{AtA} & \mathtt{post}(\mathsf{walkAB}) &= \mathsf{AtB}\land\neg\mathsf{AtA} & \mathrm{ref}(\mathsf{walkAB}) &= \emptyset \\
\mathtt{pre}(\mathsf{rideAB}) &= \mathsf{AtA} & \mathtt{post}(\mathsf{rideAB}) &= \mathsf{AtB}\land\neg\mathsf{AtA} & \mathrm{ref}(\mathsf{rideAB}) &= \emptyset \\
\mathtt{pre}(\mathsf{pay}) &= \mathsf{Money} & \mathtt{post}(\mathsf{pay}) &= \neg\mathsf{Money} & \mathrm{ref}(\mathsf{pay}) &= \emptyset
\end{array}
$$

suppose that there are no other postcondition.

$$
\begin{array}{lll}
\mathtt{pre}(\alpha) = \varphi_\alpha & \mathtt{post}(\alpha) = p & \mathrm{ref}(\alpha) = \{\beta\} \\
\mathtt{pre}(\beta) = \varphi_\beta & \mathtt{post}(\beta) = \top & \mathrm{ref}(\beta) = \{\gamma\} \\
\mathtt{pre}(\gamma) = \top & \mathtt{post}(\beta) = \neg p & \mathrm{ref}(\gamma) = \emptyset
\end{array}
$$

Next let us consider a typical example that is inspired from [Erol et al., 1996].

Example 5.2. *Suppose an agent intends to travel from* A *to* B *and the HTN planning domain* $\mathcal{D}^{\mathsf{AB}}_{\mathsf{htn}}$ *is described by Table 5.1. The set of basic actions is* $\mathtt{Act}_0 = \{\mathsf{pay}, \mathsf{rideAB}, \mathsf{walkAB}\}$*. The refinement ways of travelling from* A *to* B *are walking and going by taxi which is refined into riding the taxi and then paying. Note that* $\mathtt{post}(\mathsf{goAB})$ *does not mention the possible effect* $\neg\mathsf{Money}$*, which is only produced when* goAB *is refined to* TaxiAB*.*

5.3.2 Expressing HTN Planning Domains in PDL

Just as for classical planning, the behavior of actions is captured in PDL by the following theories:

$$
\begin{aligned}
\mathtt{Fml}(\mathtt{pre}) &= \{\langle\alpha\rangle\top \leftrightarrow \mathtt{pre}(\alpha) \mid \alpha \in \mathtt{Act}\} \\
\mathtt{Fml}(\mathtt{post}) &= \{[\alpha]\mathtt{post}(\alpha) \mid \alpha \in \mathtt{Act}\} \cup \\
&\quad \{p \to [a]p \mid a \in \mathtt{Act}_0 \text{ and } p \notin \mathtt{eff}^-(a)\} \cup \\
&\quad \{\neg p \to [a]\neg p \mid a \in \mathtt{Act}_0 \text{ and } p \notin \mathtt{eff}^+(a)\} \\
\mathtt{Fml}(\mathrm{ref}) &= \{\langle\alpha\rangle\top \to \pi\sqsubseteq\alpha \mid \alpha \in \mathtt{Act}, \pi \in \mathrm{ref}(\alpha)\}
\end{aligned}
$$

Note that all these sets are finite because $\texttt{Act}$ and $\mathbb{P}$ are so. So basic actions behave like STRIPS actions, while the description of high-level actions are less constrained, leaving room for side effects and conditional effects. We define the theory of an HTN planning domain by:

$$\texttt{Fml}(\mathcal{D}_{\mathsf{htn}}) = \texttt{Fml}(\texttt{pre}) \cup \texttt{Fml}(\texttt{post}) \cup \texttt{Fml}(\text{ref}).$$

When $M \Vdash \texttt{Fml}(\mathcal{D}_{\mathsf{htn}})$ then we say that M is a *model of* $\mathcal{D}_{\mathsf{htn}}$.

We now list some properties of HTN theories that will be useful in the sequel.

Proposition 5.3. *Let* $\alpha \in \texttt{Act}$. *Then*

$$\texttt{Fml}(\texttt{post}) \models \pi \sqsubseteq \alpha \to \big[\pi\big]\texttt{post}(\alpha).$$

Proof. Suppose M is a model of $\texttt{Fml}(\texttt{post})$ and $M, w \Vdash \pi \sqsubseteq \alpha$, i.e., $R_\pi(w) \subseteq R_\alpha(w)$. As $M, w \Vdash [\alpha]\texttt{post}(\alpha)$ we have $M, v \Vdash \texttt{post}(\alpha)$ for every $v \in R_\pi(w)$. So $M, w \Vdash [\pi]\texttt{post}(\alpha)$. □

The next result says that basic actions behave in a deterministic way.

Proposition 5.4. *Let* $a \in \texttt{Act}_0$ *be basic actions and let* $\varphi_0 \in \mathscr{L}_{\mathbb{P}}$ *be a boolean formula. Then*

$$\texttt{Fml}(\texttt{post}) \models \langle a \rangle \varphi_0 \to [a]\varphi_0.$$

Proof. Suppose M is a model of $\texttt{Fml}(\texttt{post})$. Then, for any w in M:

$$\begin{aligned} M, w \Vdash [a]\Big(\big(\bigwedge_{p \in \texttt{eff}^+(a)} p\big) \wedge \big(\bigwedge_{p \in \texttt{eff}^-(a)} \neg p\big)\Big) \wedge \\ \big(\bigwedge_{p \notin \texttt{eff}^-(a)} p \to [a]p\big) \wedge \big(\bigwedge_{p \notin \texttt{eff}^+(a)} \neg p \to [a]\neg p\big) \end{aligned}$$

It is easy to check that the valuation associated to each possible world $v \in R_a(w)$ is unique: if $p \in \texttt{eff}^+(a) \cup \texttt{eff}^-(a)$ then the truth value of p is defined with respect to the effect functions; if $p \notin \texttt{eff}^+(a) \cup \texttt{eff}^-(a)$ then the truth value of p in v is determined by its truth value at w. So if $M, v \Vdash \varphi_0$ holds for some $v \in R_a(w)$ then $M, v \Vdash \varphi_0$ holds for every $v \in R_a(w)$. $\square$

Observe that due to Proposition 5.1,

$$\texttt{Fml(post)} \models (a_1;\ldots;a_n) \sqsubseteq \pi \to \big(\big\langle a_1;\ldots;a_n\big\rangle \top \to \big\langle (a_1;\ldots;a_n) \sqcap \pi\big\rangle \top\big).$$

always holds. The converse implication fails to hold:

$$\texttt{Fml(post)} \not\models \big\langle (a_1;\ldots;a_n) \sqcap \pi\big\rangle \top \to (a_1;\ldots;a_n) \sqsubseteq \pi$$

because a possible world may have more than one $R_{a_1;\cdots a_n}$-successor.

5.4 HTN Planning Problems and Their Solutions in PDL

In this section we encode HTN planning problems and their solutions in PDL.

5.4.1 HTN Planning Problems

For an HTN planning problem $\mathcal{P}_{\mathsf{htn}} = \langle \mathfrak{D}, s_I, d\rangle$ where the tasks in the initial task network d are totally ordered by the constraints in d, we define a formula for the initial state s_I as $\mathsf{Init} = \bigwedge_{p \in s_I} p$ and suppose $\mathcal{D}_{\mathsf{htn}}$ is the dynamic theory for the HTN planning domain.[4] Then we rewrite the HTN planning problem as $\mathcal{P}_{\mathsf{htn}} = \langle \mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi^d\rangle$.

An HTN planning problem of Example 5.2 is a triple $\langle \mathcal{D}^{\mathsf{AB}}_{\mathsf{htn}}, \mathsf{Init}, \mathsf{goAB}\rangle$ where $\mathsf{Init} = \mathsf{AtA} \wedge \neg\mathsf{AtB} \wedge \mathsf{Money}$.

[4] It is usually supposed that Init is a complete description of a state, but we do not need that requirement here.

5.4.2 Solutions of HTN Planning Problems

The definitions of solutions of HTN planning problems that can be found in the literature are mainly in terms of fixed-points [Erol et al., 1994a]. These solutions are basically obtained by refining the 'goal program' step-by-step until a primitive program is obtained. We now recast this definition in PDL, in three steps.

First, for a primitive program π_0 which does not include any high-level actions, we can define its completion in PDL as follows:

$$\mathsf{comp}(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi_0) = \{a_1;\ldots;a_n \mid \mathtt{Fml}(\mathtt{pre}) \cup \mathtt{Fml}(\mathtt{post}) \models \mathsf{Init} \to \langle (a_1;\ldots;a_n) \sqcap \pi_0 \rangle \top\}.$$

So the completion of a primitive program is the set of all plans that are executable in the initial state and compatible with the primitive program.

Second, the reduction of a program π in an HTN planning domain $\mathcal{D}_{\mathsf{htn}}$ is defined by:

$$\mathsf{red}(\mathcal{D}_{\mathsf{htn}}, \pi) = \{\pi^{\alpha}_{\mathtt{pre}(\alpha)?;\pi'} \mid \alpha \text{ occurs in } \pi \text{ and } \pi' \in \mathrm{ref}(\alpha)\}$$

where $\pi^{\alpha}_{\mathtt{pre}(\alpha)?;\pi'}$ is obtained from π by replacing some occurrence of α in π by $\mathtt{pre}(\alpha)?;\pi'$. Observe that when π is a primitive program then $\mathsf{red}(\mathcal{D}_{\mathsf{htn}}, \pi) = \emptyset$. The function $\mathsf{red}(\mathcal{D}_{\mathsf{htn}}, \pi)$ behaves in a way such that, under $\mathcal{D}_{\mathsf{htn}}$, only programs included in π are produced. This is stated as the next result.

Proposition 5.5. *If* $\pi' \in \mathsf{red}(\mathcal{D}_{\mathsf{htn}}, \pi)$ *then* $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \pi' \sqsubseteq \pi$.

Proof. Suppose $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}})$ is satisfiable. The proof is by induction on the structure of π and uses Proposition 5.1. For the base case π is one action α and $\alpha' \in \mathrm{ref}(\alpha)$ and then $\pi' = \mathtt{pre}(\alpha)?;\alpha'$. As $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \langle\alpha\rangle\top \to \alpha' \sqsubseteq \alpha$ and $\mathtt{pre}(\alpha) \leftrightarrow \langle\alpha\rangle\top$, then $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \mathtt{pre}(\alpha) \to \alpha' \sqsubseteq \alpha$. By Proposition 5.1, $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \mathtt{pre}(\alpha)?;\alpha' \sqsubseteq \alpha$. So $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \pi' \sqsubseteq \pi$.

For the induction step, we only prove the case $\pi = \pi_1;\pi_2$ such that $\pi_1';\pi_2 \in \mathsf{red}(\mathcal{D}_{\mathsf{htn}}, \pi)$. Suppose the induction hypothesis holds that if $\pi_1'' \in \mathsf{red}(\mathcal{D}_{\mathsf{htn}}, \pi_1)$ then

$\texttt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \pi_1'' \sqsubseteq \pi_1$. According to the definition of red, we have $\pi_1' \in \mathsf{red}(\mathcal{D}_{\mathsf{htn}}, \pi_1)$. Then $\texttt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \pi_1' \sqsubseteq \pi_1$. Then by Proposition 5.1, $\texttt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \pi_1'; \pi_2 \sqsubseteq \pi_1; \pi_2$. $\square$

Finally, the *solutions* of an HTN planning problem are primitive plans that defined recursively as follows:

$$\mathsf{sol}^1(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi) = \begin{cases} \mathsf{comp}(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi) & \text{if } \pi \text{ is primitive} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\mathsf{sol}^{k+1}(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi) = \mathsf{sol}^k(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi) \cup \bigcup_{\pi' \in \mathsf{red}(\mathcal{D}_{\mathsf{htn}}, \pi)} \mathsf{sol}^k(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi')$$

$$\mathsf{sol}(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi) = \bigcup_k \mathsf{sol}^k(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi)$$

Now we connect solutions of HTN planning problems and consequence in PDL:

Theorem 5.6. *Let $\mathcal{P}_{\mathsf{htn}} = \langle \mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi \rangle$ be an HTN planning problem. Then*

$$a_1; \ldots ; a_n \in \mathsf{sol}(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi) \quad \textit{implies} \quad \texttt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \mathsf{Init} \to \langle (a_1; \ldots ; a_n) \sqcap \pi \rangle \top.$$

Proof. Suppose $\texttt{Fml}(\mathcal{D}_{\mathsf{htn}})$ is satisfiable. The proof is by induction on the number of iterations that are required to obtain the solution $a_1; \ldots ; a_n$. For the base case $k = 1$, when $a_1; \ldots ; a_n \in \mathsf{sol}^1(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi)$ then π must be a primitive plan. It follows by the definition of comp that $\texttt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \mathsf{Init} \to \langle (a_1; \ldots ; a_n) \sqcap \pi \rangle \top$.

For the induction step, suppose the induction hypothesis holds up to k steps, i.e., $a_1; \ldots ; a_n \in \mathsf{sol}^k(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi)$ implies

$$\texttt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \mathsf{Init} \to \langle (a_1; \ldots ; a_n) \sqcap \pi \rangle \top.$$

Suppose $a_1; \ldots ; a_n$ is in $\mathsf{sol}^{k+1}(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi)$, but not in $\mathsf{sol}^k(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi)$. Then by definition of sol^k there exists a $\pi' \in \mathsf{red}(\mathcal{D}_{\mathsf{htn}}, \pi)$ such that $a_1; \ldots ; a_n$ is in $\mathsf{sol}^k(\mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi')$. By induction hypothesis we have

$$\texttt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \mathsf{Init} \to \langle (a_1; \ldots ; a_n) \sqcap \pi' \rangle \top.$$

As $\pi' \in \mathsf{red}(\mathcal{D}_{\mathsf{htn}}, \pi)$, by Proposition 5.5 we have $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \pi' \sqsubseteq \pi$. Therefore by Proposition 5.1 we have

$$\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \mathsf{Init} \to \langle (a_1;\ldots;a_n) \sqcap \pi \rangle \top.$$

$\square$

The converse of Theorem 5.6 and its corollary do not hold because every plan has to be able to be generated by reductions. To see this consider Example 5.2, where $\mathtt{Fml}(\mathcal{D}^{\mathsf{AB}}_{\mathsf{htn}}) \models \mathsf{AtA} \to \langle \mathsf{rideAB} \sqcap \mathsf{goAB} \rangle \top$ but $\mathsf{rideAB} \notin \mathsf{sol}(\mathcal{D}^{\mathsf{AB}}_{\mathsf{htn}}, \mathsf{AtA}, \mathsf{goAB})$. That is, the plan rideAB of just taking the taxi without paying also achieves the primary effect of goAB, but it cannot be obtained by refinements.

The following proposition bridges PDL and the plan-existence problem for HTN planning.

Proposition 5.7. *Let $\mathcal{P}_{\mathsf{htn}} = \langle \mathcal{D}_{\mathsf{htn}}, \mathsf{Init}, \pi \rangle$ be an HTN planning problem. If $\mathcal{P}_{\mathsf{htn}}$ has a solution then*

$$\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \mathsf{Init} \to \langle (\bigsqcup \mathtt{Act}_0)^* \sqcap \pi \rangle \top.$$

Proof. By the valid formula $\langle a_1;\ldots;a_n \rangle \varphi \to \langle (\bigsqcup \mathtt{Act}_0)^* \rangle \varphi$. $\square$

5.5 Rationality Postulates for HTN Planning

We now formulate and discuss several postulates that, we claim, reasonable HTN planning domains should satisfy.

5.5.1 Modularity Postulate of HTN Domains

Intuitively, planning domains should satisfy several properties that can be related to the concept of modularity. Such principles were studied in the reasoning about actions literature [Herzig et al., 2006, Herzig and Varzinczak, 2007, Varzinczak, 2010]. One of these principles says that pre contains all information about action executability. (This hypothesis of complete information is also made by Reiter w.r.t. his Poss predicate specifying action preconditions [Reiter, 2001]). Our Explicit Executability Constraint (**EE**) states this principle as follows:

$$\text{If } \mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \langle\alpha\rangle\top \leftrightarrow \varphi \text{ then } \mathtt{Fml}(\mathtt{pre}) \models \langle\alpha\rangle\top \leftrightarrow \varphi. \tag{EE}$$

Let us use the following example to illustrate this constraint.

Example 5.3. *Consider the following HTN planning domain $\mathcal{D}_{\mathsf{htn}}$:*

$$\begin{aligned}
\mathtt{Fml}(\mathtt{pre}) &= \{\langle\alpha\rangle\top \leftrightarrow \varphi_\alpha,\ \langle\beta\rangle\top \leftrightarrow \varphi_\beta\}\\
\mathtt{Fml}(\mathtt{post}) &= \{[\alpha]p,\ [\beta]\neg p\}\\
\mathtt{Fml}(\mathrm{ref}) &= \{\langle\alpha\rangle\top \rightarrow \beta\sqsubseteq\alpha,\ \langle\beta\rangle\top \rightarrow \pi_\beta\sqsubseteq\beta\}
\end{aligned}$$

*Suppose M be a model of $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}})$. Suppose M contains a possible world w such that $M, w \Vdash \varphi_\alpha$. Then $R_\alpha(w)$ is non-empty due to formula $\langle\alpha\rangle\top \leftrightarrow \varphi_\alpha$ in $\mathtt{Fml}(\mathtt{pre})$. Due to formula $\langle\alpha\rangle\top \rightarrow \beta\sqsubseteq\alpha$ in $\mathtt{Fml}(\mathrm{ref})$ we moreover have $M, w \Vdash \beta\sqsubseteq\alpha$, and therefore $R_\alpha(w) \subseteq R_\beta(w)$. However, it entails that for every $w' \in R_\alpha(w)$, $M, w' \Vdash p \wedge \neg p$. Thus, such a world w cannot exist. It follows that for every model M such that $M \Vdash \mathtt{Fml}(\mathcal{D}_{\mathsf{htn}})$ we have $M \Vdash \neg\varphi_\alpha$, that is, $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \langle\alpha\rangle\top \leftrightarrow \bot$. So $\mathcal{D}_{\mathsf{htn}}$ violates Constraint (**EE**) as we have $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \langle\alpha\rangle\top \leftrightarrow \bot$ but we don't have $\mathtt{Fml}(\mathtt{pre}) \models \langle\alpha\rangle\top \leftrightarrow \bot$.*

A second principle that should hold is that information about refinement should not be relevant for the status of primitive formulas that do not include high-level actions. Primitive Modularity Constraint (**PM**) states this principle as follows: for every primitive formula φ_0,

$$\text{if } \mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \varphi_0 \text{ then } \mathtt{Fml}(\mathtt{pre}) \cup \mathtt{Fml}(\mathtt{post}) \models \varphi_0. \tag{PM}$$

The following example shows that Constraint (**PM**) is not always satisfied.

Example 5.4. *Let* $\mathtt{Act}_0 = \{a\}$ *and let* $\mathtt{Act} = \mathtt{Act}_0 \cup \{\alpha, \beta\}$. *Let* $\mathcal{D}_{\mathsf{htn}}$ *be a planning domain as captured by the following formulas:*

$$\mathtt{Fml}(\mathtt{pre}) = \{\langle\alpha\rangle\top \leftrightarrow \top,\ \langle\beta\rangle\top \leftrightarrow \top,\ \langle a\rangle\top \leftrightarrow \top\}$$
$$\mathtt{Fml}(\mathtt{post}) = \{[a]\top,\ [\alpha]q,\ [\beta]\neg q\} \cup \{p \to [a]p \mid p \in \mathbb{P}\} \cup \{\neg p \to [a]\neg p \mid p \in \mathbb{P}\}$$
$$\mathtt{Fml}(\mathrm{ref}) = \{\langle\alpha\rangle\top \to a \sqsubseteq \alpha,\ \langle\beta\rangle\top \to a \sqsubseteq \beta\}$$

We have $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models a \sqsubseteq \alpha \wedge a \sqsubseteq \beta$ *due to* $\mathtt{Fml}(\mathtt{pre})$ *and* $\mathtt{Fml}(\mathrm{ref})$. *By Proposition 5.1 we then have* $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models [a]q \wedge [a]\neg q$ *due to* $\mathtt{Fml}(\mathtt{post})$*, and therefore* $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \bot$ *due to* $\mathtt{Fml}(\mathtt{pre})$. *However, it is not the case that* $\mathtt{Fml}(\mathtt{pre}) \cup \mathtt{Fml}(\mathtt{post}) \models \bot$ *and consequently Constraint* (**PM**) *is violated.*

5.5.2 Coherence Condition of HTN Domains

Another important postulate of HTN planning domains concerns coherence. The dynamic logic actually provides HTN planning with a logical semantics to evaluate the soundness of HTN domains.

