THE APPLICATION OF ONTOLOGIES TO REASONING WITH PROCESS MODELING FORMALISMS

by

Xing Tan

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Copyright © 2012 by Xing Tan
Abstract

The Application of Ontologies to Reasoning with Process Modeling Formalisms

Xing Tan

Doctor of Philosophy

Graduate Department of Mechanical and Industrial Engineering

University of Toronto

2012

Reasoning about processes in applications such as manufacturing, web services, enterprise modeling, and planning requires the representation of composite processes with complicated flows of control. Previous research in process representation has used formalisms such as Event Systems, Petri nets, and the Unified Modeling Language activity diagrams. The computational hardness of temporal projection problems in Event Systems has been extensively examined in the literature, whereas Petri nets and UML activity diagrams are applied to describe more elaborate processes. This thesis takes a systematic look into the temporal reasoning problems in Event Systems and assigns accurate semantics to both Petri nets and, for the first time, to UML activity diagrams.

We give an analysis of computational complexity in temporal projection problems by exploring the boundary between their tractable and intractable subproblems. Our results provide new insights into the prominent role the properties of partial ordering play, however we also show that partial ordering is not the sole source of the intractability as has been claimed in an earlier work by Nebel and Bäckström. Two influential modeling languages, Petri nets and UML activity diagrams, are axiomatized as two Basic Action Theories of Situation Calculus. They are called, respectively, SCOPE (Situation Calculus Ontology of PEtri nets) and SCAD (Situation Calculus theory of Activity Diagrams). We provide a Prolog implementation of SCOPE and prove the correctness of this program for regressable queries. We use SCAD to axiomatize the structural and dynamic properties
of UML activity diagrams and also provide the first set of computational results with regard to the reachability problems in activity diagrams. The correctness of each of these two axiomatizations is also demonstrated by proving that the theory is satisfiable, and the intended interpretation corresponds to a model of the theory.
Acknowledgements

This thesis would be impossible to come out without the assistance and encouragement of many people.

First of all, I would like to thank my advisor and mentor Professor Michael Gruninger, for taking me as his apprentice, and for his insight, support and guidance throughout my doctorate studies: every page of this thesis is a reflection of his insightful ideas and his enlightening discussions. His enthusiasm to fundamental research and his dedication to conceptual precision and mathematical rigorousness has made an accumulated and profound impact on my education.

Many thanks to my committee members Professor Sheila McIlraith and Professor Chris Beck, for their invaluable, and always timely, comments, inputs, and guidance to my research over the years.

I in particular thank Professor Mikhail Soutchanski, for his cogent comments on the technical details of the thesis and numerous suggestions on improving the writing of the thesis.

I am fortunate to have Professor Mark Fox sitting in the final committee.

I wish to thank Yilan Gu, for her knowledgable assistance on topics of Situation Calculus. I would like to thank Scott Sanner, who kindly helped me run Vampire, to check the consistency of PSL.

Thanks to Prof. Gruninger, Professor Markus Bussmann, Professor Lily Shu, Professor Roy Kwon, Prof. Beck and the department of MIE, for offering me several teaching assistantship positions. Thanks to Prof. Gruninger and the department of MIE, for granting me a sessional lecturer position. As such, my academic training here at Toronto is significantly enriched with extensive teaching experience. I in particular would like to thank Ms. Brenda Fung at the office of graduate studies of MIE, for her administrative support over the years.

It has been a great pleasure to have these folks around: Mir Sadek Ali, Arnaud
Delaval, Lei Duan, Maryam Fazel-Zarandi, Torsten Hahmann, Ali Hashemi, Megan Katsumi, Xin Gu, Jingwei Huang, Darren Ong, Atalay Ozgovde, Steven Shapiro, Lydia Silva-Munoz.

I am grateful to my parents, who bore me, raised me, and saw me off. I am heavily indebted to my wife Qiong, for her sacrifice to the family over a span of years that reaches well back into the last century. I hope one day my sons Frank and Raymond would read this thesis and be motivated to do something somehow positively meaningful.
Contents

1 Introduction
   1.1 Process Ontologies and Their Applications .................. 1
   1.2 Overview of Thesis ................................……. 4
   1.3 Summary of Contributions ................................... 8

2 Background
   2.1 Concepts Related to Temporal Projection Problems .......... 9
      2.1.1 Event Systems .......................................... 9
      2.1.2 Graph-theoretic Concepts .............................. 14
      2.1.3 Review of Intractable Problems ....................... 16
      2.1.4 Kambhampati and Nau’s Formalisms ................. 17
   2.2 Situation Calculus ..................................... 21
      2.2.1 Concepts .............................................. 22
      2.2.2 Basic Action Theories ................................. 23
      2.2.3 Prolog Implementations ............................... 25
   2.3 Process Specification Languages ............................ 29
      2.3.1 Introduction ......................................... 29
      2.3.2 Modularity .......................................... 32
   2.4 Petri Nets .............................................. 32
      2.4.1 Basic Definitions ..................................... 33
      2.4.2 Petri Nets Properties ................................. 34
   2.5 UML Activity Diagrams ................................... 37

3 Dual Temporal Projection Problems ........................... 39
   3.1 Introduction ............................................. 39
   3.2 Motivations and Correctness of the Axiomatization ......... 41
## 3.3 A Theory of Event Systems

### 3.4 Four Sets of Temporal Projection Duals

- **3.4.1 Nebel and Bäckström’s Duals**
- **3.4.2 Chapman’s Duals**
- **3.4.3 Dean and Boddy’s Duals**
- **3.4.4 Kambhampati and Nau’s Duals (Partial Truth Duals)**

### 3.5 On Alternative Negations

- **3.5.1 Complete Versus Partial Negation**

### 3.6 Summary

## 4 Possible Truth Problems

### 4.1 Introduction

### 4.2 Admissible Possible Truth

- **4.2.1 Intractable Computational Results**
- **4.2.2 Tractable Computational Results**
- **4.2.3 Remarks**

### 4.3 Inadmissible Possible Truth

- **4.3.1 2-layered Planar DAG**
- **4.3.2 Partial Orders**
- **4.3.3 Remarks**

## 5 Partially Ordered Activities in PSL

### 5.1 Introduction

### 5.2 Formalization of the Entailment Problems

- **5.2.1 Queries**
- **5.2.2 Classification of Activities**
- **5.2.3 Process Descriptions**

### 5.3 Complexity of Reasoning Problems

### 5.4 Summary

## 6 SCOPE: A Situation Calculus Ontology of Petri Nets

### 6.1 Introduction

### 6.2 Correctness and Limitations of the Axiomatization

### 6.3 SCOPE

- **6.3.1 The Ontology**
6.3.2 A Basic Action Theory ........................................ 112

6.4 Satisfiability of $\mathcal{D}_{\text{scope}}$ ........................................ 114

6.5 Applications of SCOPE .................................................. 117
   6.5.1 Axiomatization of SCOPE Subclasses ......................... 117
   6.5.2 Examples ............................................................... 119

6.6 Prolog Implementation of SCOPE ...................................... 122
   6.6.1 Implementation ....................................................... 123
   6.6.2 An Example ............................................................ 125

6.7 Summary ................................................................. 127

7 UML Activity Diagrams .................................................. 130
   7.1 Introduction ............................................................. 130
   7.2 Motivations .............................................................. 133
   7.3 UML Activity Diagrams ................................................. 134
      7.3.1 Basic Definitions .................................................. 135
      7.3.2 Markings and Reachability Problem ....................... 135
   7.4 Correctness and Limitations of the Axiomatization ............ 139
   7.5 SCAD ..................................................................... 141
      7.5.1 The Ontology ....................................................... 141
      7.5.2 A Basic Action Theory ......................................... 145
   7.6 Satisfiability of $\mathcal{D}_{\text{scad}}$ .................................. 146
   7.7 Applications .............................................................. 148
      7.7.1 Axiomatization of SCAD Subclasses ......................... 149
      7.7.2 Computational Complexity .................................... 151
   7.8 Summary ................................................................. 159

8 Conclusions ............................................................... 161
   8.1 Contributions ........................................................... 161
   8.2 Future Research .......................................................... 163
   8.3 Conclusions .............................................................. 166

A Process Specification Language ....................................... 167
   A.1 PSL-Core ................................................................. 167
      A.1.1 Informal Semantics for PSL-Core ....................... 168
      A.1.2 Axioms of PSL-Core ........................................... 170
List of Tables

1.1 Formalisms and their axiomatizations in Situation Calculus and PSL . . 4
3.1 Complexity results for Nebel and Bäckström’s Dualities . . . . . . . . 51
3.2 Complexity results for Chapman’s Dualities . . . . . . . . . . . . . . 53
3.3 Complexity results for Dean and Boddy’s Dualities . . . . . . . . . . 56
3.4 Complexity results for Partial Dualities . . . . . . . . . . . . . . . . . 56
4.1 Existing and new complexity results on Possible Truth problems . . . . 66
7.1 Four levels of UML Activities . . . . . . . . . . . . . . . . . . . . . . . 132
# List of Figures