Definition 5.1. An HTN planning domain $\mathcal{D}_{\mathsf{htn}}$ is *coherent* if $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}})$ is satisfiable.

Intuitively, the soundness of HTN planning domains captures that once an action is performed, its postcondition will hold and that once an action is refined, its refinement will satisfy its postcondition.

Let us revisit the abstract example:

Example 5.5. *The planning domain of Example 5.1 is captured by*

$$\mathtt{Fml}(\mathtt{pre}) = \{\langle\alpha\rangle\top \leftrightarrow \varphi_\alpha,\ \langle\beta\rangle\top \leftrightarrow \varphi_\beta,\ \langle\gamma\rangle\top \leftrightarrow \top\}$$
$$\mathtt{Fml}(\mathtt{post}) = \{[\alpha]p,\ [\beta]\top,\ [\gamma]\neg p,\}$$
$$\mathtt{Fml}(\mathrm{ref}) = \{\langle\alpha\rangle\top \to \beta \sqsubseteq \alpha,\ \langle\beta\rangle\top \to \gamma \sqsubseteq \beta\}$$

Suppose the above HTN domain is coherent and M is one of its model. When $\varphi_\alpha \wedge \varphi_\beta$ is satisfiable then there exists a pointed model (M, w) such that $M, w \Vdash$

$\varphi_\alpha \wedge \varphi_\beta$. Then we have $M, w \Vdash \beta \sqsubseteq \alpha \wedge \gamma \sqsubseteq \beta$, and in consequence $M, w \Vdash \gamma \sqsubseteq \alpha$. By $[\alpha]p \wedge [\gamma]\neg p$, for every $w' \in R_\alpha(w)$, $M, w' \Vdash p \wedge \neg p$. Thus, such a world w cannot exist and there is some logical inconsistency between the precondition of the actions.

When $\varphi_\alpha \wedge \varphi_\beta$ is unsatisfiable then the HTN domain is coherent, but action α can never be refined into a plan.

5.5.3 Soundness Postulate of Actions

We now formulate a soundness postulate for high-level actions. It requires that when a high-level action α is executable then every possible refinement of α guarantees the postcondition of α. This is conditional on the precondition of α: if they are false then there is no point in refining α and π may have arbitrary consequences.

Definition 5.2. Given a model M and world w in M of HTN planning domain $\mathcal{D}_{\mathsf{htn}}$, we say that the high-level action $\alpha \in \mathtt{Act}$ is *soundly refinable at* (M, w) on $\mathcal{D}_{\mathsf{htn}}$ if and only if either $M, w \nVdash \mathtt{pre}(\alpha)$ or for every $\pi \in \mathrm{ref}(\alpha)$ and $v \in R_\pi(w)$, $M, v \Vdash \mathtt{post}(\alpha)$.

If the action α is soundly refinable at every pointed model (M, w) in HTN planning domain $\mathcal{D}_{\mathsf{htn}}$, we say α is soundly refinable in $\mathcal{D}_{\mathsf{htn}}$.

Indeed a coherent HTN planning domain guarantees the soundly refinability of actions, as the theorem shows:

Theorem 5.8. *If the HTN planning domain $\mathcal{D}_{\mathsf{htn}}$ is coherent, then every high-level action in* $\mathtt{Act}$ *is soundly refinable in $\mathcal{D}_{\mathsf{htn}}$.*

Proof. Suppose the HTN planning domain $\mathcal{D}_{\mathsf{htn}}$ is sound then $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}})$ is satisfiable. Assume M is a model of $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}})$ and M contains a possible world w. If $M, w \nVdash \mathtt{pre}(\alpha)$ then α is soundly refinable at (M, w). Otherwise, by $\mathtt{Fml}(\mathrm{ref})$, for all $\pi \in \mathrm{ref}(\alpha)$, we have $M, w \Vdash \pi \sqsubseteq \alpha$. As $\mathtt{Fml}(\mathtt{post})$, $M, w \Vdash [\alpha]\mathtt{post}(\alpha)$. It entails that for every $v \in R_\pi(w)$, $M, v \Vdash \texttt{post}(\alpha)$, and in consequence that α is soundly refinable at (M, w). $\square$

From the above theorem, we can conclude that for every high-level action, the following holds:

$$\texttt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \texttt{pre}(\alpha) \to \big[\bigsqcup \mathrm{ref}(\alpha)\big]\texttt{post}(\alpha).$$

Actually, the consequent captures the soundly refinability of action α.

5.5.4 Completeness Postulate of Actions

Symmetrically to soundness postulate, one may formulate a postulate of completeness: when the precondition of a high-level action is true then it should be refinable in some way.

Definition 5.3. Given a model M and world w in M of HTN planning domain $\mathcal{D}_{\mathsf{htn}}$ we say that high-level action $\alpha \in \texttt{Act}$ is *completely refinable at* (M, w) on $\mathcal{D}_{\mathsf{htn}}$ if and only if either $M, w \nVdash \texttt{pre}(\alpha)$ or there is a $\pi \in \mathrm{ref}(\alpha)$ such that $R_\pi(w)$ is not empty.

If the action α is completely refinable at every pointed model (M, w) in HTN planning domain $\mathcal{D}_{\mathsf{htn}}$, we say α is completely refinable in $\mathcal{D}_{\mathsf{htn}}$.

In other words, in every possible, as long as the precondition of α is true, then one of the programs refining α is executable. Next we show that the completely refinability can be captured in PDL.

Theorem 5.9. *A high-level action* $\alpha \in \texttt{Act} \setminus \texttt{Act}_0$ *is completely refinable in HTN planning domain* $\mathcal{D}_{\mathsf{htn}}$ *if and only if*

$$\texttt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \texttt{pre}(\alpha) \to \big\langle \bigsqcup \mathrm{ref}(\alpha)\big\rangle \top.$$

Proof. "$\Rightarrow$:" Assume the high-level action α is completely refinable in the HTN domain $\mathcal{D}_{\mathsf{htn}}$. Suppose M is a model of $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}})$ and $M, w \Vdash \mathtt{pre}(\alpha)$. Then there is a $\pi \in \mathrm{ref}(\alpha)$ such that $R_\pi(w)$ is not empty. So $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \langle \pi \rangle \top$. By the existence of π, we have $\langle \bigsqcup \mathrm{ref}(\alpha) \rangle \top$.

"$\Leftarrow$:" Assume $\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \mathtt{pre}(\alpha) \to \langle \bigsqcup \mathrm{ref}(\alpha) \rangle \top$. If $M, w \nVdash \mathtt{pre}(\alpha)$ then α is completely refinable at (M, w). Else as $\langle \bigsqcup \mathrm{ref}(\alpha) \rangle \top$, we have one $\pi \in \mathrm{ref}(\alpha)$ such that $M, w \Vdash \langle \pi \rangle \top$. It means $R_\pi(w)$ is not empty, entailing that α is completely refinable at (M, w). □

The soundness of HTN planning domains can not guarantee the completely refinability of every high-level action. Let us take up the example of travelling from A to B:

Example 5.6. *Consider the theory* $\mathtt{Fml}(\mathcal{D}^{\mathsf{AB}}_{\mathsf{htn}})$ *in Example 5.2, it is not difficult to check* $\mathtt{Fml}(\mathcal{D}^{\mathsf{AB}}_{\mathsf{htn}})$ *is satisfiable, that is, the HTN planning domain* $\mathcal{D}^{\mathsf{AB}}_{\mathsf{htn}}$ *is sound. The high-level action* goAB *is completely refinable in* $\mathcal{D}^{\mathsf{AB}}_{\mathsf{htn}}$*, because* goAB *can be refined into* walkAB*, it means that once the precondition* AtA *of the action* goAB *holds, one of its refinements* walkAB *can performed. But, the high-level action* TaxiAB *is not completely refinable in* $\mathcal{D}^{\mathsf{AB}}_{\mathsf{htn}}$*, because its unique refinement* (rideAB; pay) *requires* Money *which the precondition* AtA *of the action* TaxiAB *can not entail.*

We conclude this section by showing that complete refinability can be weakened, viz. by requiring that an executable high-level α must be refinable *unless there is no primitive plan achieving the primary effect of* α. In formulas, we require

$$\mathtt{Fml}(\mathcal{D}_{\mathsf{htn}}) \models \big(\mathtt{pre}(\alpha) \wedge \big\langle \big(\bigsqcup \mathtt{Act}_0\big)^* \big\rangle \mathtt{post}(\alpha)\big) \to \big\langle \bigsqcup \mathrm{ref}(\alpha) \big\rangle \top.$$

This is similar to what is called *planner completeness* in [Kambhampati et al., 1998], which, as we understand it, requires that every solution that can be obtained by a classical planner is also obtainable by the HTN planner. This requirement is different from what is called *schema completeness* in [Kambhampati et al., 1998], which is difficult to capture formally: it basically requires that ref lists all refinements that are intuitively desirable.

Example 5.7. *For the weaken completeness postulate, the high-level action* TaxiAB *in Example 5.2 is still not completely refinable in* $\mathcal{D}_{\mathsf{htn}}^{\mathsf{AB}}$.

5.6 Discussion and Summary

Our work is related to the representation of HTN in Situation Calculus by Baral and Son [1999] who extended the high-level action programming language Con-Golog by adding a special HTN construct. Later, Gabaldon [2002] encoded HTN planning problem by Golog and ConGolog by means of the their **proc** operator, without adding an HTN construct. In both cases, they did not question the properties of the HTN planning domain by taking advantage of the reasoning potential of the Situation Calculus. More recently, Goldman [2009] gave a semantics in terms of ConGolog where HTN domains are restricted to be totally ordered by the constraints or not to be ordered at all.

The soundness problem of HTN planning also concerns the verification problem. [Behnke et al., 2015] defined HTN plan verification as the problem of determining whether a task network is a solution to a planning problem. Later they considered to verify a sequence of primitive tasks instead of a primitive task network [Behnke et al., 2017, Bercher et al., 2016]. But in this chapter, we evaluate the coherence of the HTN planning domain rather than evaluate the verification of a task network. Indeed, for an HTN problem with an incoherent planning domain, it is possible to find a solution to the planning problem, in the case that there are no contradictive decomposition methods selected. In other words, we focus on the whole planning domain while the verification problem is in terms of a sequence of applying decomposition methods.

In this chapter we proposed a representation of HTN in PDL augmented by program inclusion. Taking advantage of the PDL program operators, we can represent more general HTN decomposition methods compared with the standard HTN planner, such as the iteration of actions. Moreover, we discussed postulates which a reasonable HTN planning domain should have, including modularity postulate and coherence condition for domains and soundness and completeness postulates for high-level actions, with formalizing them in PDL.

The extension of PDL with the inclusion of programs is related to grammar logics [Demri, 2001, Fariñas del Cerro and Penttonen, 1988]. Later, Zhou and Zhang [2009] proposed the idea of inclusion in PDL and proved that the extension is more expressive than the standard PDL. But the decidability and complexity are absent. However, given results on grammar logics, our extension of PDL is undecidable where the undecidability comes from the reduction of the ambiguity problem for the context-free grammar. However, we are convinced that in particular for HTN planning, it is possible to find a fragment corresponding to regular grammars are decidable by restricting the syntax of PDL.

In this chapter, we only consider a restricted HTN problems where all tasks are totally ordered by the constraints and the "maintenance" state constraints are not allowed so that they can be expressed by sequential composition and tests. We believe that the rich language of programs of PDL, with such as the iteration, helps us to design a more general way to refine high-level actions.

Chapter 6

HTN with Task Insertion and State Constraints

As mentioned before, solutions of hierarchical task network (HTN) planning are generated only by refining high-level actions step-by-step. It is usually a challenge to provide a complete domain which includes all possible decomposition methods for all compound tasks, while defining only a partially hierarchical domain is not sufficient to produce all desired solutions. Hierarchical task network with task insertion (TIHTN) planning [Geier and Bercher, 2011] relaxes the restriction on solutions and allows solutions generated not only by the decomposition of high-level actions, but also by the insertion of basic actions from outside the decomposition hierarchy.

The complexity of the plan-existence problem for HTN planning is undecidable, even for propositional HTN planning [Erol et al., 1994a]. While for TIHTN planning, the complexity of the plan-existence problem is reduced to **NEXPTIME**-complete for propositional case and to 2-**NEXPTIME**-complete for lifted case [Alford et al., 2015b]. The lifted TIHTN problem differs from propositional TIHTN problem in the input language: the former accepts first-order language without function while the latter accepts a propositional language.

Unfortunately, only ordering constraints are considered in TIHTN planning. In this chapter, we further extend the TIHTN planning by adding state constraints which are used to specify some states should satisfy some formulas. We show that just as for TIHTN planning, the solutions of extended TIHTN planning can be obtained by acyclic decomposition and task insertion, entailing that it is decidable without any restriction on decomposition methods. We also prove that the extension by state constraints does not cause an increase in the complexity of the plan-existence problem, which stays 2-**NEXPTIME**-complete (**NEXPTIME**-complete for the propositional case), based on an acyclic progression operator. We also show that TIHTNS planning includes lifted TIHTN planning. As under task insertion semantics, hierarchical goal network (HGN) planning [Shivashankar et al., 2013] and goal-task network (GTN) planning [Alford et al., 2016] can be converted into lifted TIHTN planning. Thus, our framework, TIHTNS planning, actually covers these two kinds of planning approaches. In addition, we give an alternative embedding of hierarchy-relaxed HGN (HR-HGN) planning [Shivashankar et al., 2017] in TIHTNS planning without introducing fresh actions.

The rest of the chapter is structured as follows. Section 6.1 shows the motivation of taking state constraints into account. Section 6.2 presents the definitions of TIHTNS planning. Section 6.3 presents the acyclic decomposition property of TIHTNS planning. Section 6.4 proposes the decidability and complexity results of TIHTNS Section 6.5 shows how to embed lifted TIHTN and HR-HGN into TIHTNS. Section 6.6 summarizes this chapter.

6.1 State Constraints

State constraints are considered in the conventional HTN planning [Erol et al., 1994a], but they are not taken into account in TIHTN planning. State constraints can capture the pre- and postcondition of compound tasks, though in the standard HTN planning there is no notion of pre- and postcondition of compound tasks. A compound task is considered as accomplished if its subtasks are accomplished.

With a state constraint, a formula, as a postcondition, can be required to hold after accomplishing a compound task.

Every ordering constraint requires that a task t_1 must be performed ahead of another task t_2, denoted by $t_1 \prec t_2$. However, ordering constraints cannot fully represent state constraints. The 'immediate' state constraint, which requires a formula holds immediately before or after a compound task, can be simulated via introducing a virtual subtask to check whether the formula holds. However, the 'maintenance' state constraints (also called trajectory constraints in the Planning Domain Definition Language 3 (PDDL3) [Gerevini and Long, 2005]) cannot be represented easily. For instance, suppose a robot is required to always keep 10% battery for emergency. Its initial compound task is to "clean a room", decomposed into "clean the ground" (t_1) and "clean the table" (t_2). Suppose that "clean the ground" requires the full battery. Then there is no solution for the TIHTNS planning problem with such a state constraint. For the original TIHTN planning, there is an intuitive attempt to simulate that constraint: introducing a virtual primitive task pt whose precondition is "more than 10% battery" and introducing two ordering constraints $t_1 \prec pt$ and $pt \prec t_2$. Assume "charge" means to charge the battery, then the plan $\langle t_1$;"charge";pt;$t_2\rangle$ is a solution of the original TIHTN planning problem. But it is counter-intuitive and the state constraint of keeping 10% battery is still violated after performing t_1.

Furthermore, state constraints are introduced into PDDL3 [Gerevini and Long, 2005] to support hard constraints over state properties of a trajectory and the specification of preferences. Later, [Sohrabi et al., 2009] extends PDDL3 into HTN planning where state constraints are used to capture user preferences. State constraints are also necessary in real-world applications, such as in web service composition where state constraints are used to describe user preferences [Lin et al., 2008] and to additionally capture the enforcement of regulations [Sohrabi and McIlraith, 2009].

6.2 HTN with Task Insertion and State Constraints

In the section, we show how state constraints can be smoothly integrated into TIHTN planning and we call the extension as hierarchical task network planning with task insertion and state constraints (TIHTNS).

Actually, [Geier and Bercher, 2011] proposes an alternative simplified formalism of HTN planning, compared to the conventional HTN of [Erol et al., 1994a]. They also extend propositional HTN planning into propositional TIHTN planning, based on the simplified formalism. The lifted TIHTN planning [Alford et al., 2015b] extends the propositional TIHTN planning by accepting first-order language. Their semantics are the same and after grounding, a lifted TIHTN problem becomes a propositional TIHTN problem. Now, we extend the lifted TIHTN planning into TIHTNS planning, following their formalism. As we show in Chapter 2, Geier and Bercher use an alternative terminology which is different from that of Erol. For example, Geier and Bercher use "task name" to denote "action type" while Erol use "task symbol". Let us recall the unification of the terminology: the action type for the compound task is called "high-level action" and for the primitive task is called "basic action" and that the action token is simply called "task".

Next we adapt the input language of the lifted TIHTN planning in [Alford et al., 2015b] based on our unified terminology.

First, we define a function-free first order language $\mathscr{L}$ from a set of variables and a finite set $\mathscr{L}_0$ of predicates and constants. We use $\mathsf{Atm}(\mathscr{L})$ to denote the set of atomic formulas of $\mathscr{L}$ and $\mathsf{Atm}(\mathscr{L}_0)$ to denote the set of ground atoms of $\mathscr{L}$.

Next we use *tasks*, which are syntactically variables, to identify different action tokens. Every task is associated with an *action*, which is syntactically a first-order atom with variables and constants in $\mathscr{L}$. That is, every action typically is associated with an arity and contains variables which are quantified in a static domain, so that they can be grounded in the usual way. Those actions which can be executed directly are called *basic actions*, noted $\mathcal{O}$, while others are called *high-level actions*, noted $\mathcal{C}$.

Just as for the database perspective, we use a dynamic theory to describe the behavior of actions.[1] Similar to Definition 3.1, a dynamic theory is a tuple $\mathcal{D} = \langle \mathtt{pre}, \mathtt{eff}^{+}, \mathtt{eff}^{-} \rangle$ where $\mathtt{pre} : \mathcal{O} \longrightarrow \mathscr{L}$ and $\mathtt{eff}^{+}, \mathtt{eff}^{-} : \mathcal{O} \longrightarrow 2^{\mathsf{Atm}(\mathscr{L})}$ where $\mathtt{eff}^{+}(o) \cap \mathtt{eff}^{-}(o) = \emptyset$ for every $o \in \mathcal{O}$.

We suppose that $\mathcal{O}$ contains the 'empty' basic action skip where $\mathtt{pre}(\mathsf{skip}) = \top$ and $\mathtt{eff}^{+}(\mathsf{skip}) = \mathtt{eff}^{-}(\mathsf{skip}) = \emptyset$.

For a function $f : R \longrightarrow S$, its restriction to a subset X of its domain is $f|_X = \{(r, s) \in f \mid r \in X\}$. Given a binary relation $Q \subseteq R \times R$, we define its restriction to $X \subseteq R$ by $Q|_X = Q \cap (X \times X)$; similarly for ternary relations. We extend the set union basic action $\cup$ to relations: $(R_1, R_2) \cup (R'_1, R'_2) = (R_1 \cup R'_1, R_2 \cup R'_2)$ and extend functions to sequences: $f(\langle t_1, \ldots, t_n \rangle) = \langle f(t_1), \ldots, f(t_n) \rangle$.