2.1 An example of situation tree (Source: Figure 4.1 of [59]). ................. 23  
2.2 PSL core theories ................................................................................. 33  
2.3 An example of Petri nets (Source: Figure 17-h of [50]) ....................... 34  
2.4 An example of Petri nets: not-1-bounded, non-live, irreversible (Source:  
  Figure 17-a of [50]) ............................................................................ 35  
4.1 NP-complete admissible Possible Truth Problems ................................. 66  
4.2 A digraph and its Hamiltonian path ...................................................... 71  
4.3 The transformed Almost-simple Event System ..................................... 72  
4.4 The transformed Simple Event System ............................................... 76  
4.5 The transformed Almost-simple, DAG, Event System ......................... 78  
4.6 The computational hierarchy of admissible Possible Truth problems .... 80  
4.7 Transformation from an One-In-Three 3SAT Instance to a PAFP Problem 83  
4.8 The computational hierarchy of inadmissible Possible Truth problems .. 89  
5.1 Example of a permuted activity tree. .................................................... 96  
5.2 Example of a nonpermuted activity tree. ............................................. 96  
5.3 Example of a folded activity tree. ....................................................... 96  
5.4 Example of a strong poset activity tree. ............................................ 97  
5.5 Example of a concurrent poset activity tree. ..................................... 98  
6.1 An examples of Petri nets: 1-bounded, non-live, and irreversible (Source:  
  Figure 17-d of [50]) ............................................................................ 120  
7.1 An example of a UML activity diagram with ambiguous execution semantics131  
7.2 An example of an intermediate UML activity diagram ......................... 134  
7.3 Another example of a UML activity diagram .................................... 139  
7.4 The action sequences that reach the Fin node ..................................... 140
Chapter 1

Introduction

1.1 Process Ontologies and Their Applications

Process ontologies are controlled vocabularies to describe complicated processes in diverse domains ranging from manufacturing to commonsense reasoning. Formal process ontologies are enriched with axioms written in a logical language. This enables automated reasoning about the processes, such as predicting the outcome of a process. In addition, the consistent integration of processes from different sources can be performed without the need to reformat the process descriptions and additional efforts to ensure that these processes have shared and unambiguous meaning.

A typical process consists of a set of interdependent components. Representing the flows of control on these components and reasoning over their innate intricacy arises frequently in many disciplines, such as industrial engineering [35, 82], software engineering [18], business process management [83], enterprise modeling [20], and manufacturing process planning [6]. Typical modeling languages used to represent process information are graphical ones, where labeled nodes are used to represent concepts, and concepts are connected by directed edges to specify their relationships. Constraints might also be represented by additional graphical notations. One example of such a graphical modeling language is the Unified Modeling Language (UML) activity diagrams [54], which have widespread use in software industry for controlling the software system development processes. Another example is Petri net diagrams [50], which, in addition to providing their graphical notations for process constructs such as choice, iteration, and concurrent execution, have a precise mathematical definition of the semantics of their dynamical behaviors. Event Systems [12, 53], an influential framework for modeling actions and
changes in dynamical domains of Artificial Intelligence, are textual descriptions of events, the preconditions of events, the consequences of event occurrences, and the ordering relations between events.

The verification of the dynamical properties of a system represented in these formalisms is primarily done by applying the techniques of simulation. However, as the size of a system increases, these techniques usually require exponential increase in the amount of time to run. Simulation therefore cannot offer any theoretical assurance of the dynamical properties for large-scale systems. Alternatively, formal methods, (those based on mathematical logic for the specification and verification of system properties) achieve verification by providing formal deductive proofs that are based on formal descriptions of the systems and their properties. Given that almost none of the formalisms discussed above are presented with complete and rigorous axiomatizations, we are motivated accordingly to propose the question: How can we efficiently reason about the dynamical properties of systems represented in these formalisms? More precisely, how can we specify reasoning problems for these formalisms as entailments from sets of axioms? And what is the complexity of these reasoning problems?

The central topic of this dissertation is the evaluation of potential applications of process ontologies to the domain of process representation and reasoning. In the thesis, we axiomatized the intended semantics of three process representation formalisms, Event Systems, Petri nets, and UML activity diagrams, whereas interesting classes of queries that are associated with these formalisms are also axiomatized. Our basic approach is to define these new ontologies by extending existing process ontologies – Situation Calculus and the Process Specification Language (PSL).

Situation Calculus is an ontology representing simple and atomic actions and facilitates approaches for reasoning about these actions. A Basic Action Theory is a formal theory of simple actions specified in Reiter’s version of Situation Calculus [59]; it consists of the foundational axioms for situations, application-specific action precondition axioms, successor state axioms for fluents, initial situation axioms, and a definition of executable situations. Complex actions can be described through defining macros via expanding Basic Action Theories in Situation Calculus (e.g., Golog [59]). Alternatively, we can directly reify complex actions as objects. For example, the design of the Process Specification Language [29, 31] opts to include complex actions, and complex action occurrences as genuine first-order terms. The PSL ontology is a modular set of theories in the language of first-order logic. Within PSL, the core theory, referred to as PSL-Core,
introduces the four primitive concepts that are possibly associated with the description of a process: activities, activity occurrences, time points, and objects that participate in activities. Additional theories are consistent extensions of PSL-Core. These theories capture the intuitions for the composition of activities, and describe the relationship between an occurrence of a complex activity, and the occurrences of its subactivities. More details on Situation Calculus, PSL, and their relationship, are covered in Chapter 2.

This thesis demonstrates powerful applications of Situation Calculus and the Process Specification Language (PSL) for representing and reasoning about processes:

- Event Systems, Petri nets, and UML activity diagrams are axiomatized as Basic Action Theories in Situation Calculus, whereas a complex activity with partially-ordered, non-deterministic component action occurrences is axiomatized as a PSL domain theory.

- In addition, these ontologies are used
  
  - to facilitate analytical investigation of complicated reasoning problems and the relations between these problems;
  
  - to characterize subclasses of graphical models;
  
  - to perform deductive reasoning to verify properties of dynamical systems, and

  - to support analysis of computational complexity of querying.

Process ontologies built in this thesis are summarized in Table 1.1. Using Situation Calculus, the process modeling formalisms of Event Systems, Petri nets, UML activity diagrams, are axiomatized, respectively, as three Basic Action Theories $\mathcal{D}_{cs}$, $\mathcal{D}_{scope}$, and $\mathcal{D}_{scad}$. Using the Process Specification Language, a plan composed of partially ordered subactivity occurrences is axiomatized as a union of PSL core theories and $\sum_{pd}(P)$, which is a process description of the plan. Each axiomatization of a process modeling formalism is presented with a mathematical justification of its correctness. For the axiomatization of Petri nets and UML activity diagrams, we prove the satisfiability of the theories, that is, we prove that their intended interpretations in the form of mathematical definitions of a graphical process diagram (i.e., a Petri net or a UML activity diagram), are indeed models of the theories.

Equipped with a formal semantics, we investigate a series of reasoning tasks which
Table 1.1: Formalisms and their axiomatizations in Situation Calculus and PSL

<table>
<thead>
<tr>
<th>Formalism</th>
<th>Process Ontologies</th>
<th>Axiomatization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Systems</td>
<td>Situation Calculus</td>
<td>$D_{es}$</td>
</tr>
<tr>
<td>Petri Nets</td>
<td>Situation Calculus</td>
<td>$D_{scope}$</td>
</tr>
<tr>
<td>UML activity diagrams</td>
<td>Situation Calculus</td>
<td>$D_{scad}$</td>
</tr>
<tr>
<td>Process Plans</td>
<td>PSL</td>
<td>$\mathcal{T}<em>{psl} \cup \sum</em>{pd}(P)$</td>
</tr>
</tbody>
</table>

can be specified as a logical entailment problem of the form

$$D \models \phi.$$ 

On the left-hand side, the expression $D$ is a logical theory that describes a particular process instance that is specified in terms of one of the process representation frameworks mentioned above. In general, the theory $D$ consists of a union of $D_{ontology}$ and $D_{domain}$, where the ontological part $D_{ontology}$ is the generic, reusable portion of the theory that is instance-independent and the domain theory $D_{domain}$ defines an instance, i.e., a particular use case scenario for the ontology. The right-hand side of the expression $\phi$ is a query sentence in the language of the ontology and a given domain theory. Once a correct process theory is offered, we look into its corresponding entailment problem from three different perspectives: (1) either we provide further axiomatizations on restricted subclasses of the theory where the execution semantics of the theory is in accordance with certain structural or behavioral properties, or (2) we evaluate whether certain relations between query sentences (such as the equivalence between formula $\phi_1$ and formula $\phi_2$) would be admitted by a given theory, or (3) we simply examine the computational difficulty of evaluating the entailment problem by delineating several boundaries between tractable and intractable.

## 1.2 Overview of Thesis

The major chapters in this thesis can be divided into three parts. Part one (Chapter 3, 4) looks into the class of temporal projection problems. Part two (Chapter 5) primarily addresses partially ordered activities with nondeterministic occurrences. Part three (Chapter 6 and 7) turns to two graphical models: Petri nets and UML activity diagrams.

Chapter 3 starts by axiomatizing Event Systems as a Basic Action Theory of Reiter’s
Chapter 1. Introduction

version of Situation Calculus. Using the language of the theory, we specify three basic concepts that are related to a linear completion of actual events in an Event System: order, which means checking whether the completion satisfies all of the order constraints; admissibility, which means checking whether all preconditions of all actual events in the completion are satisfied; and achievability, which means checking whether certain propositional literals can be achieved by the completion. Four sets of temporal projection problems, each of which characterizes a unique relationship between order, admissibility, and achievability, are then introduced as query sentences. Each set contains eight reasoning problems, distinguished by the possible version or the necessary version of a statement about the truth value of a given propositional literal. In addition, for each set, we demonstrate four temporal dualities in the form of four equivalences between these problems that hold in any theory of Event Systems. Equipped with this framework, we determine the computational complexity of entailing each of these queries.

Having specified several different classes of problems of temporal duality, we move on in Chapter 4 to analyzing the computational complexity of a particular class of temporal projection problems, called Possible Truth problem. Our complexity analysis makes a substantive extension to the existing results. More precisely, it is known that one particularly restricted Possible Truth problem, first conjectured in [12] to be polynomial-time solvable, is later shown by Bernhard Nebel and Christer Bäckström in [53] to be NP-complete. This intractability result provides a correction to the complexity results of temporal projection problems reported in [12], making it affirmative that the set of partial orders is the major source of the intractability of temporal projection problems. We continue this line of work by focusing in Chapter 4 on a variant of temporal projection problems, where the admissibility of event occurrences is enforced, we looked into several combinations of different constraints with regard to this variant and obtained several interesting NP-completeness results before the tractability is finally achieved. These results, summarized in Section 4.2, provide solid evidence that, aside from partial orders on events, several other problem characteristics all contribute to the intractability of the problem, including the size of the preconditions lists and effects lists, the size of the initial set, and the topological structure of the cause-and-effect diagrams. In Section 4.3, we return to the original boundary case Possible Truth problem, which does not require admissibility and is associated with the class of Simple Event Systems. Two types of constraints, one on the graph-theoretic representation of the cause-and-effect relationships between events and the other on the partial orders of events, are added
in an attempt to gain tractability. In particular, we show that the problem is still NP-complete even if the cause-and-effect graphs maintain a strongly restricted topology, whereas it is tractable if the graph is closely associated with the set of pairwise disjoint partial orders, which is hierarchically structured. The results verify the claims in [53] that the role of partial orders of events on intractability of the problems is prominent, and that in order to obtain tractability, the partial orders of events must have a highly restricted mathematical structure.

The application of PSL is summarized in Chapter 5. In particular, we formalize in first-order logic (1) a set of queries that are related to occurrences of PSL complex activities and the ordering constraints on those occurrences, and (2) four classes of activity trees, namely permuted, folded, strong poset, and concurrent poset activity trees, with their respective model-theoretic definitions. We then evaluate the computational complexity of evaluating these queries as entailment problems in PSL ontologies whose activity trees can be characterized by these classes, and provide a clear boundary between tractable PSL query problems and their counterpart NP-complete ones.

As noted earlier, Petri nets and UML activity diagrams are the two graphical models which are considered in this thesis. The dynamical aspects in a system of Petri nets are captured by the firing of transitions that move tokens (possibly with the total number of tokens in the system changed) between places. UML activity diagrams include several components (such as Fork, Join, Branch, Merge) to specify the flow of controls of a system described by an activity diagram.

In Chapter 6, the semantics of Petri nets are axiomatized as a second-order formal ontology called SCOPE (Situation Calculus Ontology of PEtri nets). The ontology SCOPE is built as a Basic Action Theory in Reiter’s version of Situation Calculus ($\mathcal{D}_{scope}$). Note, in this thesis we consistently use SCOPE to refer to the ontology we have developed and $\mathcal{D}_{scope}$ to refer to the formal axiomatization of SCOPE. By virtue of the Relative Satisfiability Theorem, we show that $\mathcal{D}_{scope}$ is satisfiable.\(^1\) A set of structural and dynamical properties of Petri nets (e.g., reachability, k-boundedness, liveness, S-systems, T-systems) are also axiomatized, with example analysis given. In the chapter, a concise and mathematically rigorous Prolog implementation of $\mathcal{D}_{scope}$ is further presented. The correctness of using this program to evaluate the class of regressable queries in a Petri

\(^{1}\)Actually, the models for some core part of SCOPE are characterized by mathematical structures (see Section 6.4 of the thesis).
Chapter 1. Introduction

A net system is proven. As a result, the program can be used in practice to test whether a sequence of Petri net token firing is executable, and to check the results of an executable firing sequence.

In a similar manner, we axiomatize the semantics of UML activity diagrams in Chapter 7. Unlike Petri nets, the syntax and semantics of activity diagrams are currently only specified by textual statements. Consequently, we start by providing graph-theoretic definitions of the structure of activity diagrams as directed graphs with typed nodes, while the dynamical properties are captured by introducing the concepts of markings and firing of tokens. UML activity diagrams are axiomatized in a similar manner to the way we created SCOPE for Petri nets. The resulting activity diagram theory is called SCAD (Situation Calculus theory of Activity Diagrams). Note, in this thesis we consistently use SCAD to refer to the ontology we have developed and $D_{scad}$ to refer to the formal axiomatization of SCAD. The satisfiability of $D_{scad}$ is proven, and the correctness of $D_{scad}$ on the intended interpretation is partially proven. The deductive verification of several well-known dynamical properties are also given using $D_{scad}$. The ontology SCAD provides a firm foundation for two types of important tasks on the analysis of activity diagrams: (1) defining subclasses of activity diagrams that satisfy certain dynamical properties such as reversibility and reachability; and (2) addressing alternative choices of the intended semantics. With the execution semantics of activity diagrams in hand, we then provide the first set of complexity results with respect to activity diagrams. More precisely, we show that the reachability problems (checking whether a node can be reached from some initial settings such that all restrictions on flows of control are satisfied) are at least PSPACE-hard, whereas their acyclic variants are NP-complete.

Background knowledge for the various formalisms and ontologies that are discussed in the thesis is provided in Chapter 2. Definitions of preliminaries for Event System are presented in Section 2.1. A review of Reiter’s version of Situation Calculus (Section 2.2) and PSL (Section 2.3) are also included. The emphasis in these sections is placed on introducing the foundational axioms, which characterize the models of activity trees (for Situation Calculus), and on showing the correspondence between activity trees and occurrence trees (in PSL). All PSL axioms that are essential to understanding the material presented in Chapter 5 are listed in Appendix A. Finally, the basics for Petri nets and UML activity diagrams are introduced in Section 2.4 and Section 2.5, respectively.
1.3 Summary of Contributions

Major contributions of this thesis are as follows:

- Within Event Systems axiomatized by Situation Calculus, we consider the temporal projection problems where the admissability of events is required. Our results show that although this variant problem is simpler than the original, it is still intractable in general. We demonstrate that a wide variety of constraints all contribute to the computational intractability of temporal projection problems.

- For the inadmissible temporal projection problems, we demonstrate two distinct but also related cases where tractability is obtained. This work brings in significant new insights on how to develop empirical algorithms to tackle these intractable problems.

- We study domains with nondeterministic activity occurrences, with the assumption that their occurrences do not require any preconditions. We provide a precise delineation of tractable classes of complex activities that are constructed from partially ordered component activities.

- Important dynamical and structural properties of Petri nets are axiomatized. We provide several examples of deductive reasoning to verify the properties, such as liveness and reversibility and 1-boundedness, of Petri nets and of activity diagrams. We provide a Prolog-based interpreter for Petri nets, which differs from existing Prolog interpreters for Petri nets in that only our implementation guarantees the preservation of the semantics from the transformation between these two formalisms.