6.2.1 HTN Problems

We start to introduce the syntax of HTN planning by task networks.

Definition 6.1 (**Task networks**)**.** A task network $\mathsf{tn} = (T, \Delta, \alpha)$ is a tuple, where

- T is a finite and non-empty set of tasks;
- $\Delta \subseteq (T \cup \{\mathsf{nil}\}) \times \mathscr{L} \times (T \cup \{\mathsf{nil}\})$ is a set of constraints over T
- $\alpha : T \to \mathcal{C} \cup \mathcal{O}$ labels every task with an action.

A task is an instance of an action. With the function α, we allow multiple instances of an action in a task network. If a task is associated with a high-level action, we call it *compound task*, otherwise *primitive task*. A task network is *primitive* if it only contains primitive tasks, otherwise non-primitive.

[1] In the literature of TIHTN planning, basic actions (operators) are denoted in the form of tuples of name, precondition, postcondition, as we show in Section 2.2.

Compared to the ordering constraints in TIHTN planning which are in form of task-task pairs, we use a triple (t_i, φ, t_j) to denote a state constraint which intuitively means that formula φ must be true in all states between t_i and t_j. Specially, we introduce an idle task nil which designates a task that is accomplished immediately: $(\mathsf{nil}, \varphi, t_j)$ and $(t_i, \varphi, \mathsf{nil})$ mean formula φ holds immediately before t_j and after t_i, respectively. We suppose nil only occurs in constraints. When φ is the truth constant $\top$ then the state constraint (t_i, φ, t_j) becomes an ordering constraint that just requires that t_i is before t_j.

Definition 6.2 (Isomorphic task networks). We say that two task networks $\mathsf{tn} = (T, \Delta, \alpha)$ and $\mathsf{tn}' = (T', \Delta', \alpha')$ are *isomorphic*, noted $\mathsf{tn} \cong \mathsf{tn}'$, if there exists a bijection $\delta : T \to T'$ where for all $t, t' \in T$ it holds that $\alpha(t) = \alpha'(\delta(t))$ and $(t, \varphi, t') \in \Delta$ iff $(\delta(t), \varphi, \delta(t')) \in \Delta'$.

Non-primitive task networks contain compound tasks which cannot be executed directly by the agent, and decomposition methods tell us how to decompose these hierarchically.

Definition 6.3 (Methods). Each decomposition method m is a tuple (c, tn_m), where c is a high-level action, called the decomposition method's head, and tn_m is a task network, whose inner tasks are called the decomposition method's subtasks.

The intuition of decomposition methods is that high-level action c can be reduced by the subtask network tn_m.

A TIHTNS (planning) problem only differs from an HTN problem or a TIHTN problem in the solution criterion and they share the syntactical problem description. An HTN problem is based on an HTN domain.

Definition 6.4 (HTN domains). An HTN domain $\mathfrak{D}$ is a tuple $(\mathscr{L}, \mathcal{C}, \mathcal{O}, \mathcal{D}, \mathcal{M})$ where $\mathscr{L}$ is the input language, $\mathcal{M}$ is a set of decomposition methods, $\mathcal{D}$ is the dynamic theory, $\mathcal{C}$ and $\mathcal{O}$ are the sets of high-level actions and basic actions, respectively, such that $\mathcal{C} \cap \mathcal{O} = \emptyset$.

Now we have prepared to define HTN problems.

Definition 6.5 (HTN problems). Given an HTN domain $\mathfrak{D}$, an HTN problem is a tuple $\mathcal{P} = (\mathfrak{D}, s_I, \mathsf{tn}_I)$ where s_I is the ground initial state which is a subset of $\mathsf{Atm}(\mathscr{L}_0)$ and tn_I is the initial task network.

Particularly, if $\mathfrak{D}$ does not contain any variables, we call the problem $\mathcal{P}$ a propositional problem.

The semantics of HTN planning is given through grounding. As the set of predicates and constants $\mathscr{L}_0$ is finite, it is easy to translate a lifted HTN problem into a ground HTN problem. Next we introduce how to ground the HTN problem $\mathcal{P} = (\mathscr{L}, \mathcal{C}, \mathcal{O}, \mathcal{D}, \mathcal{M}, s_I, \mathsf{tn}_I)$.

Formulas in the first-order language $\mathscr{L}$ and actions are grounded in the usual way: assigning variables in $\mathscr{L}$ to constants in $\mathscr{L}$. Let V be all assignments of $\mathscr{L}$, we use $C = \{c[v] \mid c \in \mathcal{C}\}$ and $O = \{o[v] \mid o \in \mathcal{O}\}$ to denote respectively the ground high-level action set and the basic action set, where for every assignment $v \in V$, the syntax $\chi[v]$ s.t. $(\chi = c, o, \mathsf{tn})$, denote syntactic variable substitution of the variables in expression χ, with the matching terms from v. For example, by grounding the action "$\mathsf{openDoor}(X)$" where X is quantified in a "door" domain $\{d_1, d_2, ...\}$, we can obtain a set of ground actions "$\mathsf{openDoor}(d_1)$", "$\mathsf{openDoor}(d_2)$", etc.

The dynamic theory needs to be adapted to the basic actions: the variables in $\mathtt{pre}(o), \mathtt{eff}^+(o)$ and $\mathtt{eff}^-(o)$ associated with the variables in the basic action o are instantiated simultaneously while other variables are grounded in the usual way. For example, given the basic action $o = \mathsf{openDoor}(X)$ where $\mathtt{pre}(o) = \exists Y.HasKey(Y)$ and $\mathtt{eff}^+(o) = Opened(X)$, $\mathtt{eff}^-(o) = \emptyset$ where Y is quantified in a "key" domain K, by substituting X with d_1, we obtain a basic action $o' = \mathsf{openDoor}(d_1)$ with $\mathtt{pre}(o') = \bigvee_{k \in K} HasKey(k)$ and $\mathtt{eff}^+(o') = Opened(d_1)$, $\mathtt{eff}^-(o') = \emptyset$. We use $\mathcal{D}_0$ to denote the ground dynamic theory obtained from the dynamic theory $\mathcal{D}$.

Similarly, the ground decomposition method set M obtained from decomposition method set $\mathcal{M}$ is given by $\bigcup_{v \in V} \{(c[v], \mathsf{tn}[v]) \mid (c, \mathsf{tn}) \in \mathcal{M}\}$.

Then we call the tuple $\mathfrak{D}^0 = (\mathscr{L}, C, O, \mathcal{D}_0, M)$ a ground HTN domain of the domain $\mathfrak{D} = (\mathscr{L}, \mathcal{C}, \mathcal{O}, \mathcal{D}, \mathcal{M})$. For the problem $\mathcal{P} = (\mathfrak{D}, s_I, \mathsf{tn}_I)$, we say $\mathcal{P}^0 = (\mathfrak{D}^0, s_I, \mathsf{tn}_I^0)$, where tn_I^0 is a grounding of the initial task network tn_I, is a ground problem of the problem $\mathcal{P}$. The solutions of an HTN problem are all solutions of its ground problems. Observe that the ground problem is actually a propositional HTN problem.

A ground state is a subset of $\mathsf{Atm}(\mathscr{L}_0)$. A set of basic actions O determines a state-transition function $\gamma : 2^{\mathsf{Atm}(\mathscr{L}_0)} \times O \longrightarrow 2^{\mathsf{Atm}(\mathscr{L}_0)}$, where:

- $\gamma(s, o)$ is defined iff $s \models \mathtt{pre}(o)$;
- $\gamma(s, o) = (s \setminus \mathtt{eff}^-(o)) \cup \mathtt{eff}^+(o)$ if $\gamma(s, o)$ is defined.

A sequence of basic actions $\langle o_1, \ldots, o_n \rangle$ is executable in a state s_0 iff there exists a sequence of states $s_1, \ldots, s_n$ such that for all $1 \leq i \leq n$, $\gamma(s_{i-1}, o_i) = s_i$.

Here is a simple example of propositional HTN planning problem.

Example 6.1. *Suppose we have an initial high-level action* gMC *for "go to Melbourne center", the basic action* flyM *for "fly to Melbourne", and the basic action* takeTaxi *for "take a taxi to the center". We use* At(Mc)*and* At(Ma) *to denote "being at Melbourne center" and "being at Melbourne airport". The dynamic theory* $\mathfrak{D}^0$ *is given by:*

- $\mathtt{pre}(\mathsf{flyM}) = \top, \mathtt{eff}^+(\mathsf{flyM}) = \mathsf{At(Ma)}, \mathtt{eff}^-(\mathsf{flyM}) = \emptyset$
- $\mathtt{pre}(\mathsf{takeTaxi}) = \mathsf{At(Ma)}, \mathtt{eff}^+(\mathsf{takeTaxi}) = \mathsf{At(Mc)}, \mathtt{eff}^-(\mathsf{takeTaxi}) = \mathsf{At(Ma)})$

and the decomposition method is:

- $m = (\mathsf{gMC}, \mathsf{tn}_m)$, *where* $\mathsf{tn}_m = (t_2, \emptyset, (t_2, \mathsf{flyM}))$

Then we have an HTN problem $\mathcal{P} = (\mathscr{L}, \mathsf{gMC}, \{\mathsf{flyM}, \mathsf{takeTaxi}\}, \mathcal{D}_0, m, s_I, \mathsf{tn}_I)$ *with* $s_I = \emptyset$ *and* $\mathsf{tn}_I = (t_1, (t_1, \mathsf{At(Mc)}, \mathsf{nil}), (t_1, \mathsf{gMC}))$.[2]

[2] We will sometimes omit the braces of singleton sets.

6.2.2 Task Decomposition

Next we borrow the notion of decomposition in [Geier and Bercher, 2011] and define how to decompose a compound task with a task network.

In order to indicate the starting point and the end point of a compound task t, we introduce a pair of 'dummy' primitive tasks, noted $*t$ and $t*$.

As nil only occurs in constraints, we suppose the restriction of constraint set Δ to a set of tasks T is $\Delta|_T = \Delta \cap ((T \cup \{\mathsf{nil}\}) \times \mathscr{L} \times (T \cup \{\mathsf{nil}\}))$.

Definition 6.6 (**Decomposition**). Given a ground HTN domain, and a task network tn=(T, Δ, α). Let $t \in T$ be a compound task and m =$(\alpha(t), (T_m, \Delta_m, \alpha_m))$ be a decomposition method. Suppose a task network $\mathsf{tn}'_m = (T'_m, \Delta'_m, \alpha'_m)$ such that $\mathsf{tn}'_m \cong \mathsf{tn}_m$ and $T'_m \cap T = \emptyset$. The decomposition of task t by decomposition method m is $\mathsf{tn}' = (T', \Delta', \alpha')$ where

$$\begin{aligned}
T' &= (T \setminus \{t\}) \cup T'_m \cup \{*t, t*\} \\
\Delta' &= \Delta|_{T\setminus\{t\}} \cup \Delta'_m \cup \{(*t, \top, t_j), (t_j, \top, t*) \mid t_j \in T'_m\} \\
&\quad \cup \{(t_1, \varphi, *t) \mid (t_1, \varphi, t) \in \Delta\} \\
&\quad \cup \{(t*, \varphi, t_2) \mid (t, \varphi, t_2) \in \Delta\} \\
\alpha' &= \alpha|_{T\setminus\{t\}} \cup \alpha'_m \cup \{(*t, \mathsf{skip}), (t*, \mathsf{skip})\}
\end{aligned}$$

We write $\mathsf{tn} \underset{t,m}{\longrightarrow} \mathsf{tn}'$ when tn' is the decomposition of t by m.

In the resulting task network, the decomposed compound task is replaced with subtasks defined by the decomposition method applied and its corresponding starting and terminating tasks. The latter two are dummy tasks and are mapped to the action skip. All state constraints about the decomposed task t are propagated to $*t$ and $t*$ in Δ'. More precisely, if φ holds before t then it also holds before $*t$ and if φ' holds after t then it also holds after $t*$. Subtasks should satisfy the inner constraints introduced by the decomposition method m and should be performed between $*t$ and $t*$. In order to distinguish the tasks in the original task network

tn, the introduced subtask network tn'_m is isomorphic with the subtask network tn_m defined in the domain, guaranteeing that all introduced subtasks T'_m are fresh.

Here we show an example of a step of task decomposition.

Example 6.2 (Example 6.1 continued)**.** *We apply the decomposition method m in tn_I to decompose t_1, i.e., $\mathsf{tn}_I \xrightarrow[t_1,m]{} \mathsf{tn}'$, where tn' is:*

$$\begin{aligned}
T' =& \{t_2, *t_1, t_1*\} \\
\Delta' =& \{(t_2, \top, t_1*), (*t_1, \top, t_2),\ (t_1*, \mathsf{At}(\mathsf{Mc}), \mathsf{nil})\} \\
\alpha' =& \{(t_2, \mathsf{flyM}), (*t_1, \mathsf{skip}), (t_1*, \mathsf{skip})\}
\end{aligned}$$

After a step of task decomposition, we introduce the hierarchical procedure of task decomposition which can be viewed as a tree. As the initial task network may contain more than one task and every task generates a tree via decompositions, it leads to a forest for the initial task network. In order to integrate them into a tree, we create a “root”: we introduce a new high-level action c_{top} not occurring in $\mathfrak{D}$ with meaning “to accomplish all the initial tasks” and use a new task t_0 to identify it. We restrict c_{top} to only decompose into the original initial task network tn_I for the ground HTN problem.

Definition 6.7 (**Decomposition trees**)**.** Given a ground HTN domain $\mathfrak{D}^0 = (\mathscr{L}, C, O, \mathcal{D}_0, M)$, a decomposition tree is a five-tuple $\mathsf{Tr} = (T, E, \Delta, \alpha, \beta)$ where

- (T, E) is a tree, rooted in t_0, with nodes T and with directed edges E pointing towards the leaves;
- Δ is a set of constraints over T;
- Function $\alpha : T \longrightarrow C \cup O \cup \{c_{top}\}$ links tasks and actions where $\alpha(t_0) = c_{top}$ such that c_{top} is the unique high-level action;
- Function $\beta : T \longrightarrow M'$ labels inner nodes with decomposition methods where $M' = M \cup \{(c_{top}, \mathsf{tn}_0)\}$ and tn_0 is some task network.

We write $\tau(\mathsf{Tr})$ for the set of the nodes (tasks) of the decomposition tree Tr and $\mathsf{ch}(\mathsf{Tr}, t)$ for the set of the direct children of $t \in \tau(\mathsf{Tr})$ in Tr. We use $\mathsf{sub}(t)$ to denote the set of subtasks of t. We say the task network introduced by the leaf nodes of Tr together with the constraints about these nodes as the leaf task network of Tr, denoted by $\vartheta(\mathsf{Tr})$.

Definition 6.8 (**Valid decomposition trees**). A decomposition tree Tr is valid w.r.t. a ground HTN problem $\mathcal{P}^0 = (\mathfrak{D}^0, s_I, \mathsf{tn}_I^0)$ iff the root node of Tr is t_0 where $\beta(t_0) = (c_{top}, \mathsf{tn}_I^0)$ for any inner node t where $\beta(t) = (c, \mathsf{tn}_m)$, the following hold:

1. $\alpha(t) = c$
2. $\mathsf{ch}(\mathsf{Tr}, t) = \mathsf{sub}(t) \cup \{*t, t*\}$ such that
   - $(\mathsf{sub}(t), \Delta|_{\mathsf{sub}(t)}, \alpha|_{\mathsf{sub}(t)}) \cong \mathsf{tn}_m$
   - for every $t' \in \mathsf{sub}(t)$: $(t', \top, t*), (*t, \top, t') \in \Delta$
3. for every $t' \in \tau(\mathsf{Tr}) \cup \{\mathsf{nil}\}$:
   - if $(t, \varphi, t') \in \Delta$ then $(t*, \varphi, t') \in \Delta$
   - if $(t', \varphi, t) \in \Delta$ then $(t', \varphi, *t) \in \Delta$
4. there is no constraint in Δ except for those demanded by criterion 2. and 3.

When tn' is reachable from tn by a finite sequence of decompositions then we write $\mathsf{tn} \longrightarrow_D^* \mathsf{tn}'$.

The decomposition tree focuses on the whole procedure of decomposition and records all intermediate tasks and constraints during the procedure. In contrast, in a task network the application of a decomposition method abandons the decomposed task. However, they are compatible as the following proposition states.

Proposition 6.1. *Given a ground HTN problem $\mathcal{P}^0$, $\mathsf{tn}_I^0 \longrightarrow_D^* \mathsf{tn}$ iff there exists a valid decomposition tree Tr with respect to $\mathcal{P}$ where $\vartheta(\mathsf{Tr}) = \mathsf{tn}$.*

Proof. "$\Rightarrow$": We use induction over the number of inner nodes of the valid decomposition tree Tr. As a valid decomposition tree has a root t_0 with $\beta(t_0) = (c_{top}, \mathsf{tn}_0)$, Tr at least has one inner node and $|T_I^0|$ leaf nodes where $|T_I^0|$ is the number of the tasks in the initial task network tn_I^0. We take one inner node as the base case and then we have $\vartheta(\mathsf{Tr}) = \mathsf{sub}(t_0) = \mathsf{tn}_I^0$.

For the inductive step, suppose a valid decomposition tree Tr with n inner nodes and $\mathsf{tn}_I^0 \longrightarrow_D^* \mathsf{tn}$ and $\vartheta(\mathsf{Tr}) = \mathsf{tn}$. Let a compound task t be leaf node in Tr and suppose t has a decomposition method $m = (\alpha(t), \mathsf{tn}_m)$ and the task network $\mathsf{tn}'_m = (T'_m, C'_m, \alpha'_m)$ where symbols in T_m are fresh is isomorphic with tn_m. By adding edges from t to nodes $*t$, $t*$ and all nodes T_m, we can get a new decomposition tree

$$\mathsf{Tr}' = (T \cup T', E \cup E_m, \Delta \cup \Delta', \alpha \cup \alpha', \beta \cup \{(t, m)\})$$

where T', Δ' and α' are defined as in the definition of decomposition (Definition 6.6) and E_m is the set of the adding edges. It is easy to check that Tr' satisfies all criteria in the definition of the valid decomposition tree (Definition 6.8) and that Tr' is a valid decomposition tree. Then $\vartheta(\mathsf{Tr}') = (T', \Delta', \alpha')$ and $\mathsf{tn} \underset{t,m}{\longrightarrow} \vartheta(\mathsf{Tr}')$.

"$\Leftarrow$": We use induction over the steps of decomposition starting from tn_I^0 to prove there exists such a valid decomposition tree. For the base case that there is no decomposition applied, we have a valid decomposition tree Tr whose root is t_0 and $\vartheta(\mathsf{Tr}) = \mathsf{tn}_I^0$.

For the inductive step, suppose tn_2 is obtained by n steps of decomposition from tn_I^0 and Tr is a valid decomposition tree with $\vartheta(\mathsf{Tr}) = \mathsf{tn}_2$. Suppose $\mathsf{tn}_2 \underset{t,m}{\longrightarrow} \mathsf{tn}_1$. We can extend $\mathsf{Tr} = (T, E, \Delta, \alpha, \beta)$ as

$$\mathsf{Tr}' = (T \cup T', E \cup E', \Delta \cup \Delta_1, \alpha \cup \alpha_1, \beta \cup \{(t, m)\})$$

where $T' = T_1 \backslash T_2$, E' is the set of edges from t to nodes in T' and T_1, T_2 are the task sets of tn_1 and tn_2, respectively. The adding tasks T' actually are $T_m \cup \{*t, t*\}$ and the adding constraints $\Delta_1 \setminus \Delta$ are about the nodes in T' and inherit the constraints about t. We have that Tr' satisfies all the criteria in the definition of valid decomposition tree (Definition 6.8) and is therefore a valid decomposition tree; moreover, $\vartheta(\mathsf{Tr}') = \mathsf{tn}_1$. □

The above proposition also states that every set of decomposition methods corresponds to a valid decomposition tree.

The following proposition states that the procedure of decomposition propagates the state constraints about compound tasks into primitive tasks.