- We define reachability problems in UML activity diagrams, which are of great importance in studying the behavior and structural properties of UML activity diagrams, and provide the first set of intractability complexity results for this problem.
Chapter 2

Background

We begin this chapter by reviewing the concepts of Event Systems and the temporal projection problems in Event Systems. Two primary process ontologies that will be used throughout the thesis – Situation Calculus and the Process Specification Language, are then introduced. Finally, we introduce the two flow models which will be analyzed in the remaining chapters – Petri nets, and UML activity diagrams.

2.1 Concepts Related to Temporal Projection Problems

2.1.1 Event Systems

Most definitions in this section are adapted from [53], and they offer a refinement, but are essentially equivalent to the definitions in [12].

Definition 1 (Causal Structure) A Causal Structure is defined as a 3-tuple \( \Phi = \langle P, T, R \rangle \) where

- \( P = \{ p_1, p_2, \ldots, p_n \} \) is a set of propositional atoms (conditions);
- \( T = \{ t_1, t_2, \ldots, t_m \} \) is a set of event types;
- \( R = \{ r_1, r_2, \ldots, r_p \} \) is a set of causal rules in the form \( r_i = \langle t_i, \varphi_i, \alpha_i, \delta_i \rangle \) where
  - \( t_i \in T \) is the event type that triggers the application of \( r_i \),
  - \( \varphi_i \subseteq P \) is the set of preconditions,
– $\alpha_i \subseteq \mathcal{P}$ is the set of added conditions,
– $\delta_i \subseteq \mathcal{P}$ is the set of deleted conditions.

Informally, the concept of Causal Structure is introduced to specify how a given state, which is a subset of $\mathcal{P}$, evolves over actual occurrences of events. The notion of Event System is introduced to describe a set of actual events that are subject to temporal constraints in the form of partial orders:

**Definition 2 (Event System)** An Event System is a 6-tuple $\Theta = \langle \mathcal{P}, \mathcal{T}, \mathcal{R}, \mathcal{E}, \mathcal{O}, \mathcal{I} \rangle$ where
- $\mathcal{P}, \mathcal{T}, \mathcal{R}$ are the same as the ones defined in $\Phi$,
- $\mathcal{E} = \{e_1, e_2, \ldots, e_p\}$ is a set of actual events, such that for each $e_i$, type($e_i$) $\in \mathcal{T}$,
- $\mathcal{O}$ is a set of strict partial orders\(^1\) on $\mathcal{E}$,
- $\mathcal{I}$ is the initial state, a subset of $\mathcal{P}$.

**Definition 3 (Almost-simple Event System)** An Event System $\Theta$ is Almost-simple iff
- it is unconditional (i.e. for each event type $t \in \mathcal{T}$, there exists only one causal rule for $t$), and
- for each causal rule in the form $r = \langle t, \varphi, \alpha, \delta \rangle$
  - $|\varphi| = |\alpha| = |\delta| = 1$,
  - $|\mathcal{I}| \geq 1$,
  - $\delta = \varphi$.

**Definition 4 (Simple Event System)** A Simple Event System $\Theta$ is an Almost-simple event system, with additional constraint $|\mathcal{I}| = 1$.

**Definition 5 (Cause-and-Effect Graph)** A Cause-and-Effect Graph of an Almost-simple Event System $\Theta$ is a directed graph. Each node in the graph corresponds to a condition, whereas each directed edge in the graph corresponds to a causal rule in $\Theta$.

---

\(^1\)A strict partial order is an irreflexive, antisymmetric, and transitive binary relation [78].
Given a causal structure $\Phi$ and a state $State$, an actual event $a$ with event type $t \in T$ is applicable in $State$ if and only if the preconditions for $t$ are satisfied in $State$. If applicable, $a$ changes $State$ by adding some conditions to $State$ and removing some others from it, in the way specified by the rule $R$. However, if not applicable, $a$ is effectless to $State$. A sequence of applicable events changes the state $State$ sequentially.

Formally, let $app(State, e)$ denotes the set of all applicable rules for an event $e \in E$ in state $State$, $e$ is admissible in $State$ iff $app(State, e) \neq \emptyset$. Also, let $f = \langle f_1, \ldots, f_k, \ldots, f_m \rangle$ be an event sequence, $\langle f_1, \ldots, f_k \rangle$ is denoted by $f/f_k$, whereas $\langle f_1, \ldots, f_{k-1} \rangle$ is denoted by $f \setminus f_k$. Further, we write $f; g$ to denote $\langle f_1, \ldots, f_m, g \rangle$.

**Definition 6 (Result)** The change of states (subset of $P$) over event sequences $f$ (initially empty, denoted by $\langle \rangle$) is defined recursively as follows

- $Result(State, \langle \rangle) = State$,
- $Result(State, (f; g)) = Result(State, f) - \{\delta(r) | r \in app(Result(State, f), g)\} \cup \{\alpha(r) | r \in app(Result(State, f), g)\}$.

**Definition 7 (Admissability)** An event sequence $f = \langle f_1, \ldots, f_k, \ldots, f_m \rangle$ is admissible in a state $State$ iff each $f_i$, for $1 \leq i \leq m$, is admissible in $Result(State, f \setminus f_i)$.

Here is a naive example to illustrate the definitions. There are two rooms, room $alpha$ and room $beta$. In addition, there is a coffee maker in room $beta$. The set of conditions is $P = \{a, b, c\}$, referring to the facts that the robot ray is in room $alpha$, ray is in room $beta$, and, coffee is ready, respectively. The set of event types is $T = \{t_{ab}, t_{ba}, t_{mc}\}$, referring to the event types of ray moving from $alpha$ to $beta$, from $beta$ to $alpha$, making coffee at room $beta$, respectively. In addition, we have the set of causal rules

$$R = \{\langle t_{ab}, \{a\}, \{b\}, \{a\}\rangle, \langle t_{ba}, \{b\}, \{a\}, \{b\}\rangle, \langle t_{mc}, \{b\}, \{c\}, \{\}\rangle\}.$$  

That is, ray can move to room $beta$ if he is in room $alpha$, and vice versa. And ray can make coffee if he is in room $beta$.

Additionally, we define a particular Event System $\Theta_{ray}$, where we have (1) the set of actual events $E = \{\text{from}A, \text{from}B_1, \text{from}B_2, C\}$, where $\text{type}(\text{from}A) = t_{ab}$, $\text{type}(\text{from}B_1) = t_{ba}$, $\text{type}(\text{from}B_2) = t_{ba}$, $\text{type}(C) = t_{mc}$; (2) The robot ray is initially in room beta, i.e., $I = \{b\}$; (3) The set of partial orders is $O = \{\text{from}A \prec C\}$, that
is, for some unknown reason, ray must visit room alpha before he makes coffee in room beta.

Within $\Theta_{\text{ray}}$, consider a sequence “fromB$_1$ fromB$_2$”. It is easy to see that fromB$_1$ is admissible and fromB$_2$ is inadmissible, since after fromB$_1$, ray is in room alpha, thus the precondition for fromB$_2$ to execute admissibly no longer holds. That is to say, after the inadmissible execution of fromB$_2$, ray is still in room alpha.

Meanwhile, the sequence “fromB$_2$ fromA fromB$_1$” is admissible, satisfies the only constraint in C, and achieves the condition c.

Given a set of events $E$ in an Event System $\Theta$, a set of completions (i.e., linear extensions) with respect to $E$ is defined as a set of permutations of all events in $E$, with the constraints of partial orders on the set of events (i.e., $O \in \Theta$) satisfied. This set is denoted by $\mathcal{CS}(\Theta)$. Meanwhile, the set of all completions that are admissible with respect to the initial state $I$ is denoted by $\mathcal{ACS}(\Theta)$. Accordingly, two basic temporal projection decision problems, i.e., the problem of deciding possible consequence of events and the problem of deciding necessary consequence of events, involve deciding in an Event System $\Theta$ whether a given condition $p$ holds in the state obtained from execution of one (or all) linear extensions in $\mathcal{CS}(\Theta)$ (or $\mathcal{ACS}(\Theta)$), respectively. More precisely, depending whether admissibility is imposed, two variants of the Possible Truth Problems and the Necessary Truth problems are defined as follows. Note that the predicate $\text{Adm}(I, f/e)$, which appears in the definitions for admissible problems, denotes that the sequence $f/e$ is admissible in the state $I$.