Proposition 6.2. *For two task networks* $\mathsf{tn} = (T, \Delta, \alpha)$ *and* $\mathsf{tn}' = (T', \Delta', \alpha')$ *such that* $\mathsf{tn} \rightarrow_D^* \mathsf{tn}'$*, the following holds:*

- *for every* $(\mathsf{nil}, \varphi, t) \in \Delta$*,* $(\mathsf{nil}, \varphi, *t) \in \Delta'$
- *for every* $(t, \varphi, \mathsf{nil}) \in \Delta$*,* $(t*, \varphi, \mathsf{nil}) \in \Delta'$
- *for every* $(t_i, \varphi, t_j) \in \Delta$ *with* $\alpha(t_i), \alpha(t_j) \notin O$*,* $(t_i*, \varphi, *t_j) \in \Delta'$ [3]

Proof. It is straightforward by Definition 6.6. □

6.2.3 Solutions

A solution of an HTN problem is a sequence of primitive tasks which is also called a *plan* of the problem.

Definition 6.9 (**Consistency with constraints**)**.** Given a primitive task network $\mathsf{tn} = (T, \Delta, \alpha)$ where $|T| = n$, let $\sigma : T \rightarrow \{1, \ldots, n\}$ be a bijection. We use σ to form a total ordering, noted $\sigma(\mathsf{tn})$, of tasks in T as: $\langle \sigma^-(1), \ldots, \sigma^-(n) \rangle$ where σ^- is the inverse function of σ, i.e., $\sigma^-(\sigma(t)) = t$. Suppose $\alpha(\sigma(\mathsf{tn}))$ is executable in s_0, i.e., there exists a sequence $s_1, \ldots, s_n$ such that $\gamma(s_{i-1}, \alpha(t_i)) = s_i$ for every i such that $1 \leq i \leq n$. We say that $\sigma(\mathsf{tn})$ is consistent with Δ in s_0 if for every $\sigma^-(i), \sigma^-(j) \in T$ the following hold:

- for every $(\mathsf{nil}, \varphi, \sigma^-(j)) \in \Delta$, $s_{j-1} \models \varphi$;

[3] t_i, t_j cannot be nil because $\alpha(\mathsf{nil})$ is not defined

- for every $(\sigma^-(i), \varphi, \mathsf{nil}) \in \Delta$, $s_i \models \varphi$;
- for every $(\sigma^-(i), \varphi, \sigma^-(j)) \in \Delta$, $i < j$ and $s_k \models \varphi$ for every $i \leq k < j$.

Intuitively, a 'maintenance' state constraint (t, φ, t') is satisfied if all states between t and t' satisfy φ. An 'immediate' state constraint $(\mathsf{nil}, \varphi, t)$ is satisfied if φ holds in the state right before $*t$ occurs (or t if t is a primitive task), in other words right before all subtasks of t. Finally, an 'immediate' state constraint $(t, \varphi, \mathsf{nil})$ is satisfied if φ holds right after the state where $*t$ (or t if t primitive task) occurs, in other words right after all subtasks of t. Note that it is impossible to satisfy $(t, \bot, t')$ because there is no state s such that $s \models \bot$.

As a solution, a sequence of primitive tasks should be executable as well.

Definition 6.10 (**Executability**). A task network tn is primitive iff it contains only primitive tasks. A primitive task network tn is executable in a state s iff there exists a total ordering $\sigma(\mathsf{tn})$ of the tasks in tn that is consistent with Δ in s. We called such a $\sigma(\mathsf{tn})$ a plan of tn in s, noted $\sigma_{\mathsf{tn},s}$.

It is possible that the task network is not executable in a state. For instance, the primitive task network tn' in Example 6.2 is not executable in s_I because apart from skip it only involves the basic action flyM and it is impossible to satisfy $\mathsf{At}(\mathsf{Mc})$. However, if we extend the task network by inserting some tasks, then we can make it executable.

Definition 6.11 (**Insertion**). Let $\mathsf{tn} = (T, \Delta, \alpha)$ and $\mathsf{tn}' = (T', \Delta', \alpha')$ be two task networks where tn' is primitive. Inserting tn' into tn results in the task network $\mathsf{tn}_1 = \mathsf{tn} \cup \mathsf{tn}'$.

Note that it is not required that $T' \cap T = \emptyset$ because tn' may involve some constraints about tasks in tn.

With respect to some task network tn, if tn' is reachable by a finite sequence of decompositions and an insertion, we write $\mathsf{tn} \longrightarrow^*_{DI} \mathsf{tn}'$. Now we have already prepared to define the solutions of HTN and TIHTNS problems. As mentioned above, HTN problems and TIHTNS problems have the identical problem description, they differ from each other in the ways of generating solutions.

Definition 6.12 (**Solutions**)**.** A plan obtained only by decomposition is called an *HTN solution*; a plan obtained additionally by an insertion is called a *TIHTNS solution*:

Let tn be a primitive task network such that there exists a plan of tn in s_I. Given a propositional HTN problem $\mathcal{P}$, we call σ_{tn,s_I} an HTN solution of $\mathcal{P}$ if $\mathsf{tn}_I \to_D^* \mathsf{tn}$; we call σ_{tn,s_I} a TIHTNS solution of $\mathcal{P}$ if $\mathsf{tn}_I \to_{DI}^* \mathsf{tn}$.

The solutions of an HTN problem are the set of all solutions of its ground problems.

Indeed, HTN problems and TIHTNS problems only differ in the criterion of solutions. When considering whether a problem $\mathcal{P}$ has a TIHTNS solution, we call $\mathcal{P}$ a TIHTNS problem.

Example 6.3 (Example 6.2 continued)**.** *The plan* $\langle *t_1, t_2, t_3, t_1* \rangle$ *where* $\alpha(t_3) = \mathsf{takeTaxi}$ *is a TIHTNS solution of* $\mathcal{P}$*.*

The next proposition states that the state constraints about compound tasks introduced in the decomposition procedure are satisfied by the solutions of the problem.

Proposition 6.3. *Given a TIHTNS problem* $\mathcal{P}$*, suppose* Tr *is a valid decomposition tree with respect to* $\mathcal{P}$ *and* tn' *is a primitive task network obtained from* $\vartheta(\mathsf{Tr})$ *by an insertion and* $\sigma_{\mathsf{tn}',s_I}$ *is a solution of* $\mathcal{P}$*. Then all constraints in the decomposition tree* Tr *are also satisfied by* $\sigma_{\mathsf{tn}',s_I}$*.*

Proof. Suppose there are two compound tasks t_i and t_j in Tr and $\mathsf{tn}' = (T', \Delta', \alpha')$.

For every $(t_i, \varphi, t_j) \in \Delta$, we have $(t_i*, \varphi, *t_j) \in \Delta'$ and then all states s_k such that $\sigma(t_i*) \leq k < \sigma(*t_j)$, $s_k \models \varphi$.

For every $(t_i, \varphi, \mathsf{nil}) \in \Delta$, we have $(t_i*, \varphi, \mathsf{nil}), (st, \top, t_i*) \in \Delta'$ for all subtasks $st \in \mathsf{sub}(t_i)$ and then $s_{\sigma(t_i*)} \models \varphi$ and $\sigma(st) < \sigma(t_i*)$ for all $st \in \mathsf{sub}(t_i)$.

For every $(\mathsf{nil}, \varphi, t_j) \in \Delta$, we have $(\mathsf{nil}, \varphi, *t_j), (*t_j, \top, st) \in \Delta'$ for all subtasks $st \in \mathsf{sub}(t_j)$ and then $s_{\sigma(*t_j)} \models \varphi$ and $\sigma(st) > \sigma(*t_j)$ for all $st \in \mathsf{sub}(t_j)$.

If one or both of t_i, t_j are primitive then just consider $*t_j$ and t_i* as t_j and t_i, respectively. The above three cases still hold.

Then all constraints in Tr are satisfied. □

Remark 6.1. In the formalism of [Alford et al., 2015b] and in TIHTNS planning, the grounding of the problem has been done in the beginning while in the formalism of Erol's [Erol et al., 1995], the grounding happens after a primitive task network is obtained.

Remark 6.2. The semantics of state constraints here is weaker than the semantics of the original HTN planning [Erol et al., 1995]. Their semantics considers every compound task starts by its first 'real' subtask and terminates by its last 'real' subtask instead of the virtual starting and terminating tasks. The distinction between two semantics stands on whether it is allowed to insert tasks between $*t$ and the first real subtask of t (between the last real subtask and $t*$). Our weaker semantics can capture the pre- and postcondition of high-level actions in a better way. In Example 6.1, after the subtask t_2 of gMC is accomplished, the agent's desirable goal "being in the center" is not satisfied while by allowing the insertion of t_3, the goal is achieved.

The following proposition states that if a TIHTNS problem is solvable, then its subproblems are also solvable.

Proposition 6.4. *If the problem* $\mathcal{P} = (\mathfrak{D}, s_I, \mathsf{tn}_I)$ *where* $\mathsf{tn}_I = (T_I, \Delta_I, \alpha_I)$ *has a TIHTNS solution, then problem* $\mathcal{P}' = (\mathfrak{D}, s_I, \mathsf{tn}'_I)$ *where* $\mathsf{tn}_I = (T'_I, \Delta_I|_{T'_I}, \alpha_I|_{T'_I})$ *and* $T'_I \subseteq T_I$ *also has a TIHTNS solution.*

Proof. Suppose Tr is a valid decomposition tree with respect to the ground HTN problem $\mathcal{P}^0$ of $\mathcal{P}$. If we prune the branches about nodes in $T_I \setminus T'_I$ and abandon the constraints about these subtrees and replace (c_{top}, tn_I) by $(c_{top}, \mathsf{tn}'_I)$, then the new decomposition tree Tr' is valid with respect to $\mathcal{P}'$. As the task set and constraint set in the leaf task network $\vartheta(\mathsf{Tr}')$ are the subsets of the task set and constraints set in $\vartheta(\mathsf{Tr})$ respectively, Tr' can form a plan of $\mathcal{P}'$. $\square$

The following proposition states that if a compound task has a decomposition method, then the resulting task network obtained by decomposition preserves solvability.

Proposition 6.5. *For a TIHTNS problem $\mathcal{P}_1$=$(\mathfrak{D}, s_I, \mathsf{tn}_1)$ where tn_1=$(T_1, \Delta_1, \alpha_1)$, if $\mathcal{P}_1$ is solvable and there exists $t \in T_1$ such that $\alpha_1(t)$ has only one decomposition method m in M, then $\mathcal{P}_2$=$(\mathfrak{D}, s_I, \mathsf{tn}_2)$ where $\mathsf{tn}_1 \xrightarrow[t,m]{} \mathsf{tn}_2$ is also solvable.*

Proof. It is straightforward. $\square$

6.3 Acyclic Decomposition

In this section, we introduce the notion of acyclic decomposition sequence. We show that all solutions of TIHTNS problems can be obtained by acyclic decomposition sequence and insertion, which is the foundation of the decidability of the plan-existence problem. It allows us to compute the solution in a finite space rather than to search the solution in a infinitely iterative decomposition.

We first start by adapting the operation of subtree substitution initially proposed in [Geier and Bercher, 2011] which replaces a subtree with another subtree.

Given a decomposition tree $(T, E, \Delta, \alpha, \beta)$ and a node $t \in T$, we define the subtree of Tr induced by t, as $\mathsf{Tr}[t] = (T', E', \Delta|_{T'}, \alpha|_{T'}, \beta|_{T'})$ where (T', E') is the subtree in (T, E) which is rooted in t.

Definition 6.13 (Subtree substitution). Let $\mathsf{Tr} = (T, E, \Delta, \alpha, \beta)$ be a decomposition tree and $t_i, t_j \in T$ be two nodes of Tr where t_i is an ancestor of t_j. We define the result of the subtree substitution on Tr that substitutes t_i by t_j, written $\mathsf{Tr}[t_i \leftarrow t_j] = (T', E', \Delta', \alpha|_{T'}, \beta|_{T'})$, where

$$T' = (T \setminus \tau(\mathsf{Tr}[t_i])) \cup \tau(\mathsf{Tr}[t_j])$$

$$
\begin{aligned}
E' &= E|_{T'} \cup \{(p, t_j) \mid (p, t_i) \in E\} \\
\Delta' &= (\Delta|_{T'} \setminus \{(\mathsf{nil}, \varphi_1, *t_j), (t_j*, \varphi_2, \mathsf{nil}) \in \Delta\}) \\
&\qquad \cup \{(t_j, \varphi_1, t_1), (t_j*, \varphi_1, t_1) \mid (t_i, \varphi_1, t_1) \in \Delta\} \\
&\qquad \cup \{(t_2, \varphi_2, t_j), (t_2, \varphi_2, *t_j) \mid (t_2, \varphi_2, t_i) \in \Delta\}
\end{aligned}
$$

Different from the decomposition tree in [Geier and Bercher, 2011], the decomposition tree defined in this thesis, the constraints about a compound task are not propagated to its subtasks. So, the constraints about the starting and terminating tasks corresponding to t_j need to be dropped, because they are generated from t_j's parent in the original tree. The following proposition states that the resulting tree is still valid.

Proposition 6.6. *Let* $\mathsf{Tr} = (T, E, \Delta, \alpha, \beta)$ *be a valid decomposition tree with respect to TIHTNS problem* $\mathcal{P}$ *and two nodes* $t_i \in T, t_j \in \tau(\mathsf{Tr}[t_i])$ *with* $\alpha(t_i) = \alpha(t_j)$. *Then* $\mathsf{Tr}[t_i \leftarrow t_j]$ *is also a valid decomposition tree w.r.t.* $\mathcal{P}$.

Proof. Suppose $\mathsf{Tr}' = \mathsf{Tr}[t_i \leftarrow t_j] = (T', E', \Delta', \alpha', \beta')$. We first show that (T', E') is still a tree whose root is still t_0 by the definition of the decomposition tree (Definition 6.7) Next we prove that Tr' satisfies the criteria in the definition of the valid decomposition tree (Definition 6.8).

As $\alpha' = \alpha|_{T'}$ and $\beta' = \beta|_{T'}$, we have $\alpha' \subseteq \alpha$ and $\beta' \subseteq \beta$. Then criterion 1 holds because $\alpha(t_i) = \alpha(t_j)$.

Suppose p is the parent of t_i in Tr.

For criterion 2, we consider two cases: $t = p$ and $t \neq p$. If $t = p$, no matter whether $p = t_0$ or not, $\mathsf{ch}(\mathsf{Tr}', t)$ differs from $\mathsf{ch}(\mathsf{Tr}, t)$ in t_j instead of t_i because $(p, t_j) \in E'$. According to the definition of the valid decomposition tree (Definition 6.8) those constraints about t_i are also with respect to t_j. So, the task network induced by $\mathsf{sub}(t)$ in Tr' is also isomorphic to tn_m of p. Also, if $p \neq t_0$, we have $(t_j, \top, t*)$ and $(*t, \top, t_j)$. Thus, when $t = p$, criterion 2 holds. For the case $t \neq p$, if t is in the subtree $\mathsf{Tr}[t_j]$, criterion 2 holds because $\mathsf{Tr}[t_j]$ is a subtree of a valid decomposition tree Tr. As the substitution is in $\mathsf{Tr}[t_i]$, it does not change other nodes in Tr except for p or those nodes in $\mathsf{Tr}[t_j]$ and they still satisfy criterion 2.

For criterion 3, we know that for all (t_i, φ_1, t_1), $(t_2, \varphi_2, t_i) \in \Delta$, t_1 and t_2 must be in $\mathsf{ch}(\mathsf{Tr}, p)$. By the definition of the valid decomposition tree (Definition 6.8) if $(t_i, \varphi_1, t_1) \in \Delta$, then (t_j, φ_1, t_1), $(t_j*, \varphi_1, t_1) \in \Delta'$. It also holds for the case $(t_2, \varphi_2, t_i) \in \Delta$. Thus, for t_j criterion 3 holds. In addition, if $(t_i, \varphi_1, *t_1) \in \Delta$, we have $(t_j, \varphi_1, *t_1) \in \Delta'$ and if $(t_2*, \varphi_2, t_i) \in \Delta$ we have $(t_2*, \varphi_2, t_j) \in \Delta'$. Therefore, all children of p satisfy criterion 3. As $\mathsf{Tr}[t_j]$ is a subtree of a valid decomposition tree, criterion 3 holds for all nodes of Tr'.

For criterion 4, we need to show that all constraints in Δ' are demanded by criterion 2 and 3. We first consider two cases: $t = p$ and $t \neq p$. If $t \neq p$, the constraints $(\mathsf{nil}, \varphi_1, *t_j)$, $(t_j*, \varphi_2, \mathsf{nil}) \in \Delta$ are demanded by the decomposition method of the parent of t_j in Tr and they are removed in the Tr'. Other constraints in $\Delta|_{T'}$ necessary for criterion 2 and 3 with respect to Tr are still necessary for criterion 2 and 3 with respect to Tr'. When $t = p$, those constraints $(t_j, \varphi_1, t_1), (t_2, \varphi_2, t_j) \in \Delta'$ are necessary due to being isomorphic with the subtask network tn_m; $(t_j*, \top, t_1), (t_2, \top, *t_j) \in \Delta'$ are necessary due to the starting and ending tasks of t_j which is also a condition in criterion 2; and $(t_j*, \varphi_1, t_1), (t_2, \varphi_2, *t_j) \in \Delta'$ are necessary due to criterion 3. □

Now we define the notion of acyclic decomposition sequence. The sequence of decompositions is *acyclic* if for every node t in its corresponding decomposition tree Tr, the ancestors of t have different actions. We also say the decomposition tree Tr is acyclic.

The insertion of tasks allows us to break the loop of generating same compound tasks during the decomposition procedure and find a shortcut to generate the solution. The idea of eliminating the loop by subtree substitution and task insertion is shown in 6.1.

The following theorem states that we only need to consider the acyclic sequences of decompositions to compute the solution.

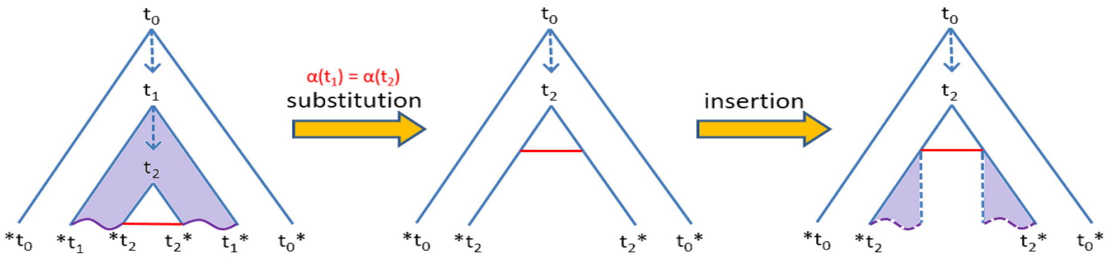

FIGURE 6.1: Elimination of Recursion by Subtree Substitution and Insertion

Theorem 6.7. *A TIHTNS problem $\mathcal{P}$ has a solution iff $\mathcal{P}$ has a solution which is generated by an acyclic sequence of decompositions and an insertion.*

Proof. The right-to-left direction is straightforward. Now we prove the left-to-right direction.

Suppose σ_{tn,s_I} is a TIHTNS solution of $\mathcal{P}$ where $\mathsf{tn}_I \to_D^* \mathsf{tn}_1$ and tn is obtained from tn_1 by insertion. Then tn_1 is a primitive task network. If the sequence of decompositions from tn_I to tn_1 is acyclic, then it is proved.