**Definition 8** (Adopted from Definition 2.6 and Definition 6.1 of [53]) Given an Event System $\Theta$, an event $e \in E$, and a condition $p \in P$:

* $p \in \text{Poss}^+(e, \Theta)$ iff $\exists f \in \mathcal{CS}(\Theta) : p \in \text{Result}(I, f/e)$, that is, the condition $p$ is possibly true in an Event System $\Theta$, iff there exists an event sequence $f$, which is a completion of $\Theta$, such that $p$ is true after the execution of the event $e$ in $f$ from the initial state $I$ in $\Theta$;

* $p \in \text{Poss}_A^+(e, \Theta)$ iff $\exists f \in \mathcal{CS}(\Theta) : \text{Adm}(I, f/e) \land p \in \text{Result}(I, f/e)$, that is, the condition $p$ is possibly and admissibly true in an Event System $\Theta$, iff there exists an event sequence $f$, which is a completion of $\Theta$, such that $p$ is true after the admissible execution of the event $e$ in $f$ from the initial state $I$ in $\Theta$;

* $p \in \text{Nec}^+(e, \Theta)$ iff $\forall f \in \mathcal{CS}(\Theta) : p \in \text{Result}(I, f/e)$, that is, the condition $p$ is
necessarily true in an Event System \( \Theta \), iff for any event sequence \( f \), which is a completion of \( \Theta \), the condition \( p \) is true after the execution of the event \( e \) in \( f \) from the initial state \( I \) in \( \Theta \):

- \( p \in \text{Nec}^+_A(e, \Theta) \) iff \( \forall f \in \text{CS}(\Theta) : \text{Adm}(I, f/e) \land p \in \text{Result}(I, f/e) \). that is, the condition \( p \) is necessarily and admissibly true in an Event System \( \Theta \), iff for any event sequence \( f \), which is a completion of \( \Theta \), the condition \( p \) is true after the execution of the admissible event \( e \) in \( f \) from the initial state \( I \) in \( \Theta \);

With Definition 8, several theorems are provided in the literature. The following Proposition is adapted from Theorem 2.1 of [12].

**Proposition 1** Deciding \( p \in \text{Poss}^+(e, \Theta) \) is an NP-complete problem.

The following Proposition is adapted from Theorem 2.2 of [12].

**Proposition 2** Deciding \( p \in \text{Nec}^+(e, \Theta) \) is an NP-hard problem.

The following Proposition is adapted from Theorem 3.3 of [53].

**Proposition 3** Deciding \( p \in \text{Poss}^+(e, \Theta) \), where \( \Theta \) is an Simple Event System, is an NP-complete problem.

The following Proposition is adapted from Corollary 6.5 of [53].

**Proposition 4** Deciding \( p \in \text{Poss}^+_A(e, \Theta) \) is NP-complete problems.

The following Proposition is adapted from Theorem 6.3 of [53].

**Proposition 5** Deciding \( p \in \text{Nec}^+_A(e, \Theta) \) is polynomial-time problems.

Note that for all problems introduced in Definition 8, the value of \( p \) is checked right after \( e \). However, a set of slightly different problems is presented as follows, where the value of \( p \) is checked after all actual events in \( \mathcal{E} \) of \( \Theta \) are executed instead.

**Definition 9** Given an Event System \( \Theta \), and a condition \( p \in \mathcal{P} \):

- \( p \in \text{Poss}^+(\Theta) \) iff \( \exists f \in \text{CS}(\Theta) : p \in \text{Result}(I, f) \), that is, the condition \( p \) is possibly true in an Event System \( \Theta \), iff there exists an event sequence \( f \), which is a completion of \( \Theta \), such that \( p \) is true after the execution of \( f \) from the initial state \( I \) in \( \Theta \);
Chapter 2. Background

• \( p \in \text{Poss}^+(\Theta) \) iff \( \exists f \in \text{ACS}(\Theta) : p \in \text{Result}(I, f) \), that is, the condition \( p \) is possibly true in an Event System \( \Theta \), iff there exists an event sequence \( f \), which is an admissible completion of \( \Theta \), such that \( p \) is true after the execution of \( f \) from the initial state \( I \) in \( \Theta \);

• \( p \in \text{Nec}^+(\Theta) \) iff \( \forall f \in \text{CS}(\Theta) : p \in \text{Result}(I, f) \), that is, the condition \( p \) is necessarily true in an Event System \( \Theta \), iff for any event sequence \( f \), which is a completion of \( \Theta \), the condition \( p \) is true after the execution of \( f \) from the initial state \( I \) in \( \Theta \);

• \( p \in \text{Nec}^+_A(\Theta) \) iff \( \forall f \in \text{ACS}(\Theta) : p \in \text{Result}(I, f) \). that is, the condition \( p \) is necessarily true in an Event System \( \Theta \), iff for any event sequence \( f \), which is an admissible completion of \( \Theta \), the condition \( p \) is true after the execution of \( f \) from the initial state \( I \) in \( \Theta \);

A problem defined in Definition 9 can be treated as a special case of its corresponding problem defined in Definition 8. For example, an instance of the problem \( p \in \text{Poss}^+(\Theta) \) can be transformed into an instance of the problem \( p \in \text{Poss}^+(e_{\text{new}}, \Theta_{\text{new}}) \), by introducing a new event \( e_{\text{new}} \) that does not occur previously in \( E \) and is proceeded by all other events in \( \Theta_{\text{new}} \). Thus, if deciding \( p \in \text{Poss}^+(\Theta) \) is intractable, deciding \( p \in \text{Poss}^+(e_{\text{new}}, \Theta_{\text{new}}) \) should be intractable as well. Meanwhile, a polytime algorithm for \( p \in \text{Poss}^+(\Theta) \) can be easily modified to solve \( p \in \text{Poss}^+(e_{\text{new}}, \Theta_{\text{new}}) \) in polytime. Hence, from a complexity-theoretic point of view, Definition 8 and Definition 9 are equivalent. In the thesis, we mostly use Definition 9, because

• in Chapter 3, we are going to address the validity of the results on dual truths in [38], and these definitions match exactly with theirs;

• the relationship between temporal projection problems and planning problems can be addressed more clearly, thus a straightforward presentation of our results is facilitated (as indicated in Chapter 4).

2.1.2 Graph-theoretic Concepts

A digraph is a pair \( G = \langle N, E \rangle \) where \( N \) is a finite set of nodes and \( E \) consists of edges, i.e., ordered pairs of nodes on \( N \). A directed graph is acyclic (DAG) if there does not exist a sequence \( \langle v_1, v_2, \ldots, v_{k-1}, v_k \rangle \) of nodes such that \( v_1 = v_k \) and \( (v_i, v_{i+1}) \in E \) for
A DAG is *layered* if $N$ can be partitioned into a sequence of $p$ disjoint subsets (layers), $l_1, \ldots, l_p$, such that all edges in $E$ are between consecutive layers. A DAG is *planar* if it can be drawn on the plane without crossing edges.

Among all of the following definitions and propositions, the Path Avoid Forbidden Pairs problems (PAFP) play a pivotal role. In Section 4.3.1, we are going to prove that PAFP-E (i.e., path avoid forbidden pairs of edges) in a highly restricted class of DAG (namely, 2-Layered Planar s-t-DAG, defined below) is still NP-complete, and then we use this result to present an NP-completeness result for a restricted class of inadmissible Possible Truth problems. In Section 4.3.2, we prove that PAFP-E, where the forbidden pairs of edges are with certain special structure (namely, Edges Hierarchical Structure), can be solved in polynomial time and then we show that the problem is polynomial-time equivalent to another class of inadmissible Possible Truth problems.

**Definition 10** A 2-Layered Planar s-t-DAG is a layered planar DAG with exactly one source node $s$ and one sink node $t$, and the size of each layer is at most 2.

**Definition 11 (PAFP of Vertices)** Given a digraph $G = (V, E)$, specified vertices $s, t \in V$, a set $F_{\text{vertices}}$ of pairs of vertices from $V$, the problem of finding a path avoiding forbidden pairs of vertices (PAFP-V) involves finding a s-t path such that at most one vertex from any pair in $F_{\text{vertices}}$ is contained in the path.

The following Proposition is given in [23] and included in [24].

**Proposition 6** PAFP-V, and its variant PAFP-E, in which the set of forbidden pairs $F_{\text{edges}}$ consists of edges, are NP-complete.

**Definition 12 (Vertices Hierarchical Structure)** Consider two nodes $n_1, n_2 \in N$, if there exists a path from $n_1$ to $n_2$ we write $n_1 \prec_N n_2$. The set $F_{\text{vertices}}$ in PAFP-V, has the Vertices Hierarchical Structure (VHS) if for any two pairs, say $(n_1, n_2), (n_3, n_4) \in F_{\text{vertices}}$, there does not exist a path in $G$ such that $n_1 \prec_N n_3 \prec_N n_2 \prec_N n_4$.

VHS is first introduced in [39], EHS as follows is a similar structure on edges instead of vertices.

**Definition 13 (Edges Hierarchical Structure)** Consider two edges $e_1, e_2 \in E$, if there exists a path from $e_1$ to $e_2$ we write $e_1 \prec_E e_2$. The set $F_{\text{edges}}$ in PAFP-E has the Edges Hierarchical Structure (EHS) if for any two pairs, say $[e_1, e_2], [e_3, e_4] \in F_{\text{edges}}$, there does not exist a path in $G$ such that $e_1 \prec_E e_3 \prec_E e_2 \prec_E e_4$. 
The following Proposition is given in [39].