Now we suppose the sequence is not acyclic. Then in its corresponding tree Tr, there exist two ancestors t_i, t_j of some t such that $\alpha(t_i) = \alpha(t_j)$. By Proposition 6.6, the tree $\mathsf{Tr}[t_i \leftarrow t_j]$ is valid with respect to $\mathcal{P}$. By Proposition 6.1, there exists a primitive task network tn_2 such that $\mathsf{tn}_2 = \vartheta(\mathsf{Tr}[t_i \leftarrow t_j])$ and $\mathsf{tn}_I \to_D^* \mathsf{tn}_2$. As the substitution does not introduce any new node, the leaf nodes of $\mathsf{Tr}[t_i \leftarrow t_j]$ are a subset of the leaf nodes of Tr. The only state constraints about leaf nodes which are introduced by the substitution are (t_j*, φ_1, t_1) and $(*t_j, \varphi_2, t_2)$. Now we replace all occurrences of t_j* with t_i*, $*t_j$ with $*t_i$ in $\mathsf{Tr}[t_i \leftarrow t_j]$. Then except for those constraints between subtasks of t_j and either t_j* or $*t_j$, in form of $(*t_j, \top, t)$ and $(t, \top, t_j*)$, the constraints in tn_2 are a subset of the constraints of tn_1. Because t_i is an ancestor of t_j in Tr, there must be some tasks $t', t'', \ldots, t^{(k)}$such that $(*t_i, \top, t'), (t', \top, t''), \ldots, (t^{(k)}, \top, *t_j)$. Thus, in the plan σ_{tn,s_I}, $*t_i$ is before $*t_j$ and then before all subtasks of t_j. Then those constraints $(*t_j, \top, t)$ are satisfied; the case of t_j* is similar. So σ_{tn,s_I} includes all tasks in tn_2 and is consistent with all constraints in tn_2 and then tn can be obtained from tn_2 by inserting an appropriate task network. □

6.4 Decidability and Complexity

The complexity of the plan-existence problem for HTN planning is undecidable even for propositional HTN planning [Erol et al., 1994a]. With different restrictions, the complexity for HTN planning ranges from **PSPACE** to 2-**NEXPTIME**, as shown in [Alford et al., 2015a].

In the above section, we show that the solutions of TIHTNS can be obtained by acyclic decomposition. Based on it, we in this section show that the plan-existence problem of TIHTNS is decidable and the introduction of state constraints does not increase the complexity, staying 2-**NEXPTIME**-complete.

Theorem 6.7 allows us to only check the solutions generated by an acyclic decomposition sequence for the plan-existence problem. Next we show that the tasks generated by an acyclic decomposition sequence are finite.

By the grounding way introduced above, we know that the cardinality of ground atoms $\mathsf{Atm}(\mathscr{L}_0)$ depends on their arity and the number of constants and predicates. Let c and p be the number of constants and predicates in $\mathscr{L}$ respectively and n be the maximal arity of any predicate, then we have $|\mathsf{Atm}(\mathscr{L}_0)| \leq p \times c^n$. Similarly, the cardinality of ground high-level actions, which are syntactically atomic formulas, also depends on their arity and the number of constants in $\mathscr{L}$, apart from the cardinality of high-level actions. More precisely, $|C| \leq |\mathcal{C}| \times c^m$, where c is the number of constants and m is the maximal arity of any high-level action.

Lemma 6.8. *Given a TIHTNS problem* $\mathcal{P} = (\mathscr{L}, \mathcal{C}, \mathcal{O}, \mathcal{D}, \mathcal{M}, s_I, \mathsf{tn}_I)$*, for any acyclic valid decomposition tree* Tr *with respect to* $\mathcal{P}$*, the number of the tasks in the leaf task network is bounded by* $(k+2)^{(|\mathcal{C}| \times c^m)+1}$*, where* k *is the maximal number of subtasks in any decomposition method in* $\mathcal{M}$ *and the initial tasks* T_I*.*

Proof. Because there is no recursion in Tr, it entails that in the path from the root t_0 to a leaf node, different tasks correspond to different high-level actions. That is, the depth of Tr is at most $|C| + 1$ where $|C| \leq |\mathcal{C}| \times c^m$ and 1 is because of the unique high-level action c_{top}. In the decomposition tree Tr, every inner node has at most $k+2$ children nodes which include the starting and ending tasks $*t$ and $t*$. Thus, Tr has at most $(k+2)^{(|\mathcal{C}|\times c^m)+1}$ leaf nodes. $\square$

Besides the tasks obtained by acyclic decompositions are finite, the next lemma shows that the number of the inserted tasks between two tasks obtained by decomposition is also finite.

Lemma 6.9. *Suppose the TIHTNS problem $\mathcal{P} = (\mathfrak{D}, s_I, \mathsf{tn}_I)$ has a plan σ_{tn,s_I} and Tr is its decomposition tree, then $\mathcal{P}$ has a plan with a length bounded by $|T| \times 2^{p\times c^n}$ where T is the task set in leaf task network $\vartheta(\mathsf{Tr})$.*

Proof. Suppose two primitive tasks t_i and t_j in T which are neighbors in the sequence σ_{tn,s_I}, regardless of tasks not in T. The plan σ_{tn,s_I} defines a sequence of states starting from s_I and we use σ_s to denote this state sequence. As the number of states is bounded by $2^{|\mathsf{Atm}(\mathscr{L}_0)|}$, if there are more than $2^{|\mathsf{Atm}(\mathscr{L}_0)|}$ tasks inserted between t_i and t_j in σ_{tn,s_I}, there must be one state being visited twice and hence σ_s must contain a cycle of states. By removing all states in the cycle, we can form a new sequence of primitive tasks from σ_{tn,s_I} where there are at most $2^{|\mathsf{Atm}(\mathscr{L}_0)|}$ tasks inserted between t_i and t_j. The new sequence of primitive tasks is still executable in the state s_I, entailing that it is still a solution. By eliminating all cycles, there is a solution with a length of at most $|T| \times 2^{|\mathsf{Atm}(\mathscr{L}_0)|}$ where $|\mathsf{Atm}(\mathscr{L}_0)| \leq p \times c^n$. $\square$

With the above two lemmas, we only need to check whether there is a solution with an upper bound on length for the plan-existence problem.

Proposition 6.10. *If TIHTNS problem $\mathcal{P} = (\mathscr{L}, \mathcal{C}, \mathcal{O}, \mathcal{D}, \mathcal{M}, s_I, \mathsf{tn}_I)$ has a solution then $\mathcal{P}$ also has a solution with a length of at most $(k+2)^{(|\mathcal{C}|\times c^m)+1} \times 2^{p\times c^n}$.*

Proof. By Lemma 6.8, an acyclic valid decomposition tree Tr has only at most $(k+2)^{|C|+1}$ leaf nodes. By Lemma 6.9, there exists a plan of $\mathcal{P}$ with a length of at most $(k+2)^{|C|+1} \times 2^{|\mathsf{Atm}(\mathscr{L}_0)|}$ where $|\mathsf{Atm}(\mathscr{L}_0)| \leq p \times c^n$ and $|C| \leq |\mathcal{C}| \times c^m$. $\square$

Up to now, we have shown that checking solutions reduces to checking finite solutions. Next we introduce an acyclic progression operator to find the solution.

Before define the progression operator, we first define the regression of the basic action. As the state-transition function γ satisfies the frame axiom, before performing a basic action we can check whether a formula is true in the next state. Given a formula φ and a basic action o, we use φ_o^- to denote the formula obtained from φ by replacing p with $\top$ and q with $\bot$ for all $p \in \texttt{eff}^+(o)$ and $q \in \texttt{eff}^-(o)$. As all variables outside the effect of basic action o keep their truth value, we can check whether a formula φ holds in the next state before basic action o is performed, as the following lemma states.

Lemma 6.11. *$s \models \varphi_o^-$ iff $\gamma(s, o) \models \varphi$.*

Proof. Suppose φ and φ_o^- are in negation normal form (NNF) where all negation connectives $\neg$ are only applied before atoms and only conjunction connectives $\wedge$ and disjunction connectives $\vee$ are allowed. According to [Robinson and Voronkov, 2001], every propositional formula has an equivalent formula in NNF.

We prove the lemma by induction in the construction of ground formula.

Base case. If φ is a positive ground literal p, there are three cases: $p \in \texttt{eff}^+(o)$, $p \in \texttt{eff}^-(o)$ and $p \notin \texttt{eff}^+(o) \cup \texttt{eff}^-(o)$. Then there are three cases respectively for formula φ_o^-: $\top$, $\bot$ and p. For every ground atom q in $\mathsf{Atm}(\mathscr{L}_0)$, we have $s \models q$ iff $q \in s$.

- In the case of $p \in \texttt{eff}^+(o)$, we have $p \in \gamma(s, o)$ and then $\gamma(s, o) \models p$. So, $s \models \top$ iff $\gamma(s, o) \models p$.
- In the case of $p \in \texttt{eff}^-(o)$, we have $p \notin \gamma(s, o)$ and then $\gamma(s, o) \not\models p$. So, $s \models \bot$ iff $\gamma(s, o) \models p$.
- In the case of $p \notin \texttt{eff}^+(o) \cup \texttt{eff}^-(o)$, $p \in s$ iff $p \in \gamma(s, o)$. So, $s \models p$ iff $\gamma(s, o) \models p$.

Now let us consider φ is a negative ground literal $\neg p$. Similarly, there are three cases: $p \in \texttt{eff}^+(o)$, $p \in \texttt{eff}^-(o)$ and $p \notin \texttt{eff}^+(o) \cup \texttt{eff}^-(o)$. Then there are still three cases respectively for formula φ_o^-: $\bot$, $\top$ and $\neg p$.

- In the case of $p \in \texttt{eff}^+(o)$, we have $p \in \gamma(s,o)$ and then $\gamma(s,o) \models p$. Thus, $\gamma(s,o) \not\models \neg p$. So, $s \models \bot$ iff $\gamma(s,o) \models \neg p$.

- In the case of $p \in \texttt{eff}^-(o)$, we have $p \notin \gamma(s,o)$ and then $\gamma(s,o) \models \neg p$. So, $s \models \top$ iff $\gamma(s,o) \models \neg p$.

- In the case of $p \notin \texttt{eff}^+(o) \cup \texttt{eff}^-(o)$, $p \notin s$ iff $p \notin \gamma(s,o)$. So, $s \models \neg p$ iff $\gamma(s,o) \models \neg p$.

Therefore, if φ is a ground literal, we have $s \models \varphi_o^-$ iff $\gamma(s,o) \models \varphi$.

Inductive step. Consider the forms of formula: $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$.

In the case of $\varphi_1 \wedge \varphi_2$. As an inductive step, we suppose that $s \models (\varphi_1)_o^-$ iff $\gamma(s,o) \models \varphi_1$ and that $s \models (\varphi_2)_o^-$ iff $\gamma(s,o) \models \varphi_2$. Suppose $s \models (\varphi_1)_o^- \wedge (\varphi_2)_o^-$. Then we have $\gamma(s,o) \models \varphi_1 \wedge \varphi_2$. As $(\varphi_1 \wedge \varphi_2)_o^- = (\varphi_1)_o^- \wedge (\varphi_2)_o^-$, $s \models \varphi_o^-$ entails $\gamma(s,o) \models \varphi$. Vice verses. So, $s \models \varphi_o^-$ iff $\gamma(s,o) \models \varphi$.

In the case of $\varphi_1 \vee \varphi_2$. As an inductive step, we suppose that $s \models (\varphi_1)_o^-$ iff $\gamma(s,o) \models \varphi_1$ and that $s \models (\varphi_2)_o^-$ iff $\gamma(s,o) \models \varphi_2$. Suppose $s \models (\varphi_1)_o^- \vee (\varphi_2)_o^-$. Then we have $s \models (\varphi_1)_o^-$ or $s \models (\varphi_2)_o^-$. Thus, we have $\gamma(s,o) \models \varphi_1$ or $\gamma(s,o) \models \varphi_2$. By the semantics of propositional logic, we have $\gamma(s,o) \models \varphi_1 \vee \varphi_2$. As $(\varphi_1 \vee \varphi_2)_o^- = (\varphi_1)_o^- \vee (\varphi_2)_o^-$, $s \models \varphi_o^-$ entails $\gamma(s,o) \models \varphi$. Vice verses. So, $s \models \varphi_o^-$ iff $\gamma(s,o) \models \varphi$. □

Now we adapt the acyclic progression operator proposed in [Alford et al., 2015b] to TIHTNS planning.

To capture the 'maintenance' state constraints we introduce a set Σ of pairs (φ, t) of a formula and a task which means φ should be satisfied until performing t. We use $\mathsf{Fml}(\Sigma)$ to denote the conjunction of all formulas in Σ where the identical formulas only occur once. So, by checking whether $(\mathsf{Fml}(\Sigma))_o^-$ holds at the current state s, we know whether constraints are going to hold in the next state after performing o. To forbid the recursion of tasks, when decomposing a compound task t by decomposition method m, its subtasks cannot contain t's ancestors denoted by function $\mathsf{h}(t)$, where $\mathsf{h} : T \longrightarrow 2^C$. We say a task t in task network $\mathsf{tn} = (T, \Delta, \alpha)$ is *unconstraint* if there is no t' such that $t' \neq \mathsf{nil}$ and $(t', \varphi, t) \in \Delta$.

We define an acyclic progression operator for TIHTNS planning as a procedure that performs progression on the current state. Formally, we use a tuple $(s, \mathsf{tn}, \Sigma, \mathsf{h})$ to represent that task network tn still needs to achieve at the current state s.

Definition 6.14 (**Acyclic progression**). Given a tuple $(s, \mathsf{tn}, \Sigma, \mathsf{h})$, we define its acyclic progression is $(s', \mathsf{tn}', \Sigma', \mathsf{h}')$ with $\mathsf{tn}' = (T', \Delta', \alpha')$, which is obtained from three possible options:

- Task insertion: if $s \models \mathtt{pre}(o) \wedge (\mathsf{Fml}(\Sigma))^-_o$ then
  - $s' = \gamma(s, o)$
  - $\mathsf{tn}' = \mathsf{tn}, \Sigma' = \Sigma, \mathsf{h}' = \mathsf{h}$

  The inserted task should satisfy the state constraints.

- Task performance: for a primitive task $t \in T$, if t is unconstraint and $s \models \mathtt{pre}(\alpha(t))$ and the following hold:
  - $s \models (\mathsf{Fml}(\Sigma_{-t}))^-_{\alpha(t)}$ where $\Sigma_{-t} = \Sigma \setminus (\mathscr{L} \times \{t, \bar{t}\})$ where $\bar{t}$ is the original compound task if t is its starting task, i.e., if $t = *t_j$, then $\bar{t} = t_j$
  - for every $(\mathsf{nil}, \varphi, t) \in \Delta$, $s \models \varphi$
  - for every $(t, \varphi, t') \in \Delta$, $s \models \varphi^-_{\alpha(t)}$

  then

  - $T' = T \setminus \{t\}$, $\Delta' = \Delta|_{T'}$, $\alpha' = \alpha|_{T'}$
  - $\Sigma' = \Sigma_{-t} \cup \{(\varphi, t') \mid (t, \varphi, t') \in \Delta \text{ and } t' \neq \mathsf{nil}\}$
  - $\mathsf{h}' = \mathsf{h}|_{T'}$ and $s' = \gamma(s, \alpha(t))$

  A primitive task is chosen only if its all predecessors have been accomplished, its precondition is satisfied and the state constraints in Σ, except for those constraints ending with the primitive task, will hold after performing it. The 'maintenance' state constraints starting from the primitive task will be added into Σ.

- Task decomposition: for a compound task $t \in T$ and a decomposition method $m = (\alpha(t), (T_m, \Delta_m, \alpha_m)) \in M$, if $\mathsf{h}(t) \cap T_m = \emptyset$, then $\mathsf{tn} \underset{t,m}{\longrightarrow} \mathsf{tn}'$ and
  - $\mathsf{h}' = \mathsf{h}|_{T'} \cup \{(t_m, \mathsf{h}(t) \cup \{\alpha(t)\}) \mid t_m \in T_m\}$
  - $\Sigma' = \Sigma|_{T'} \cup \{(\varphi, *t) \mid (\varphi, t) \in \Sigma\}$

  For a compound task, the decomposition method chosen to decomposed cannot contain an action which is its ancestor in order to avoid the recursion of tasks.

Note that the cardinality of Σ may be exponential on size but $\mathsf{Fml}(\Sigma)$ is polynomial because $\mathsf{Fml}(\Sigma)$ consists of the formulas of the constraints in M. It entails that applying a step of acyclic progression is in $\mathbf{P}$.

Notably, acyclic progression is only acyclic over decompositions, not states reached. That is, it is possible that a state is visited more than one time. Actually, in the sequence of applying progression, the decomposition operators commute with the operators of task performance and task insertion, as the following lemma states.

Lemma 6.12. *If $(s, \mathsf{tn}, \Sigma, h)$ is obtained from $(s_I, \mathsf{tn}_I, \emptyset, \mathsf{h}_I)$ by applying acyclic progressions, then there exists a tuple $\chi = (s_I, \mathsf{tn}', \Sigma', h')$ such that there exist a sequence of decomposition operators from $(s_I, \mathsf{tn}_I, \emptyset, \mathsf{h}_I)$ to χ and a sequence of task performance and insertion from χ to $(s, \mathsf{tn}, \Sigma, h)$.*

Proof. Suppose the sequence of acyclic progressions applied from $(s_I, \mathsf{tn}_I, \emptyset, \mathsf{h}_I)$ to $(s, \mathsf{tn}, \Sigma, h)$ is σ_{AP}. Because for every compound task t, all constraints (t', φ, t) are propagated to $(t', \varphi, *t)$ and in every operator of task decomposition $(\varphi, *t)$ are added into Σ', the decomposition operators in the sequence σ_{AP} can put ahead of the insertion and performance operators. When performing $*t$, the constraint $(\varphi, *t)$ is not in Σ_{-*t}, neither is (φ, t). In other words, the formula φ holds until t if and only if φ holds until its start task $*t$, even though t has been decomposed and removed. Thus, there is a sequence σ'_{AP} from $(s_I, \mathsf{tn}_I, \emptyset, \mathsf{h}_I)$ to $(s_I, \mathsf{tn}', \Sigma', \mathsf{h}')$ and then to $(s, \mathsf{tn}_\emptyset, \emptyset, \emptyset)$ where tn' is a primitive task network. The sequence σ'_{AP} is partitioned into two subsequences: the former subsequence only contains decomposition operators and the latter subsequence only contains operators of insertion and performing. □

The task network in the tuple includes the tasks needs to accomplish and a step of acyclic progression may eliminate a task in the task network. If all tasks are eliminated by the acyclic progression, then the TIHTNS problem has a solution:

Proposition 6.13. *Given a ground TIHTNS problem $\mathcal{P}^0 = (\mathfrak{D}^0, s_I, \mathsf{tn}_I)$, $\mathcal{P}^0$ has a solution iff there is a sequence of acyclic progressions from $(s_I, \mathsf{tn}_I, \emptyset, \mathsf{h}_I)$ to $(s, \mathsf{tn}_\emptyset, \emptyset, \emptyset)$ where $\mathsf{h}_I = \alpha_I$ and $\mathsf{tn}_\emptyset$ is the empty task network.*

Proof. "$\Leftarrow$:" If $\mathcal{P}^0$ has a solution, by Theorem 6.7, there exists a solution σ_{tn,s_I} where tn is generated by an acyclic decomposition sequence and an insertion from tn_I. Suppose Tr is the valid decomposition tree corresponding to σ_{tn,s_I}. Then Tr is acyclic. As function h models the acyclic property of the decomposition tree, we can applied task decompositions according to Tr and generate a sequence of acyclic progressions from $(s_I, \mathsf{tn}_I, \emptyset, \mathsf{h}_I)$ to $(s_I, \vartheta(\mathsf{Tr}), \Sigma, \mathsf{h})$.

By Proposition 6.3, all constraints in Tr are satisfied in σ_{tn,s_I}. Then we can apply the operators of task insertion and task performance according to the order in σ_{tn,s_I} on the tuple $(s_I, \vartheta(\mathsf{Tr}), \Sigma, \mathsf{h})$. As all primitive tasks in $\vartheta(\mathsf{Tr})$ are also included in σ_{tn,s_I} and every step of task performance removes a task t in $\vartheta(\mathsf{Tr})$ and the pairs corresponding to t in Σ and h, we finally obtain a tuple $(s, \mathsf{tn}_\emptyset, \emptyset, \emptyset)$.

"$\Rightarrow$:" Suppose there is a sequence σ_{AP} of acyclic progressions from $(s_I, \mathsf{tn}_I, \emptyset, \mathsf{h}_I)$ to $(s, \mathsf{tn}_\emptyset, \emptyset, \emptyset)$. By Lemma 6.12, there is a sequence σ'_{AP} from $(s_I, \mathsf{tn}_I, \emptyset, \mathsf{h}_I)$ to $(s_I, \mathsf{tn}', \Sigma', \mathsf{h}')$ and then to $(s, \mathsf{tn}_\emptyset, \emptyset, \emptyset)$ where tn' is a primitive task network and σ'_{AP} is partitioned two subsequence: the former subsequence σ'_D only contains decomposition operators and the latter subsequence σ'_{PI} only contains operators of insertion and performing.

Since task performance only removes primitive tasks, σ'_D is a sequence of decomposition from the initial task network tn_I to the primitive task network tn'. Function h guarantees that σ_D is acyclic. As for every step of insertions and performances, the constraints in Σ are satisfied in the state and the next state, the sequence σ of basic actions corresponding to σ'_{PI} is consistent with the constraint set in tn'. As σ'_{PI} starts from the tuple $(s_I, \mathsf{tn}', \Sigma', \mathsf{h}')$, the basic action sequence σ is executable in the initial state s_I. Therefore, σ is a solution of $\mathcal{P}^0$. □

The next theorem states the upper bound of the complexity of the plan-existence problem for TIHTNS planning.

Theorem 6.14. *Deciding whether a propositional TIHTNS planning problem has a solution is in* ***NEXPTIME****.*

Proof. Theorem 6.7 reduces deciding the problem into checking all solutions generated by acyclic decomposition and Proposition 6.10 reduces further deciding the problem into checking solutions with an upper bound on length of $(k+2)^{|C|+1} \times 2^{\mathsf{Atm}(\mathscr{L}_0)}$. By introducing a counter on basic actions according to [Alford et al., 2015b], the length of a solution can be restricted with a polynomial translation on the problem. Additionally because the number of decomposing compound tasks is bounded by $k^{|C|}$, every sequence of acyclic progression must terminate after an exponential number of steps of insertions, performings and decompositions. Therefore, by a non-deterministic application of acyclic progressions with an exponential number, either it can reach a solution or it cannot progress any more which entails that the problem has no solution. □

Theorem 6.15. *Deciding whether a TIHTNS planning problem has a solution is in 2-****NEXPTIME****.*

Proof. As the solutions are bounded by $(k+2)^{(|\mathcal{C}|\times c^m)+1} \times 2^{p\times c^n}$, Every sequence of acyclic progression must terminate after a double-exponential number of steps of insertions, performances and decompositions. So, by a non-deterministic application of acyclic progressions with a double-exponential number, either it can reach a solution or it cannot progress any more which entails that the problem has no solution. □

The upper bound of the complexity entails that deciding whether a TIHTNS problem has a solution is decidable.

Corollary 6.16. *The plan-existence problem for TIHTNS planning is decidable.*

The lower bound of the complexity for TIHTNS comes from lifted TIHTN planning, whose plan-existence problem is 2-**NEXPTIME**-complete [Alford et al., 2015b].

Let us now translate lifted TIHTN into our extended TIHTNS. By replacing all occurrences of $t_i \prec t_j$ in lifted TIHTN problem $\mathcal{P}$ with $(t_i, \top, t_j)$, we obtain a TIHTNS problem, noted $\Gamma_T(\mathcal{P})$. It is easy to check that the translation is polynomial.

The following proposition shows that the two problems are equivalent.

Proposition 6.17. *For a lifted TIHTN problem $\mathcal{P}$, a sequence of basic action σ is a solution of $\mathcal{P}$ iff σ is also a solution of $\Gamma_T(\mathcal{P})$.*

Proof. Under the lifted TIHTN semantics, consider the decomposition of t by m, the new ordering constraint set is obtained as:

(1) keep the constraints not involving t;

(2) introduce the constraints about the subtasks;

(3) if t is before t' then all subtasks are before t';

(4) if t is after t' then all subtasks are after t'.

Under the extended semantics, according to Definition 6.6, the change of state constraints includes (1) and (2) and is analogous to (3) and (4). If t is before t', i.e., $(t, \top, t')$, there will be $(t*, \top, t')$ introduced. As $(t_m, \top, t*)$ for all subtasks t_m are introduced, it entails that $(t_m, \top, t')$ which means t_m is before t'. Then (3) is simulated and the case of (4) is similar. The proposition follows. □

The above translation tells us that every TIHTN problem can translated polynomially into an equivalent TIHTNS problem, entailing that the TIHTNS problem is at least as hard as TIHTN problems. With the lower bound obtained from TIHTN problem, we have the completeness of the complexity for TIHTNS planning:

Theorem 6.18. *Deciding whether a TIHTNS planning problem has a solution is 2-**NEXPTIME**-complete.*

Proof. As the translation Γ_T from lifted TIHTN problem to TIHTNS problem is polynomial and the plan-existence problem for lifted TIHTNS planning is 2-**NEXPTIME**-complete, the plan-existence problem for TIHTNS planning is 2-**NEXPTIME**-hard.

With the upper bound stated in Theorem 6.15, the plan-existence problem for TIHTNS planning is 2-**NEXPTIME**-complete. □

Initially, the complexity of plan-existence problem for propositional TIHTN planning was shown in **EXPSPACE** in [Geier and Bercher, 2011]. But the complexity has been tightened in **NEXPTIME**-complete in [Alford et al., 2015b], which actually provides a lower bound of the complexity for propositional TIHTNS planning:

Theorem 6.19. *Deciding whether a propositional TIHTNS planning problem has a solution is **NEXPTIME**-complete.*

Proof. It is proved by Theorem 6.14 and Theorem 6.18. □

6.5 Relation with GTN and HGN

Hierarchical planning approaches are often chosen for real world application scenarios, such as [Biundo et al., 2011, Lin et al., 2008], due to the ability to specify solution strategies in terms of decomposition methods, but also because human expert knowledge is often structured in a hierarchical way and can thus be smoothly integrated into HTN planning models. On the other side, these decomposition methods also make HTN planning less flexible than non-hierarchical approaches, because only those solutions may be generated that are "reachable" via the decomposition of the decomposition methods. Therefore, defining only a partially hierarchical domain is not sufficient to produce all desired solutions.

Recently HTN researchers work on enhancing the semantics of HTN planning and have proposed variants of HTN planning. Besides TIHTN planning, hierarchy goal network (HGN) planning [Shivashankar et al., 2012] operates over a hierarchy of goals with decomposition methods that decompose goals with further subgoals. [Alford et al., 2016] combined HTN and HGN planning into goal-task network (GTN) planning where an element of the network consists of a goal and a task. They also show that allowing task (goal) insertion, HGN and GTN planning problems can be translated polynomially into lifted TIHTN problems. Therefore, our extension TIHTNS actually also covers them.

In addition, [Shivashankar et al., 2017] relaxed the hierarchy of HGN planning and translate this variant which is called hierarchy-relaxed hierarchy goal network (HR-HGN) planning into classical planning. Hereafter, we detail that how lifted TIHTN and HR-HGN planning can be easily encoded in our framework in polynomial time.

Different from HTN planning, HGN planning talks about goal network $\mathsf{gn} = (G, \prec)$ where G is a set of formulas in disjunctive normal form over ground literals, called goal formulas, and $\prec \subseteq G \times G$ is a strict partial order on G. Just as HTN problems, an HGN problem is a tuple $\mathcal{P} = (\mathscr{L}, O, M, s_I, \mathsf{gn}_I)$ where M is a set of HGN decomposition methods and gn_I is an initial goal network.

For an HR-HGN problem, the decomposition method set M is empty. The solutions of HR-HGN are defined as the set of all basic action sequences that are executable in the initial state s_I and that achieve all initial goals according to the order.

In [Shivashankar et al., 2017], by introducing fresh basic actions with the number of $|\mathsf{gn}_I|$, an HR-HGN problem can be translated into a classical planning problem. Now, we translate HR-HGN planning into our framework without introducing any fresh basic actions. Allowing task insertion, state constraints simulate a goal formula by requiring that the formula holds immediately before an empty task. Formally, given an HR-HGN problem $\mathcal{P} = (\mathscr{L}, O, s_I, \mathsf{gn}_I)$, we define a TIHTNS problem $\Gamma_G(\mathcal{P}) = (\mathscr{L}, \emptyset, O, \emptyset, s_I, \mathsf{tn})$ as:

- for every $g \in G_I$, $\lambda_g \in \mathsf{tn}$ and $(\mathsf{nil}, g, \lambda_g) \in \Delta$ and $\alpha(\lambda_g) = \mathsf{skip}$
- for every $g_i \prec g_j$, $(\lambda_{g_i}, \top, \lambda_{g_j})$

Next we show that the translation Γ_G is correct.

Proposition 6.20. *For an HR-HGN problem $\mathcal{P}$, a sequence of basic actions σ is a solution of $\mathcal{P}$ iff σ is also a solution of $\Gamma_G(\mathcal{P})$.*

Proof. Suppose $\sigma_{\mathsf{tn}', s_I}$ is a solution of $\Gamma_G(\mathcal{P})$. Then tn' is obtained from tn by an insertion. It forms a sequence $s_0, \ldots, s_n$ where $s_0 = s_I$. For every $g \in G_I$, we have $\lambda_g \in T'$ and $(\mathsf{nil}, g, \lambda_g) \in \Delta'$ then $s_i \models g$ where $\sigma(\lambda_g) = i$; for every $g_i \prec g_j$, we have $(\lambda_{g_i}, \top, \lambda_{g_j})\Delta'$ then $i' < j'$ and $s_{i'} \models g_i, s_{j'} \models g_j$,where $\sigma(\lambda_{g_i})$ and $\sigma(\lambda_{g_j})$. $\square$

6.6 Summary

In this chapter, we extend TIHTN planning into TIHTNS planning so that state constraints can be captured. We also show that TIHTNS planning keeps the property of acyclic decomposition, entailing that the plan-existence problem is decidable. In addition, based on this property, we propose an acyclic progression operator for TIHTNS planning. With the progression operator, we show that the extension it does not increase the complexity of plan-existence, staying in 2-**NEXPTIME**. Finally, we investigate the relation between TIHTNS planning and other two kind of HTN-like planning formalism: TIHTNS planning can cover not only HR-HGN planning, but also GTN and HGN with allowing task (goal) insertion.

In PDDL3 [Gerevini and Long, 2005], the trajectory constraints are associated with time which are in form of formulas in linear temporal logic. Note that in the definition of the consistency with state constraints (Definition 6.9), the consistency is defined on the sequence of states. It actually paves the way for TIHTNS planning to model the linear temporal property of state constraints and then to capture trajectory constraints.

Generally, HTN planners solve problems either using decomposition directly, such as [Erol et al., 1994b] and [Bercher et al., 2014], or using progression [Nau et al., 2003]. Since progression-based HTN algorithms can be efficient across a number of syntactically identifiable classes of HTN problems [Alford et al., 2015a], we are convinced that our acyclic progression operator is a starting point for designing an efficient TIHTNS planner.

Chapter 7

Conclusion and Future Work

This thesis aims to provide a comprehensive analysis of intention refinement. In the last chapter, let's come back to the research questions and discuss what could be done in the future.

7.1 Summary

In this thesis, we analyze the refinement in two main frameworks: the belief-intention database and HTN planning. Let us start this section by answering the questions proposed in Chapter 1 on these two frameworks:

1. *How can we model high-level intentions?*

   A high-level intention is considered as a high-level action with a flexible duration in belief-intention database and as a compound task in HTN planning, respectively.

2. *How to model the refinement of a high-level intention?*

   Database Whether an intention can be refined into a set of intentions is based on the entailment relation defined over belief-intention databases.

   HTN The refinement relation is precompiled via decomposition methods.

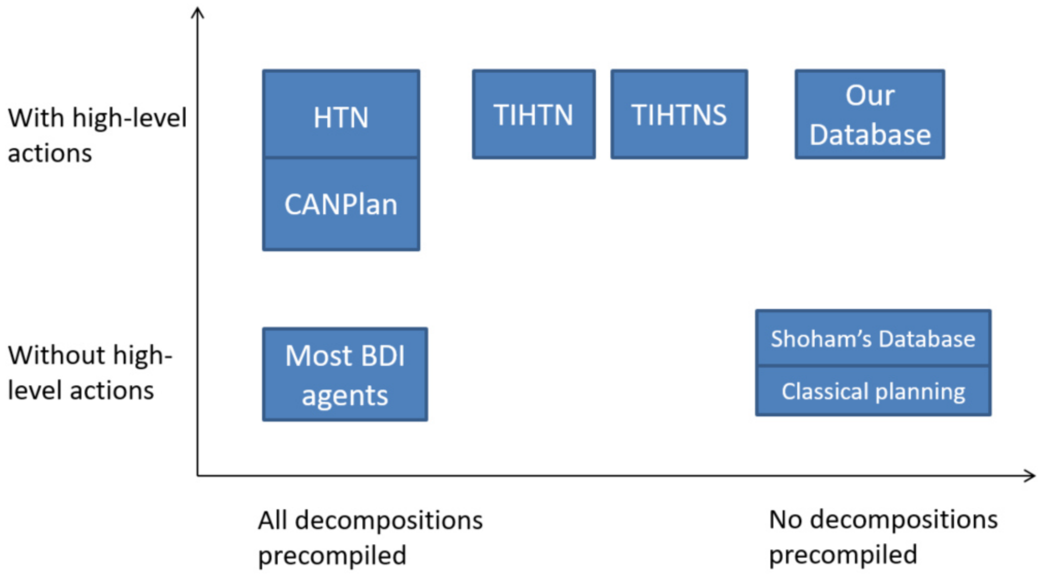

Figure 7.1: Comparison between BDI Theories and Automated Planning 3

3. *How to refine an intention into executable plans?*

   Database Every step of refining intentions expands the database, leading it to be more concrete. The process of refinement is finite and finally results in an executable plan.

   HTN Plans are built via step-wise refinement of high-level actions into lower-level actions in a top-down manner, according to the predefined decomposition methods.

4. *Is the refinement operation correct?*

   Database The refined intention and the lower-level intentions satisfy the instrumentality relation.

   HTN The refinement relation among actions is predefined in the decomposition methods and we evaluate the coherence condition for the domain via PDL.

Now we have a more complete picture on the frameworks concerning this thesis. By updating Figure 2.1, we compare them in Figure 7.1. In the figure, our belief-intention database framework lies above Shoham's database framework, because our database is an extension of Shoham's database by considering high-level actions. On the other hand, compared to TIHTN planning, the completeness of decomposition methods for TIHTNS planning is less demanding and for our belief-intention databases, it is not necessary to precompile decomposition methods. Interestingly, from HTN, TIHTN, TIHTNS to our database framework, the precompilation of decomposition methods required becomes less and less, while the complexity of the plan-existence or satisfiability problem tend to be lower, from undecidability to **PSPACE**-complete.

A number of challenges to establish the new generation of BDI theory are listed in [Herzig et al., 2017]. Apart from intention refinement, the issues of the frame problem, integration with automated planning and revision theory are on the list. For the frame problem, by introducing the STRIPS-like actions and events, it is addressed in the belief-intention database. To link belief-intention database and automated planning, we have uniformed the terminology about actions and used dynamic theories to syntactically describe the behaviors of actions in the presentation of both databases and HTN planning. Furthermore, a theory of intention revision should be based on the instrumentality relation which stems from refinement. We are convinced that our research on intention refinement leads to progress within the near future and it is our long term goal to tackle the new challenges which we may meet in the foundation of new BDI theories.

7.2 Going Further

7.2.1 Intention Revision

As emphasized by Bratman, intentions are high-level plans to which the agent is committed. In this thesis we have studied the "high-level" property, while the "committed" property concerns another significant concept – *intention revision*.

Being commitments, intentions are *stable* mental attitudes. Indeed, there are only two possible reasons to abandon an intention:

1. either it turns out to be impossible to satisfy,
2. or it is only instrumental for another, higher-level intention the agent is about to abandon.

Here is an example involving both processes: suppose I intend to take out a loan in order to buy an expensive house and learn that it has already been sold. The revision of my beliefs about the future should make me drop these two intentions: I first then abandon my high-level intention to buy that house because it cannot be achieved, and then my instrumental intention to borrow.

The proposed notion of instrumentality based on intention refinement paves the way for revision of intentions: when dropping a high-level intention, we also drop the lower-level intentions that are instrumental for it. Up to now, there is little work on linking intention revision with instrumentality, except for [Shapiro et al., 2012] which models intention revision by considering relations between a predefined library of plans and intentions. When it comes to revision theory, we have to mention the milestone of the belief revision theory, the AGM model [Alchourrón et al., 1985]. As mentioned in Chapter 2, [van Zee and Doder, 2016] proposed AGM-like revision postulates for beliefs and intentions. Although the instrumentality relation is not accounted and the frame problem is not addressed, we believe that our database framework, combined with them, provides a good starting point for an intention revision theory that is based on the instrumentality relation.

7.2.2 Improving HTN Domains

On one hand, the decomposition methods in HTN domains simplify the domain design by avoiding the need for the complete causal models which require the complete precondition and postcondition. On the other hand, the way to decompose high-level actions extremely depends on the expertise of the domain designer on the scenario, where the domain designer stipulates that certain actions will be performed "just because he says so." However, in most real-world domains, the designer may have the expertise for only some parts of the domains. For instance, it is possible that the non-achievement of a compound task results from the lack of some subtasks. This is probably caused by an omission: the designer forgot to consider the subtask in the decomposition method. Taking the example of Figure 1.1 in the introduction, the action of buying the flight ticket is a prerequisite of taking the plane to Melbourne and it is missed, as a subtask of the action of flying to Melbourne. Nevertheless, there is no HTN solution because the missing subtask is not in the task network and will never be produced, in consequence the initial logistics task cannot be accomplished. The approach of TIHTN planning indeed fills up the plans by adding the missing actions to accomplish the initial tasks.

Actually, we have already implement a TIHTN planner via ASP, shown in Appendix B. One benefit of implementing the TIHTN planner via ASP lies in the computation of all solutions, which include all possible insertion of basic actions. It enables to provides clues to complete decomposition methods by adding the inserted actions into the decomposition methods, and further to improve HTN domains. Preliminary experimental results show that if there exists at least a plan, the improvement obtained from the TIHTN planner speeds up problem solving.

As mentioned above, the human expertise, in many real-world scenarios, is only partial. It is necessary to help the HTN domain designer to improve the domain via finding potential refinement relation among actions. Consider refinement from the entailment perspective, in which the belief-intention database does, contributes to the discovery of the hidden refinement relation.

Appendix A

Publications

The following is a list of the published papers during my PhD study.

- Andreas Herzig, Laurent Perrussel, Zhanhao Xiao, and Dongmo Zhang. *Refinement of intentions.* In Proceedings the 15th European Conference on Logics in Artificial Intelligence (JELIA-16), 2016. *The results of this paper are contained in Chapter 3.*

- Andreas Herzig, Laurent Perrussel, and Zhanhao Xiao. *On hierarchical task networks.* In Proceedings the 15th European Conference on Logics in Artificial Intelligence (JELIA-16), 2016. *The results of this paper are contained in Chapter 5.*

- Andreas Herzig, Laurent Perrussel, Emiliano Lorini, and Zhanhao Xiao. *BDI logics for BDI architectures: old problems, new perspectives.* Küstliche Intelligenz (KI) Journal, 2016. *The part of results are contained Chapter 1.*

- Zhanhao Xiao, Andreas Herzig, Laurent Perrussel, Hai Wan, and Xiaoheng Su. *Hierarchical Task Network Planning with Task Insertion and State Constraints.* In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI-17), 2017. *The results of this paper are contained in Chapter 6.*

- Zhanhao Xiao, Andreas Herzig, Laurent Perrussel, and Dongmo Zhang. *Deciding Refinement via Propositional Linear Temporal Logic.* In Proceedings of the 16th International Conference of the Italian Association for Artificial Intelligence, 2017. *The results of this paper are contained in Section 3.5 and Section 4.1.*

Appendix B

A TIHTN Planner Based on ASP

In this appendix, we give a decomposition-first algorithm for TIHTN planning and encode it into ASP programs, implementing a TIHTN planner. In TIHTN planning, only ordering constraints are considered. Here we adjust the presentation way of TIHTNS planning in Chapter 6 by using a pair of an action and an integer to denote a task.