**Proposition 7** PAFP-V with its $F_{\text{vertices}}$ satisfying VHS can be solved in polynomial time.

### 2.1.3 Review of Intractable Problems

The problems introduced here are used for the NP-hardness proofs in this thesis.

A (directed) Hamiltonian path in a (directed) graph $G$ is a path that goes through each node exactly once. The s-t Hamiltonian path problem in a graph involves finding a directed Hamiltonian path where the starting point and the ending point, namely $s$ and $t$, are specified in the instance. All of these problems are NP-complete. However, the Hamiltonian path problem in a DAG can be solved in polynomial time (see comments on [GT39], the 39th problem under the category of graph theory, in [24]).

A Boolean formula is an expression involving

1. Boolean variables, i.e., the variables having either 1 or 0 as their values,

2. Binary operators $\land$, $\lor$, $\neg$, standing for the logical AND, OR, or Negation,

3. Parentheses that can alter the default precedence of operators: $\neg$ is higher than $\land$, which is higher than $\lor$.

A literal is a Boolean variable or a negated variable. A clause is a disjunction of one or more literals. A Boolean formula is in conjunctive normal form (CNF) if it is a conjunction of one or more clauses. It is in 3CNF if all the clauses have three literals.

A Boolean formula is satisfiable if some assignment to its variables makes the formula evaluate to 1. The SAT Problem is to test whether a Boolean formula is satisfiable. The 3SAT Problem is to test whether a Boolean formula in 3CNF is satisfiable. The One-In-Three 3SAT Problem is to test whether a Boolean formula in 3CNF is satisfiable and each clause contains one and only one literal that is assigned with value 1. It is well known that all these SAT problems are NP-complete ([24], Page 259). The NON-TAUTOLOGY Problem is to test whether there is a truth assignment that makes a Boolean formula false. The problem is also NP-complete ([L08], [24]).
2.1.4 Kambhampati and Nau’s Formalisms

We start this section with an adaptation of the definitions in [38], in order to facilitate a straightforward comparison between our results and their claims. The issue on negations is addressed in the subsequent section.

The planning language \( \mathcal{L} \) is a first-order language for the representation of planning problems. In \( \mathcal{L} \), a state is represented as a conjunction of ground and function-free atoms, i.e., positive literals.

**Definition 14 (State)** Any finite collection of ground atoms of \( \mathcal{L} \). That is, given a state \( t \) and an atom \( p \), the atom \( p \) is true in \( t \) iff \( p \) is contained in \( t \) iff \( \neg p \) is false in \( t \).

Hence, the definition uses the closed-world assumption in the sense that any atom not included in a state is assumed false in that state.\(^2\)

**Definition 15 (Step)** A 3-tuple in the form \( \langle a, \text{pre}(a), \text{post}(a) \rangle \), where \( a \) is the name of the step, the precondition \( \text{pre}(a) \) is a conjunction of literals that must hold in a state \( t \) so that the step can be executed in \( t \) and the postcondition \( \text{post}(a) \) is a conjunction of literals specifying how a state \( t \) changes over the execution of the step in \( t \). A positive literal in \( \text{post}(a) \) is asserted to be true in the state resulted from the execution of the step in the current state, whereas a negative literal is asserted to be false.

**Definition 16 (Codesignation constraint)** A syntactic expression of the form ‘\( u \approx v \)’ or ‘\( u \not\approx v \)’, where \( u \) and \( v \) are terms in \( \mathcal{L} \). With ‘\( u \approx v \)’ (or ‘\( u \not\approx v \)’), it is required that, for any ground substitution \( \theta \), \( \theta u = \theta v \) (or \( \theta u \neq \theta v \)).\(^3\)

**Definition 17 (Ordering constraint)** A syntactic expression of the form ‘\( a \prec b \)’ (that is, step \( a \) precedes step \( b \)).

**Definition 18 (Plans)** A partially ordered, partially instantiated plan \( P \) within a given \( \mathcal{L} \) is defined as a 4-tuple in the form \( \langle t_0, A, D, O \rangle \), where \( t_0 \) is the initial state; \( A \) is a set of steps; \( D \) is defined as a set of codesignation constraints on a set of terms \( P \) (i.e., the terms in \( t_0 \) and \( A \)); and \( O \) is defined as a set of ordering constraints.\(^4\)

---

\(^2\) This definition coincides with the definition of State in Event Systems.

\(^3\) Although a definition is provided, the concept of codesignation constraint does not appear elsewhere in [38].

\(^4\) See Paragraph 4 on Page 132 of [38].
A plan \( P \) is \textit{complete} if it is totally ordered and ground. That is, there exists unique total ordering over \( A \) that satisfies \( O \) and there exists a unique ground substitution \( \theta \) that satisfies \( D \). Otherwise, \( P \) is \textit{incomplete}.

\textbf{Definition 19 (Plan executability)} A complete plan \( P \) of \( n \) steps is executable if all states generated along the steps of \( P \), from initial \( t_0 \) to the second last state \( t_{n-1} \), satisfy the preconditions for steps from \( a_1 \) to \( a_{n-1} \), respectively.\(^5\)

Also, we say that the step \( a_i \) is \textit{executable} for \( 1 \leq i \leq n \). A complete plan \( P \) is only \textit{semi-executable}, if, for some maximal \( k \) such that \( k < n \), \( t_0 \) to state \( t_{k-1} \) satisfy the preconditions for steps from \( a_1 \) to \( a_{k-1} \), respectively. Also, we say that the step \( a_i \) is \textit{executable} for \( 1 \leq i \leq k \).

The concept \textit{situation} is similar, but not identical to, the concept \textit{state}. In fact, \textit{situations} are defined in terms of \textit{states}.

\textbf{Definition 20 (Situations in plans)} For any plan \( P \) (might be an incomplete one), the input situation (and output situation) of a given step \( a \) in \( P \), written as \( \text{in}(a) \) and \( \text{out}(a) \), characterizes the set of all states that are before (and after) \( a \) in the completions of \( P \). The situation \( \text{ini} \) and \( \text{fin} \) are associated with the plan \( P \) to specify the initial state and the set of final states in the completions of \( P \).\(^6\)

Note that, the truth value for a ground condition \( p \) is meaningful only in situations of complete and executable plans, where all situations are fully specified. That is, for any \( p \) and situation \( s \) in a complete and executable plan \( P \), either \( p \in s \) or \( \neg p \in s \), but not both. An incomplete plan \( P \), however, is associated with a set of completions. Only a subset of them (could be empty or include all completions) are actually executable. Hence, the truth value of a condition \( p \) in the final situations of different completions of \( P \) might vary. The truth value of \( p \) is not even defined in the final situation of an nonexecutable completion of \( P \).

Truth, and conditional truth, for complete plans are defined as follows.\(^7\)

\textbf{Definition 21 (Truth)} An atom \( p \) is true in the final situation of a complete plan \( P \) iff

\(^5\)See Paragraph 4-5 on Page 132-133 of [38].
\(^6\)See Page 133-134 of [38].
\(^7\)See Page 134 of [38], Paragraph 3.
Chapter 2. Background

1. **(Executability)** $P$ is executable;

2. **(Establishment)** $p$ is true in the initial situation (state) of $P$ or later is asserted by some step $a$ in $P$; and

3. **(Nondeletion)** $p$ is never deleted if it is initially true, or there exists a step $a_1$ that asserts $p$ and there does not exist another step $a_2$ that follows $a_1$ and deletes $p$.

**Definition 22 (Conditional truth)** An atom $p$ is conditionally true in the final situation of a complete plan $P$ iff

1. **(Establishment)** $p$ is true in the initial situation (state) of $P$ or later is asserted by some step $a$ in $P$; and

2. **(Nondeletion)** $p$ is never deleted if it is initially true, or there exists a step $a_1$ that asserts $p$ and there does not exist another step $a_2$ that follows $a_1$ and deletes $p$.

Note that the executability requirement is removed. More precisely, we might want to decide in $P$:  

- Truth $\mathcal{M}(p, \text{fin})$: $P$ is executable, and produces condition $p$ in situation $\text{fin}$. Since, for a complete plan $P$, a condition $p$ holds after its execution iff, if every step of the plan is executable (e.g., no step requires any precondition to execute), $P$ will produce $p$ in $\text{fin}$ (written as $\mathcal{C}(p, \text{fin})$) and every step is actually executable (written as $\bigwedge_{a \in P} \forall p_a \in \text{pre}(a) \mathcal{C}(p_a, \text{in}(a))$), we have

$$\mathcal{M}(p, \text{fin}) \equiv \left[ \mathcal{C}(p, \text{fin}) \land \bigwedge_{a \in P} \forall p_a \in \text{pre}(a) \mathcal{C}(p_a, \text{in}(a)) \right].$$

Note that, from the above definition on $\mathcal{M}(p, \text{fin})$, the negation of it, $\neg \mathcal{M}(p, \text{fin})$, obviously should be interpreted as that $P$ is not executable or $P$ does not produce $p$.