A task network $\mathsf{tn} = (T, \prec)$ is a tuple:

- $T \subseteq (\mathcal{C} \cup \mathcal{O}) \times \mathbb{N}$
- $\prec \subseteq T \times T$ is a set of ordering constraints over T

In a task network, every task is a pair of an action a and an integer identifier i. With the same action, different identifiers indicate different tasks. For a task $t = (a, i)$, we define $\alpha(t){=}a$. A task t is called *primitive* if $\alpha(t)$ is a basic action, otherwise it is called *compound*. A task network is called *primitive* iff it contains only primitive tasks.

High-level actions cannot be directly executed by the agent and each high-level action is decomposed into a task network according to decomposition methods. Each decomposition method $m{=}(c, \mathsf{tn}')$ consists of a high level action c and a task network tn' whose inner tasks are called subtasks.

The decomposition procedure for TIHTN planning are similar with that of TIHTNS planning. Every step of decomposition removes the decomposed task and adds the subtask network into the task network. The problem description and solution criterion for TIHTN planning are identical as those of TIHTNS planning.

B.1 Encoding HTN Problems and Operators

First we show how to translate a TIHTN problem into a set of ASP facts. As each step of applying decomposition generates a task network, we use the number `D` of the decompositions that were applied to identify the task network. More formally, we use an ASP atom `tn(A,I,D)` to denote a task `(A,I)` is in the task network `D`.

Without loss of generality, we only give the encoding for the propositional HTN problems. For a propositional HTN problem $\mathcal{P} = (\mathfrak{D}, s_I, (T_I, \prec_I))$, we define a set $\Pi_{\mathcal{P}}$ of ASP facts as follows:

- for every $o \in O$, `primitive(o)`.
- for every $c \in C$, `compound(c)`.
- for every $m_k = (c, (T_k, \prec_k)) \in M$, `method(c,k)`.
  - for every $(a, j) \in T_k$, `sub(c,k,a,j)`.
  - for every $(a1, j1) \prec_k (a2, j2)$, `mSucc(c,k,a1,j1,a2,j2)`.
- for every $p \in s_I$, `holds(p,0)`.
- for every $(a, i) \in T_I$, `tn(a,i,0)`.
- for every $(a1, i1) \prec_I (a2, i2)$, `succ(a1,i1,a2,i2,0)`.

Next we encode the state transition system and planning operators in ASP. To be simple, we suppose the precondition of each operator is in CNF and NNF. That is, every precondition can be rewritten as a conjunction of literals. In each state `S`, there is one and only one applicable basic action `O` performed. Due to negation as failure, each variable $\mathtt{P}$ is false in $\mathtt{S+1}$ by default unless either it is added in $\mathtt{S}$ or it already holds and is not deleted in $\mathtt{S}$. We use $\Pi_{\mathtt{O}}$ to denote the set of facts and formulas as follows:

For every $o \in O$ and $(\mathtt{pre}(o), \mathtt{eff}^+(o), \mathtt{eff}^-(o)) \in \mathcal{D}$,

- for every $p \in \mathtt{eff}^+(o)$, $\mathtt{add}(\mathtt{p}, \mathtt{o})$.
- for every $p \in \mathtt{eff}^-(o)$, $\mathtt{del}(\mathtt{p}, \mathtt{o})$.
- for $\mathtt{pre}(o) = p_1 \wedge ... \wedge p_n \wedge \neg q_1 \wedge ... \wedge \neg q_m$,
  $\mathtt{pre}(\mathtt{o},\mathtt{S})$:- $\mathtt{holds}(\mathtt{p_1},\mathtt{S})$, ..., $\mathtt{holds}(\mathtt{p_n},\mathtt{S})$, not $\mathtt{holds}(\mathtt{q_1},\mathtt{S})$,...,not $\mathtt{holds}(\mathtt{q_m},\mathtt{S})$.

and

$$\mathtt{1}\ \{\mathtt{do}(\mathtt{O}, \mathtt{S}) : \mathtt{operator}(\mathtt{O})\}\ \mathtt{1}\ \text{:- } \mathtt{state}(\mathtt{S}). \tag{B.1}$$

$$\text{:- not } \mathtt{pre}(\mathtt{O}, \mathtt{S}), \mathtt{do}(\mathtt{O}, \mathtt{S}). \tag{B.2}$$

$$\mathtt{holds}(\mathtt{P}, \mathtt{S+1})\ \text{:- } \mathtt{add}(\mathtt{P}, \mathtt{O}), \mathtt{do}(\mathtt{O}, \mathtt{S}). \tag{B.3}$$

$$\mathtt{holds}(\mathtt{P}, \mathtt{S+1})\ \text{:- not } \mathtt{del}(\mathtt{P}, \mathtt{O}), \mathtt{do}(\mathtt{O}, \mathtt{S}), \mathtt{holds}(\mathtt{P}, \mathtt{S}). \tag{B.4}$$

B.2 Acyclic Decomposition

In Chapter 6, we have proposed an acyclic progression operator on the current state and task network. Every step of progression updates the state or the task network. In the state s and task network $\mathtt{tn}$, there are three choices to progress: (1) insertion is to find an applicable basic action in s to do; (2) performing a primitive task t requires that t is unconstrained, i.e., all predecessors of t have already been performed and that t's precondition holds in s, and removes t from $\mathtt{tn}$; (3) selecting a compound task t and its decomposition method m to decompose requires that t is unconstrained and that m cannot contain a subtask st where $\alpha(st)$ is in $\mathsf{h}(t)$, i.e., $\alpha(st)$ is an ancestor of t, and updates $\mathtt{tn}$ and Function h. If all tasks are removed via the progression operator, then a plan is found.

Actually, it is not necessary to require the task to be unconstrained when decomposing it, because all constraints about it are propagated to its subtasks and the decomposition does not change the state. After removing the unconstraint requirement, the progression operator can still find the plan:

Lemma B.1. *If there is a sequence of acyclic progressions from* $(s_I, \mathtt{tn}_I)$ *to* $(s, \emptyset)$*, then there exists a primitive task network* $\mathtt{tn}'$ *such that* $\mathtt{tn}_I \rightarrow^*_{AD} \mathtt{tn}'$ *and there exists a sequence of task performance and insertion from* $(s_I, \mathtt{tn}')$ *to* $(s, \emptyset)$*.*

Intuitively, if a sequence of progression forms a plan, then by putting the steps of decompositions ahead of all steps of performance and insertion, it still forms the plan. By this lemma, we first acyclically decompose the initial task network into a primitive network; then serialize it to find a plan.

Next we show how to use ASP to describe the procedure of acyclic decompositions. According to [Geier and Bercher, 2011], the maximum number of tasks in the task network which is generated via acyclic decomposition is $k^{|C|}$, where k is the maximum number of tasks inside the task networks in decomposition methods and the initial task network and $|C|$ is the number of ground high-level actions. Thus, we use $\mathtt{decompNo}(\mathtt{D})$ to denote $\mathtt{D}$ is within the scope from 0 to $k^{|C|}$.

If there exists a compound task in the task network $\mathtt{D}$, then it needs to be decomposed.

$$\mathtt{todecomp}(\mathtt{D}) \text{ :- } \mathtt{tn}(\mathtt{C},\mathtt{I},\mathtt{D}), \mathtt{compound}(\mathtt{C}), \mathtt{decompNo}(\mathtt{D}). \tag{B.5}$$

Each decomposition is caused by one and only one decomposition method of compound task. Formula (B.6) states it is possible for every compound task to be selected but formula (B.7),(B.8) guarantee only one can be selected and formula (B.9),(B.10) guarantee there is at least one selected. Formula (B.11) means if $\mathtt{C},\mathtt{I}$ is selected to decompose, then there is one and only one decomposition method $\mathtt{C},\mathtt{K}$ selected. Formula (B.12) means every subtask $\mathtt{SA}$ is not an ancestor of $\mathtt{C},\mathtt{I}$

and guarantees the procedure of decompositions is acyclic.

$$0\ \{\mathtt{sel(C,I,D)}\}\ 1 \text{ :- } \mathtt{tn(C,I,D)}, \mathtt{compound(C)}. \tag{B.6}$$

$$\text{:- } \mathtt{sel(C1,I1,D)}, \mathtt{sel(C2,I2,D)}, \mathtt{C1}!\!=\!\mathtt{C2}. \tag{B.7}$$

$$\text{:- } \mathtt{sel(C1,I1,D)}, \mathtt{sel(C2,I2,D)}, \mathtt{I1}!\!=\!\mathtt{I2}. \tag{B.8}$$

$$\mathtt{sel(D)} \text{ :- } \mathtt{sel(C,I,D)}. \tag{B.9}$$

$$\text{:- not } \mathtt{sel(D)}, \mathtt{todecomp(D)}. \tag{B.10}$$

$$1\ \{\mathtt{selM(C,K,D)} : \mathtt{method(C,K)}\}\ 1 \text{ :- } \mathtt{sel(C,I,D)}. \tag{B.11}$$

$$\text{:- } \mathtt{selM(C,K,D)}, \mathtt{sub(C,K,SA,SI)}, \mathtt{h(C,I,SA)}. \tag{B.12}$$

We use $\mathtt{max(A,I,D)}$ to denote the maximum identifier $\mathtt{I}$ for action $\mathtt{A}$ in the task network $\mathtt{D}$. If an action is not in the task network, its maximum identifier is 0.

$$\mathtt{max(A,I,D)} \text{ :- } \mathtt{tn(A,I,D)}, \text{not } \mathtt{tn(A,I{+}1,D)}. \tag{B.13}$$

$$\mathtt{max(C,0,D)} \text{ :- not } \mathtt{tn(C,1,D)}, \mathtt{compound(C)}, \mathtt{decompNo(D)}. \tag{B.14}$$

$$\mathtt{max(O,0,D)} \text{ :- not } \mathtt{tn(O,1,D)}, \mathtt{primitive(O)}, \mathtt{decompNo(D)}. \tag{B.15}$$

Next we show how the decomposition results in the new task network. The new identifier of the introduced subtask $(\mathtt{SA},\mathtt{SI})$ is its maximum identifier $\mathtt{J}$ in task network $\mathtt{D}$ plus $\mathtt{SI}$, which guarantees every introduced subtask differs from the existing tasks. By negation as failure, those tasks which are not selected to decompose remain in the task network.

$$\mathtt{tn(SA,J{+}SI,D{+}1)} \text{ :- } \mathtt{selM(C,K,D)}, \mathtt{sub(C,K,SA,SI)}, \mathtt{max(SA,J,D)}. \tag{B.16}$$

$$\mathtt{tn(A,N,D{+}1)} \text{ :- } \mathtt{tn(A,N,D)}, \text{not } \mathtt{sel(A,N,D)}, \mathtt{todecomp(D)}. \tag{B.17}$$

Subtasks inherit the constraints about the selected compound task. Formally, for every subtask $(\mathtt{SA},\mathtt{SI})$ of decomposition method $(\mathtt{C},\mathtt{K})$, if $(\mathtt{A1},\mathtt{J1}) \prec (\mathtt{C},\mathtt{I})$, then $(\mathtt{A1},\mathtt{J1}) \prec (\mathtt{SA},\mathtt{J{+}SI})$; if $(\mathtt{C},\mathtt{I}) \prec (\mathtt{A2},\mathtt{J2})$, then $(\mathtt{SA},\mathtt{J{+}SI}) \prec (\mathtt{A2},\mathtt{J2})$; Also, the ordering constraints among subtasks are kept in the resulting task network. With negation as failure, formula (B.21) means the constraints unrelated to the selected task are preserved.

$$\begin{aligned}\texttt{succ}(\texttt{A1},\texttt{J1},\texttt{SA},\texttt{J}+\texttt{SI},\texttt{D}+1) &\coloneq \texttt{succ}(\texttt{A1},\texttt{J1},\texttt{C},\texttt{I},\texttt{D}),\\ &\texttt{sel}(\texttt{C},\texttt{I},\texttt{D}),\texttt{sub}(\texttt{C},\texttt{K},\texttt{SA},\texttt{SI}),\texttt{max}(\texttt{SA},\texttt{J},\texttt{D}).\end{aligned} \tag{B.18}$$

$$\begin{aligned}\texttt{succ}(\texttt{SA},\texttt{J}+\texttt{SI},\texttt{A2},\texttt{J2},\texttt{D}+1) &\coloneq \texttt{succ}(\texttt{C},\texttt{I},\texttt{A2},\texttt{J2},\texttt{D}),\\ &\texttt{sel}(\texttt{C},\texttt{I},\texttt{D}),\texttt{sub}(\texttt{C},\texttt{K},\texttt{SA},\texttt{SI}),\texttt{max}(\texttt{SA},\texttt{J},\texttt{D}).\end{aligned} \tag{B.19}$$

$$\begin{aligned}&\texttt{succ}(\texttt{SA1},\texttt{J1}+\texttt{SI1},\texttt{SA2},\texttt{J2}+\texttt{SI2},\texttt{D}+1) \coloneq \texttt{selM}(\texttt{C},\texttt{K},\texttt{D}),\\ &\quad\texttt{mSucc}(\texttt{C},\texttt{K},\texttt{SA1},\texttt{SI1},\texttt{SA2},\texttt{SI2}),\texttt{max}(\texttt{SA1},\texttt{J1},\texttt{D}),\texttt{max}(\texttt{SA2},\texttt{J2},\texttt{D}).\end{aligned} \tag{B.20}$$

$$\begin{aligned}\texttt{succ}(\texttt{A1},\texttt{J1},\texttt{A2},\texttt{J2},\texttt{D}+1) &\coloneq \texttt{succ}(\texttt{A1},\texttt{J1},\texttt{A2},\texttt{J2},\texttt{D}),\\ &\text{not }\texttt{sel}(\texttt{A1},\texttt{J1},\texttt{D}),\text{not }\texttt{sel}(\texttt{A2},\texttt{J2},\texttt{D}),\texttt{todecomp}(\texttt{D}).\end{aligned} \tag{B.21}$$

Next we show how to update Function $\texttt{h}$. Formula (B.22) states that every compound task takes itself as an ancestor. The ancestors $\texttt{A}$ of the decomposed task $(\texttt{C},\texttt{I})$, including itself, are also the ancestors of all its subtasks $(\texttt{SA},\texttt{J}+\texttt{SI})$.

$$\texttt{h}(\texttt{C},\texttt{I},\texttt{C}) \coloneq \texttt{tn}(\texttt{C},\texttt{I},\texttt{D}),\texttt{compound}(\texttt{C}). \tag{B.22}$$

$$\begin{aligned}\texttt{h}(\texttt{SA},\texttt{J}+\texttt{SI},\texttt{A}) &\coloneq \texttt{sel}(\texttt{C},\texttt{I},\texttt{D}),\texttt{selM}(\texttt{C},\texttt{K},\texttt{D}),\texttt{h}(\texttt{C},\texttt{I},\texttt{A}),\\ &\texttt{sub}(\texttt{C},\texttt{K},\texttt{SA},\texttt{SI}),\texttt{max}(\texttt{SA},\texttt{J},\texttt{D}).\end{aligned} \tag{B.23}$$

The termination condition for the decomposition procedure is that there is no compound task. When $\texttt{endDecomp}$ is generated, the primitive task network $\texttt{D}$ s.t. $\texttt{final}(\texttt{D})$ is obtained. Formula (B.26) restricts that it has to be obtained.

$$\texttt{final}(\texttt{D}) \coloneq \text{not }\texttt{todecomp}(\texttt{D}),\texttt{todecomp}(\texttt{D-1}). \tag{B.24}$$

$$\texttt{endDecomp} \coloneq \texttt{final}(\texttt{D}). \tag{B.25}$$

$$\coloneq \text{not }\texttt{endDecomp}. \tag{B.26}$$

We use $\Pi_{\texttt{De}}$ to denote the set of formulas (B.5)–(B.26) and it describes how to refine the initial task network into a primitive task network, as the following proposition states.

Proposition B.2. *For a TIHTN problem* $\mathcal{P} = (\mathfrak{D}, s_I, \mathtt{tn}_I)$, $\mathtt{tn}_I \rightarrow^*_{AD} \mathtt{tn}$ *where* tn *is primitive iff* tn *is obtained from an answer set of* $\Pi_{\mathcal{P}} \cup \Pi_{\mathtt{De}}$.

Sketch. As $\Pi_{\mathtt{De}}$ describes the acyclic procedure of decompositions, the task network D where final(D) holds is a primitive task network. Given the upper bound $k^{|C|}$ for D, $\Pi_{\mathcal{P}} \cup \Pi_{\mathtt{De}}$ has an answer set including endDecomp iff the decomposition procedure terminates. □

B.3 Inserting and Performing Tasks

After decomposing into a primitive task network, we need to generate a plan which covers all primitive tasks and satisfies the constraints. As shown in Algorithm 1, we give a decomposition-first algorithm for TIHTN planning: first find a primitive task to do whose predecessors have already been done and whose precondition holds; if there is no such a primitive task then we choose an applicable basic action to do repeatedly until we find such a primitive task. In Algorithm 1, we define Function *dotask* to record the state in which every primitive task is done and Function *uncons* to search the unconstrained primitive tasks in the ongoing state. By Geier and Bercher [2011], if a TIHTN problem has a plan, then it has a plan where the number of the actions is at most $k^{|C|}(2^{|\mathcal{L}|}+1)$, which is the bound n of states.

As the sequences of acyclic decompositions and basic actions are finite, Algorithm 1 outputs the failure if it cannot find a plan after a traversal of all sequences.[1]

Next we implement the serialization of the primitive task network via ASP programs. Different from Algorithm 1, we first find an executable action sequence by formula (B.1) then schedule the primitive tasks into the sequence. If O is performed in S, it is possible to consider the primitive task (O, J) to be performed.

[1] We omit the conditions to jump out the loops in Algorithm 1.

Algorithm 1: Decomposition-first TIHTN Planning

input : $\mathcal{P} = (\mathscr{L}, O, C, M, s_I, \mathtt{tn}_I)$ and a state bound n
output: $o_1, \ldots, o_m$ and *dotask*

while *true* **do**
  decompose until $\mathtt{tn}$ is primitive
  repeat
    for $i = 0 \to n - 1$ **do**
      $s \leftarrow s_i$
      while *there is no* $(o, j) \in uncons(T, s)$ *such that* $s \models \mathtt{pre}(o)$ **do**
        choose an o where $s \models \mathtt{pre}(o)$
        $o_i \leftarrow o$
        $s \leftarrow (s \setminus \mathtt{eff}^-(o)) \cup \mathtt{eff}^+(o)$
        $i \leftarrow i + 1$
      choose a $(o, j) \in uncons(T)$ where $s \models \mathtt{pre}(o)$
      $dotask(o, j) \leftarrow i$
      $o_i \leftarrow o$
      $s \leftarrow (s \setminus \mathtt{eff}^-(o_i)) \cup \mathtt{eff}^+(o_i)$
      $i \leftarrow i + 1$
  until *for all* $(o, j) \in T$, $dotask(o, j)$ *is defined*;
  return $o_1, \ldots, o_m$ and *dotask*

Every primitive task is performed once and only one primitive task is selected to perform in a state.

$$0\ \{\mathtt{dotask(O,J,S)}\}\ 1 \text{ :- } \mathtt{do(O,S), tn(O,J,D), final(D)}. \tag{B.27}$$

$$\text{:- } \mathtt{dotask(O,J,S1), dotask(O,J,S2), S1!{=}S2}. \tag{B.28}$$

$$\text{:- } \mathtt{dotask(O,J1,S), dotask(O,J2,S), J1!{=}J2}. \tag{B.29}$$

The schedule of primitive tasks satisfies the constraint $\prec$:

$$\begin{aligned}&\text{:- } \mathtt{dotask(O1,J1,S1), dotask(O2,J2,S2),}\\ &\qquad \mathtt{S1{>}S2, succ(O1,J1,O2,J2,D), final(D)}.\end{aligned} \tag{B.30}$$

If every primitive task is performed, then the plan is found.

$$\mathtt{dotask(O,J)} \text{ :- } \mathtt{dotask(O,J,S)}. \tag{B.31}$$

$$\text{:- not } \mathtt{dotask(O,J), tn(O,J,D), final(D)}. \tag{B.32}$$

We use $\Pi_{\mathtt{S}}$ to denote the set of formulas (B.27)–(B.32). Then we define $\Pi = \Pi_{\mathcal{P}} \cup \Pi_{\mathtt{O}} \cup \Pi_{\mathtt{De}} \cup \Pi_{\mathtt{S}}$ and its every answer set forms a plan of the TIHTN problem:

Theorem B.3. *For a TIHTN problem $\mathcal{P}$, if Π has an answer set, then $\mathcal{P}$ is solvable.*

Sketch. As $\Pi_{\mathtt{De}}$ decomposes the initial task network into a primitive task network, $\Pi_{\mathtt{O}}$ finds an action sequence executable in s_I and $\Pi_{\mathtt{S}}$ schedules all primitive tasks into the sequence which is consistent with the constraints. Given the bound n of states, a basic action sequence is a plan of $\mathcal{P}$ iff it is formed by an answer set of Π. $\square$

If before the scheduling of tasks finishing, i.e., $\mathtt{unfin(S)}$, every action on the sequence is a task, it forms an HTN solution. Then we define Π' from Π by adding the formulas:

$$\mathtt{state\_dotask(S)} \text{ :- } \mathtt{dotask(O, J, S)}. \tag{B.33}$$

$$\mathtt{done(O, J, S)} \text{ :- } \mathtt{dotask(O, J, S)}. \tag{B.34}$$

$$\mathtt{done(O, J, S+1)} \text{ :- } \mathtt{done(O, J, S), state(S+1)}. \tag{B.35}$$

$$\mathtt{unfin(S)} \text{ :- } \mathtt{tn(O,J,D), final(D)}, \text{not } \mathtt{done(O, J, S), state(S)}. \tag{B.36}$$

$$\text{:- not } \mathtt{state\_dotask(S), unfin(S)}. \tag{B.37}$$

Lemma B.4. *For an HTN problem $\mathcal{P}$, if Π' has an answer set then $\mathcal{P}$ has a solution.*

Bibliography

Alchourrón, C. E., Gärdenfors, P., and Makinson, D. (1985). On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(2):510–530.

Alford, R., Bercher, P., and Aha, D. W. (2015a). Tight bounds for HTN planning. In *Proceedings of the 25th Intelligence Conference on Automated Planning and Scheduling (ICAPS)*, pages 7–15. AAAI Press.

Alford, R., Bercher, P., and Aha, D. W. (2015b). Tight bounds for HTN planning with task insertion. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1502–1508. AAAI Press.

Alford, R., Shivashankar, V., Roberts, M., Frank, J., and Aha, D. W. (2016). Hierarchical planning: Relating task and goal decomposition with task sharing. In *Proceedings of the 25fth International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 3022–3029. AAAI Press.

Audi, R. (1982). A theory of practical reasoning. *American Philosophical Quarterly*, 19(1):25–39.

Babiak, T., Křetínskỳ, M., Řehák, V., and Strejček, J. (2012). LTL to Büchi automata translation: fast and more deterministic. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS*, pages 95–109. Springer.

Balbiani, P., Herzig, A., and Troquard, N. (2013). Dynamic logic of propositional assignments: a well-behaved variant of PDL. In *Proceedings of the 28th Annual IEEE/ACM Symposium on Logic in Computer Science (LICS)*, pages 143–152. IEEE.

Baral, C. and Son, T. C. (1999). Extending ConGolog to allow partial ordering. In *Proceedings of the 6th International Workshop on Agent Theories, Architectures, and Languages*, pages 188–204. Springer.

Bauters, K., Liu, W., Hong, J., Sierra, C., and Godo, L. (2014a). CAN(PLAN)+: Extending the operational semantics of the BDI architecture to deal with uncertain information. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 52–61. AUAI Press.

Bauters, K., Liu, W., Hong, J., Sierra, C., and Godo, L. (2014b). CAN(PLAN)+: extending the operational semantics of the BDI architecture to deal with uncertain information. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 52–61. AUAI Press.

Behnke, G., Höller, D., and Biundo, S. (2015). On the complexity of HTN plan verification and its implications for plan recognition. *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS)*, 1:25–33.

Behnke, G., Höller, D., and Biundo, S. (2017). This is a solution! (... but is it though?) - verifying solutions of hierarchical planning problems. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 20–28. AAAI Press.

Bercher, P., Höller, D., Behnke, G., and Biundo, S. (2016). More than a name? On implications of preconditions and effects of compound HTN planning tasks. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, volume 285, pages 225–233.

Bercher, P., Keen, S., and Biundo, S. (2014). Hybrid planning heuristics based on task decomposition graphs. In *Proceedings of the 7th Annual Symposium on Combinatorial Search (SOCS)*. AAAI Press.

Biere, A., Cimatti, A., Clarke, E. M., Fujita, M., and Zhu, Y. (1999). Symbolic model checking using SAT procedures instead of BDDs. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 317–320. ACM Press.

Biundo, S., Bercher, P., Geier, T., Müller, F., and Schattenberg, B. (2011). Advanced user assistance based on AI planning. *Cognitive Systems Research*, 12(3):219–236.

Biundo, S. and Schattenberg, B. (2001). From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning). In *Proceedings of the 6th European Conference on Planning (ECP)*, pages 157–168. AAAI Press.

Bolander, T. (2014). Seeing is believing: Formalising false-belief tasks in dynamic epistemic logic. In *Proceedings of the European Conference on Social Intelligence (ECSI)*, volume 1283, pages 87–107. CEUR Workshop.

Bordini, R. H. and Hübner, J. F. (2010). Semantics for the Jason variant of AgentSpeak (plan failure and some internal actions). In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI)*, pages 635–640. IOS Press.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. J. (2007). *Programming Multi-agent Systems in AgentSpeak using Jason*, volume 8 of *Wiley Series in Agent Technology*. John Wiley & Sons.

Bratman, M. E. (1987). *Intention, Plans, and Practical Reason*. Cambridge University Press. Reedited 1999 with CSLI Publications.

Bratman, M. E. (2009). Intention, belief, and instrumental rationality. In Sobel, D. and Wall, S., editors, *Reasons for action*, pages 13–36. Cambridge University Press.

Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988a). Plans and resource-bounded practical reasoning. *Journal of Computational Intelligence*, 4(3).

Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988b). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355.

Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L.-J. (1992). Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1):165–204.

Clement, B. J. and Durfee, E. H. (1999a). Theory for coordinating concurrent hierarchical planning agents using summary information. In *Proceedings of the 16th National Conference on Artificial Intelligence and 11th Conference on Innovative Applications of Artificial Intelligence, (AAAI/IAAI)*, pages 495–502.

Clement, B. J. and Durfee, E. H. (1999b). Top-down search for coordinating the hierarchical plans of multiple agents. In *Agents*, pages 252–259.

Cohen, P. R. and Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence*, 42(2):213–261.

Dastani, M., de Boer, F., Dignum, F., and Meyer, J. C. (2003). Programming agent deliberation: An approach illustrated using the 3APL language. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 97–104. ACM Press.

de Giacomo, G. and Lenzerini, M. (1995). PDL-based framework for reasoning about actions. *Topics in Artificial Intelligence*, pages 103–114.

de Silva, L. (2017). BDI agent reasoning with guidance from HTN recipes. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 759–767. IFAAMAS Press.

de Silva, L., Lallement, R., and Alami, R. (2015). The HATP hierarchical planner: Formalisation and an initial study of its usability and practicality. In *Proceedings of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6465–6472. IEEE.

de Silva, L. and Padgham, L. (2004). A comparison of BDI based real-time reasoning and HTN based planning. In *Proceedings of the 17th Australasian Joint Conference on Artificial Intelligence*, pages 1167–1173. Springer.

de Silva, L., Sardina, S., and Padgham, L. (2009). First principles planning in BDI systems. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1105–1112. IFAAMAS Press.

Demri, S. (2001). The complexity of regularity in grammar logics and related modal logics. *Journal of Logical Computation*, 11(6):933–960.

Demri, S. and Schnoebelen, P. (2002). The complexity of propositional linear temporal logics in simple cases. *Journal of Information and Computation*, 174(1):84–103.

Dignum, F. and Conte, R. (1997). Intentional agents and goal formation. In *Proceedings of the 4th Intelligent Workshop on Agent Theories, Architectures, and Languages, (ATAL)*, pages 231–243, Berlin. Springer Verlag.

d'Inverno, M., Luck, M., Georgeff, M. P., Kinny, D., and Wooldridge, M. J. (2004). The dMars architecture: A specification of the distributed multi-agent reasoning system. *Autonomous Agents and Multi-Agent Systems*, 9(1-2):5–53.

Duret-Lutz, A. and Poitrenaud, D. (2004). Spot: an extensible model checking library using transition-based generalized büchi automata. In *Proceedings of the IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*, pages 76–83. IEEE.

Dvorak, F., Barták, R., Bit-Monnot, A., Ingrand, F., and Ghallab, M. (2014). Planning and acting with temporal and hierarchical decomposition models. In *Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 115–121. IEEE Computer Society.

Emerson, E. A. (1990). Temporal and modal logic. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 995–1072.

Erol, K., Hendler, J., and Nau, D. S. (1996). Complexity results for HTN planning. *Annals of Mathematics and Artificial Intelligence*, 18(1):69–93.

Erol, K., Hendler, J. A., and Nau, D. S. (1994a). HTN planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI)*, volume 94, pages 1123–1128.

Erol, K., Hendler, J. A., and Nau, D. S. (1994b). UMCP: A sound and complete procedure for hierarchical task-network planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 249–254. AAAI Press.

Erol, K., Hendler, J. A., and Nau, D. S. (1995). Semantics for hierarchical task-network planning. Technical report, DTIC Document.

Fariñas del Cerro, L. and Penttonen, M. (1988). Grammar logics. *Logique Et Analyse*, 31(121-122):123–134.

Gabaldon, A. (2002). Programming hierarchical task networks in the situation calculus. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling Systems Workshop on On-line Planning and Scheduling*.

Geier, T. and Bercher, P. (2011). On the decidability of HTN planning with task insertion. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, volume 22, pages 1955–1961. AAAI Press.

Gerevini, A. and Long, D. (2005). Plan constraints and preferences in pddl3. *Technical Report, Department of Electronics for Automation, University of Brescia, Italy*, 75.

Ghallab, M., Nau, D. S., and Traverso, P. (2004). *Automated Planning: Theory and Practice*. Elsevier.

Goldman, R. P. (2009). A semantics for HTN methods. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press.

Harel, D. (1984). Dynamic logic. In *Handbook of philosophical logic*, pages 497–604. Springer.

Harel, D., Kozen, D., and Tiuryn, J. (2000). *Dynamic logic*. MIT press.

Herzig, A. (2014). Belief change operations: A short history of nearly everything, told in dynamic logic of propositional assignments. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)*. AAAI Press.

Herzig, A., Lorini, E., Moisan, F., and Troquard, N. (2011). A dynamic logic of normative systems. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 228–233. AAAI Press.

Herzig, A., Lorini, E., Perrussel, L., and Xiao, Z. (2017). BDI logics for BDI architectures: old problems, new perspectives. *KI - Künstliche Intelligenz*, 31(1):73–83.

Herzig, A., Perrussel, L., and Varzinczak, I. J. (2006). Elaborating domain descriptions. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI)*, pages 397–401. IOS Press.

Herzig, A., Perrussel, L., and Xiao, Z. (2016a). On hierarchical task networks. In *Proceedings of the 15th European Conference on Logics in Artificial Intelligence (JELIA)*, pages 551–557. Springer.

Herzig, A., Perrussel, L., Xiao, Z., and Zhang, D. (2016b). Refinement of intentions. In *Proceedings of the 15th European Conference on Logics in Artificial Intelligence (JELIA)*, pages 558–563. Springer.

Herzig, A. and Varzinczak, I. J. (2007). Metatheory of actions: Beyond consistency. *Journal of Artificial Intelligence*, 171(16-17):951–984.

Hindriks, K. V. and Meyer, J. C. (2006). Agent logics as program logics: Grounding KARO. In *Proceedings of the 29th Annual German Conference on Artificial Intelligence (KI)*, pages 404–418. Springer.

Hindriks, K. V., van der Hoek, W., and Meyer, J. C. (2012). GOAL agents instantiate intention logic. In *Logic Programs, Norms and Action*, pages 196–219. Springer.

Höller, D., Behnke, G., Bercher, P., and Biundo, S. (2014). Language classification of hierarchical planning problems. In *Proceedings of the 21th European Conference on Artificial Intelligence (ECAI)*, pages 447–452. IOS Press.

Hustadt, U., Dixon, C., Schmidt, R. A., Fisher, M., Meyer, J. C., and van der Hoek, W. (2001). Reasoning about agents in the KARO framework. In *Proceedings of the 8th International Symposium on Temporal Representation and Reasoning, (TIME)*, pages 206–213. IEEE Computer Society.

Icard, T., Pacuit, E., and Shoham, Y. (2010). Joint revision of belief and intention. In *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR)*. AAAI Press.

Ingrand, F. F., Georgeff, M. P., and Rao, A. S. (1992). An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6):34–44.

Kambhampati, S., Mali, A., and Srivastava, B. (1998). Hybrid planning for partially hierarchical domains. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, pages 882–888. AAAI Press.

Labrou, Y. and Finin, T. W. (1994). A semantics approach for KQML - A general purpose communication language for software agents. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), Gaithersburg, Maryland, November 29 - December 2, 1994*, pages 447–455.

Lin, N., Kuter, U., and Sirin, E. (2008). Web service composition with user preferences. In *Proceedings of European Semantic Web Conference (EWSC)*, pages 629–643. Springer.

Lorini, E. and Herzig, A. (2008). A logic of intention and attempt. *Synthese*, 163(1):45–77.

Lorini, E. and Moisan, F. (2011). An epistemic logic of extensive games. *Electronic Notes in Theoretical Computer Science*, 278:245 – 260.

Ma, J., Liu, W., Hong, J., Godo, L., and Sierra, C. (2014). Plan selection for probabilistic BDI agents. In *Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 83–90. IEEE Computer Society.

McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. *Readings in artificial intelligence*, pages 431–450.

Meyer, J. C., de Boer, F. S., van Eijk, R. M., Hindriks, K. V., and van der Hoek, W. (2001). On programming KARO agents. *Logic Journal of the IGPL*, 9(2):245–256.

Nau, D. S., Au, T., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., and Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379–404.

O'Brien, P. D. and Nicol, R. C. (1998). Fipa—towards a standard for software agents. *BT Technology Journal*, 16(3):51–59.

Rao, A. S. and Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 473–484. Morgan Kaufmann Publishers Inc.

Reiter, R. (2001). *Knowledge In Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT press.

Robinson, A. J. and Voronkov, A. (2001). *Handbook of Automated Reasoning*, volume 1. Elsevier.

Sardina, S., de Silva, L., and Padgham, L. (2006). Hierarchical planning in BDI agent programming languages: A formal approach. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1001–1008. ACM Press.

Sardina, S. and Padgham, L. (2011). A BDI agent programming language with failure handling, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems*, 23(1):18–70.

Schut, M. C., Wooldridge, M. J., and Parsons, S. (2004). The theory and practice of intention reconsideration. *Journal of Experimental & Theoretical Artificial Intelligence*, 16(4):261–293.

Shapiro, S., Sardina, S., Thangarajah, J., Cavedon, L., and Padgham, L. (2012). Revising conflicting intention sets in BDI agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1081–1088. IFAAMAS Press.

Shaw, P. H. and Bordini, R. H. (2007). Towards alternative approaches to reasoning about goals. In *Proceedings of 5th International Workshop on Declarative Agent Languages and Technologies*, pages 104–121.

Shilov, N. V. (2005). Designing tableau-like axiomatization for propositional linear temporal logic at home of arthur prior. *Bulletin of Novosibirsk Computing Center. Series: Computer Science*, 23:113–136.

Shivashankar, V., Alford, R., and Aha, D. W. (2017). Incorporating domain-independent planning heuristics in hierarchical planning. In *Proceedings of the 31st Conference on Artificial Intelligence (AAAI)*, pages 3658–3664. AAAI Press.

Shivashankar, V., Alford, R., Kuter, U., and Nau, D. S. (2013). The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2380–2386. AAAI Press.

Shivashankar, V., Kuter, U., Nau, D., and Alford, R. (2012). A hierarchical goal-based formalism and algorithm for single-agent planning. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 981–988. IFAAMAS Press.

Shoham, Y. (2009). Logical theories of intention and the database perspective. *Journal of Philosophical Logic*, 38(6):633–647.

Shoham, Y. (2016). Why knowledge representation matters. *Journal of Communications of the ACM*, 59(1):47–49.

Shoham, Y. and Leyton-Brown, K. (2008). *Multiagent Systems: Algorithmic, Game-theoretic, and Logical Foundations*. Cambridge University Press.

Singh, M. P. (1992). A critical examination of use cohen-levesque theory of intentions. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI)*, pages 364–368.

Sohrabi, S., Baier, J. A., and McIlraith, S. A. (2009). HTN planning with preferences. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1790–1797. AAAI Press.

Sohrabi, S. and McIlraith, S. A. (2009). Optimizing web service composition while enforcing regulations. In *Proceedings of the 8th International Semantic Web Conference, (ISWC),*, pages 601–617. Springer.

Strzalecki, T. (2014). Depth of reasoning and higher order beliefs. *Journal of Economic Behavior & Organization*, 108:108–122.

Tiomkin, M. L. and Makowsky, J. A. (1985). Propositional dynamic logic with local assignment. *Theoretical Computer Science*, 36:71–87.

van Zee, M., Dastani, M., Doder, D., and Torre, L. v. d. (2015a). Consistency conditions for beliefs and intentions. In *2015 AAAI Spring Symposium Series on Logical Formalizations of Commonsense Reasoning*.

van Zee, M. and Doder, D. (2016). AGM-style revision of beliefs and intentions. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI)*, pages 1511–1519. IOS Press.

van Zee, M., Doder, D., Dastani, M., and van der Torre, L. (2015b). AGM revision of beliefs about action and time. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3250–3256. AAAI Press.

Varzinczak, I. J. (2010). On action theory change. *Journal of Artificial Intelligence Research (JAIR)*, 37:189–246.

Waters, M., Padgham, L., and Sardina, S. (2015). Improving domain-independent intention selection in BDI systems. *Autonomous Agents and Multi-Agent Systems*, 29(4):683–717.

Weinstein, J. and Yildiz, M. (2007). Impact of higher-order uncertainty. *Games and Economic Behavior*, 60(1):200–212.

Winikoff, M., Padgham, L., Harland, J., and Thangarajah, J. (2002). Declarative & procedural goals in intelligent agent systems. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 470–481. Morgan Kaufmann Publishers Inc.

Wolper, P. (1985). The tableau method for temporal logic: An overview. *Logique et Analyse*, 28(110–111):119–136.

Wooldridge, M. (2000). Reasoning about rational agents. intelligent robotics and autonomous agents. *The MIT Press, Cambridge, Massachussetts/London.*

Xiao, Z., Herzig, A., Perrussel, L., Wan, H., and Su, X. (2017a). Hierarchical task network planning with task insertion and state constraints. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4463–4469. IJCAI.

Xiao, Z., Herzig, A., Perrussel, L., and Zhang, D. (2017b). Deciding refinement via propositional linear temporal logic. In *Proceedings of the 26th International Conference of the Italian Association for Artificial Intelligence (AI*IA)*, page to appear.

Yao, Y., Logan, B., and Thangarajah, J. (2016). Robust execution of BDI agent programs by exploiting synergies between intentions. In *Proceedings of the 30th National Conference on Artificial Intelligence (AAAI)*, pages 2558–2565. AAAI Press.

Zhou, Y. and Zhang, Y. (2009). Modeling abstract behavior: A dynamic logic approach. In *Proceedings of the 22nd Australasian Joint Conference on Advances in Artificial Intelligence*, pages 538–546. Springer.