---

8 In general, the plan “$P$” should be used in the notations, but here we follow a close adaption to [38].
9 The formula corresponds to Equation 1 on Page 136 of [38].
10 Also, see Paragraph 4 on Page 134, and Equation 1 on Page 136 of [38].
Conditional Truth $C(p, fin)$: as stated in the above definition, assuming the preconditions of all of the steps along the path in the plan are removed ($P$ thus is certainly executable), $p$ is produced in $fin$.

The truth and the conditional truth, for incomplete plans are defined as follows.

**Definition 23 (Possible/Necessary truth)** An atom $p$ is possibly (necessarily) true in the final situation of some completion $P'$ (all completions $P'$s) of an incomplete plan $P$ iff

1. (**Executability**) $P'$ is executable;

2. (**Establishment**) $p$ is true in the initial situation (state) of $P$ or later is asserted by some step $a$ in $P'$; and

3. (**Nondeletion**) $p$ is never deleted if it is initially true, or there exists a step $a_1$ that asserts $p$ and there does not exist another step $a_2$ that follows $a_1$ and deletes $p$.

**Definition 24 (Possible/Necessary conditional truth)** An atom $p$ is possibly (necessarily) conditionally true in the final situation of some completion $P'$ (all completions $P'$s) of an incomplete plan $P'$ iff

1. (**Establishment**) $p$ is true in the initial situation (state) of $P$ or later is asserted by some step $a$ in $P'$; and

2. (**Nondeletion**) $p$ is never deleted if it is initially true, or there exists a step $a_1$ that asserts $p$ and there does not exist another step $a_2$ that follows $a_1$ and deletes $p$.

Note that the executability requirement is removed.

Consequently, a set of temporal reasoning decision problems can be proposed with respect to a given incomplete plan $P$. More precisely, we might want to decide in this incomplete plan $P$.\footnote{Once again, the plan “$P$” should be used in the notations, but here we follow a close adaption to [38].}
- **Necessary Truth** $\Box M(p, fin)$: every completion $P'$ of $P$ is executable, and $P'$ produces $p$ in $fin$. That is, for an incomplete plan $P$, a condition $p$ holds after an execution of every completion of $P$ iff (1) if every step in every completion is executable, every completion will produce $p$ in $fin$, and (2) every step in every completion is actually executable. Thus we have

$$\Box M(p, fin) \equiv \Box \left[ C(p, fin) \land \left[ \bigwedge_{a \in P', \forall p_a \in pre(a)} C(p_a, in(a)) \right] \right].$$

- **Possible Truth** $\Diamond M(p, fin)$: some completion $P'$ of $P$ is executable, and $P'$ produces $p$ in $fin$. That is, for an incomplete plan $P$, a condition $p$ holds after an execution of some completion of $P$ iff (1) IF every step in this completion is executable, this completion will produce $p$ in $fin$, and (2) every step in it is actually executable. Thus we have

$$\Diamond M(p, fin) \equiv \Diamond \left[ C(p, fin) \land \left[ \bigwedge_{a \in P', \forall p_a \in pre(a)} C(p_a, in(a)) \right] \right].$$

- **Necessary Conditional Truth** $\Box C(p, fin)$: assuming the preconditions of all steps are empty (every completion of $P$ is executable), every completion $P'$ produces $p$ in $fin$. That is, for an incomplete plan $P$, a condition $p$ conditionally holds after an conditional execution of every completion of $P$ iff if (1) every step in every completion is executable, every completion will produce $p$ in $fin$.

- **The Possible Conditional Truth** $\Diamond C(p, fin)$: assuming the preconditions of all steps are empty (every completion of $P$ is executable), some completion $P'$ produces $p$ in $fin$.

### 2.2 Situation Calculus

In this section, we give a brief overview of Situation Calculus, introduce a special class of theories known as Basic Action Theories, and justify a straightforward and concise approach to implementing Basic Action Theories as software programs using the logic programming language Prolog.

\[12\] The formula corresponds to Equation 2 on Page 136 of [38].

\[13\] The formula corresponds to Equation 4 on Page 136 of [38].
2.2.1 Concepts

Situation calculus is a logical language for representing actions and changes in a dynamical domain; it was first proposed by McCarthy and Hayes in 1969 [48]. The language $\mathcal{L}$ of situation calculus as stated by [59] is a second-order many-sorted language with equality.

Three disjoint sorts: action, situation, object (for everything else in the specified domain) are included in the language $\mathcal{L}$. For example, rain denotes the act of raining, and putdown$(x, y)$ denotes the act of agent $x$ putting block $y$ on the ground. A situation characterizes a sequence of actions in the domain. The constant situation $S_0$ is to denote the empty sequence of actions, whereas the function symbol $do$ is introduced to construct the term $do(a, s)$, denoting the successor situation after performing action $a$ (such as, in a car-washing scenario, clean) in situation $s$. The situation term $do(clean, do(spray, s))$ denotes the situation resulting from first spray water on the car and then clean the car body, which distinguishes itself from the other situation term $do(spray, do(clean, s))$. It is easy to see that, intuitively, a situation corresponds to a finite sequence of actions.

The binary predicate $\sqsubseteq$ specifies the order between situations. For example, $s \sqsubseteq s'$ stands for that the situation $s'$ can be reached by performing one or several actions from $s$. $s \sqsubseteq s'$ is an abbreviation of $s \sqsubseteq s' \lor s = s'$. In addition, a predicate $Poss(a, s)$ is applied to specify the legality of performing action $a$ in situation $s$. For example, $Poss(rain, s) \supset heavyCloudy(s)$ says that it is possible to rain only if the sky has heavy cloud.

The concepts of situations and their precedences can be captured axiomatically by four axioms as follows. They are called as foundational axioms, reflecting their primary role in providing the basic properties of situations.

**Definition 25** The set $\mathcal{D}_f$ consists of four foundational axioms:

\[
\begin{align*}
do(a_1, s_1) = do(a_2, s_2) & \supset a_1 = a_2 \land s_1 = s_2, \hspace{1cm} (2.1) \\
(\forall P).P(S_0) \land (\forall a, s)(P(s) \supset P(do(a, s))) & \supset (\forall s)P(s), \hspace{1cm} (2.2) \\
\neg s \sqsubseteq S_0, & \hspace{1cm} (2.3) \\
s \sqsubseteq do(a, s') & \equiv s \sqsubseteq s'. \hspace{1cm} (2.4)
\end{align*}
\]

In a model of $\mathcal{D}_f$, all situations can be represented by a tree. For example, a model depicted in Figure 2.1 (adopted from Figure 4.1 of [59]) contains $n$ (from $\alpha_1, \ldots, \alpha_n$) actions.
Definition 26 In a situation tree, executable situations are those who can be physically reached from the initial situation $S_0$. The predicate $\text{Poss}$ is used to characterize them:

$$\text{executable}(s) \overset{\text{def}}{=} (\forall a, s^*). \text{do}(a, s^*) \sqsubseteq s \supset \text{Poss}(a, s^*).$$

In a particular domain, the language might contain situation independent relations, like $\text{matchLocation}(\text{Toronto})$, and situation independent functions, like $\text{size}(\text{Plot2})$. However, in many of the more interesting cases, the values of relations and functions change between situations; accordingly, a relational fluent, or a functional fluent, in $\mathcal{L}$ is defined as a predicate, or a function, respectively, whose last argument is always a situation (e.g., $\text{captain}(\text{John}, \text{do}(\text{catchFever}, S_0))$ is a relational fluent, whereas $\text{weight}(\text{John}, \text{do}(\text{recover}, s))$ is a functional fluent).

A particular class of situation calculus theories called Basic Action Theories is specified in [59] (also briefly presented in the subsequent section), where it is shown that nice theoretical and computational properties hold for Basic Action Theories. For situation calculus at an introductory level, the reader is referred to [43] and [8], where each includes a chapter of situation calculus.

2.2.2 Basic Action Theories

A Basic Action Theory is a Situation Calculus theory whose satisfiability can be checked rather easily by virtue of the Relative Satisfiability Theorem (Theorem 4.4.6 in [59]),
whereas the entailment of regressable queries in the theory can be reduced to much simplified forms (see Theorem 4.5.5 in [59]). We leave the explanations of precondition axioms, successor state axioms, and unique names axioms to Chapter 6 and Chapter 7, where a particular theory for Petri nets, and for activity diagrams, is proposed, respectively. Also, in-depth general discussion can be found in Chapter 4 of [59].

\textbf{Definition 27} A Basic Action Theory $\mathcal{D}$ is a collection of several sets of axioms as follows:

\[ \mathcal{D} = \mathcal{D}_f \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \]

where

- $\mathcal{D}_f$ is the set of foundational axioms;
- $\mathcal{D}_{ss}$ is the set of successor state axioms for functional and relational fluents and
  - each relational fluent axiom is of the form
    \[ F(x_1, \ldots, x_n, do(a, s)) \equiv \Phi_F(x_1, \ldots, x_n, a, s) \]
    where $\Phi_F(x_1, \ldots, x_n, a, s)$ is uniform in $s$;
  - each functional fluent axiom is of the form
    \[ f(x_1, \ldots, x_n, do(a, s)) = y \equiv \Phi_f(x_1, \ldots, x_n, y, a, s) \]
    where $\Phi_f(x_1, \ldots, x_n, y, a, s)$ is uniform in $s$;
  - each functional fluent satisfies the consistency property (defined below);
- $\mathcal{D}_{ap}$ is the set of precondition axioms of the form
  \[ \text{Poss}(A(x_1, \ldots, x_n), s) \equiv \Pi_A(x_1, \ldots, x_n, s), \]
  where $A$ is an n-ary action function symbol and $\Pi_A(x_1, \ldots, x_n, s)$ is uniform in $s$;
- $\mathcal{D}_{una}$ is the action unique names axioms;
- $\mathcal{D}_{S_0}$ is a set of first-order sentences that are uniform in $S_0$.

\textbf{Definition 28} A functional fluent $f$ satisfies the consistency property if

\[ \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \models (\forall \bar{x}).(\exists y)\phi_f(\bar{x}, y, a, s) \land ((\forall y, y').\phi_f(\bar{x}, y, a, s) \land \phi_f(\bar{x}, y', a, s) \supset y = y') \]
and
\[ f(\vec{x}, do(a, s)) = y \equiv \phi_f(\vec{x}, y, a, s) \]
is the corresponding successor state axioms for \( f \) in \( D_{ss} \) (Definition 4.4.5 in [59]).

The concept of the uniform formulas is formally defined in Definition 4.4.1 of [59]. Intuitively, the uniformity requirement for the precondition axioms and successor state axioms assure that the qualification of an action and the properties in the successor situation are determined entirely by the current situation \( s \).

One important theoretical result with respect to Basic Action Theories is stated as follows. In Chapter 6 and Chapter 7, we show its technical importance towards proving the satisfiability of SCOPE, and SCAD.

**Proposition 8 (Relative Satisfiability)** A Basic Action Theory \( D \) is satisfiable iff \( D_{una} \cup D_{so} \) is (Theorem 4.4.7 in [59], and see [57] for the proof).

### 2.2.3 Prolog Implementations

A Prolog program is a set of logic clauses in the form

\[ L_1 \land \ldots \land L_m \supset A \]

where \( A \) is an atom and is called the head of the clause, and \( L_1, \ldots, L_m \), called the body of the clause, is a conjunction of literals.\(^1\) For example,

\[
\text{grandChild(charlie, alice)} :-
\text{parent(alice, bob),}
\text{parent(bob, charlie)}.
\]

can be read as: if Alice is a parent of Bob and Bob is a parent of Charlie, then Charlie is a grandchild of Alice. The body of a clause can be empty. For example, \text{male(bob)} is read as: Bob is a male. In Prolog, these clauses are called facts, whereas other clauses are called rules. The symbol \text{not} is used in Prolog for the logical negation \( \neg \), working in the manner of “negation-as-failure”. For example \text{not(female(bob))} is true iff \text{female(bob)} fails.

A first-order sentence

\[(\forall x_1, \ldots, x_n).P(x_1, \ldots, x_n) \equiv \phi,\]

\(^{14}\)In Prolog, \( "L_1 \land \ldots \land L_m \supset A" \) is written as \( "A : \neg L_1, \ldots, L_m" \).
where $P$ is a non-equality predicate and $\phi$ is a formula with free variables among $x_1, \ldots, x_n$, is a definition of $P$. A theory is definitional iff it consists of one definition for each non-equality predicate. The if-half of $P$ is

$$(\forall x_1, \ldots, x_n).\phi \supset P(x_1, \ldots, x_n).$$

One fundamental result in logic programming is Clark’s Theorem, which guarantees the soundness of implementing a Prolog program for a set of definitions, by, if possible, writing the sufficient conditions of all the definitions as Prolog clauses (see Chapter 3 of [44] and Chapter 5 of [59] for further reference).

Not all definitions satisfy the syntax for Prolog clause as stated above, in that case, Lloyd-Topor transformations should be applied. For example, the formula $B \land \neg(C \land D) \supset A$ can first be replace by $B \land (\neg C \lor \neg D) \supset A$, and then further replaced by $B \land \neg C \supset A$ and $B \land \neg D \supset A$. The reader is referred to Section 5.2 in [59] for the complete algorithm of Lloyd-Topor transformation.

The Implementation Theorem\footnote{Essentially, the theory should be definitions, see Theorem 5.3.4 of [59] for details.} (Given as Definition 4.5.1 in [59] and stated as follows), is a special version of Clark’s theorem for proving regressable queries in Basic Action Theories.

**Definition 29 (Regressable Formulas)** (See Definition 4.5.1 of [59]) A Formula $W$ is regressable iff

1. Each term of sort situation mentioned by $W$ has syntactic form $do([\alpha_1, \ldots, \alpha_n], S_0)$,\footnote{$do([\alpha_1, \ldots, \alpha_n], S_0)$ is an abbreviation of $do(\alpha_n, do(\alpha_{n-1}, \ldots, do(\alpha_1, S_0) \ldots))$.} for some $n \geq 0$, and $\alpha_1, \ldots, \alpha_n$ are actions;

2. For each atom $\text{Poss}(\alpha, \sigma)$ mentioned by $W$, $\alpha$ has the form $A(t_1, \ldots, t_n)$ for some $n$-ary action function symbol $A$;

3. $W$ does not quantify over situations;

4. $W$ does not mention the predicate symbol $\sqsubseteq$, nor does it mention any equality atom $\sigma = \sigma'$ for situation terms $\sigma$ and $\sigma'$.

**Definition 30 (Closed Initial Database)** (Definition 5.3.2 of [59]) Suppose $L_{\text{sitcalc}}$ has no functional fluents. An initial database $D_{S_0}$ of $L_{\text{sitcalc}}$ is in closed form iff
1. For every relational fluent \( F \) of \( \mathcal{L}_{\text{sitcalc}} \), \( \mathcal{D}_{S_0} \) contains exactly one sentence of the form \( F(\vec{x}, S_0) \equiv \Psi_F(\vec{x}, S_0) \), where \( \Psi_F(\vec{x}, S_0) \) is a first-order formula that is uniform in the situation term \( S_0 \), and whose free variables are among \( \vec{x} \).

2. For every non-fluent predicate symbol \( P \) of \( \mathcal{L}_{\text{sitcalc}} \), \( \mathcal{D}_{S_0} \) contains exactly one sentence of the form \( P(\vec{x}) \equiv \Theta_P(\vec{x}) \) where \( \Theta_P(\vec{x}) \) is a situation-independent first-order formula whose free variables are among \( \vec{x} \).

3. The rest of \( \mathcal{D}_{S_0} \) consist of the following equality sentences:

   (a) For each pair of distinct function symbols \( f \) and \( g \) of sort object, including constant symbols, \( f(\vec{x}) \neq g(\vec{y}) \).

   (b) For each function symbol \( f \) of sort object,

   \[
   f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \supset x_1 = y_1 \land \ldots, \land x_n = y_n.
   \]

   (c) For each term \( t \) of sort object other than the variable \( x \) of sort object, \( t[x] \neq x \).

   (d) For each term \( \tau \) of sort action other than the variable \( a \) of sort action, \( t[a] \neq a \).

In other words, the remaining sentences of \( \mathcal{D}_{S_0} \) are precisely the unique names axioms for the sort object required by Clark’s theorem, together with the schema \( t[a] \neq a \) for sort action, also required by this theorem.

For the purpose of proving regressable sentences, the definitions for \( \text{Poss} \) and for fluents, together with certain unique names axioms, are all that are needed.

**Definition 31** (Definition 5.3.4 of [59]) Let \( \mathcal{D} \) be a Basic Action Theory in a situation calculus language \( \mathcal{L}_{\text{sitcalc}} \) with the following properties:

1. \( \mathcal{L}_{\text{sitcalc}} \) has no functional fluents, and it has just finitely many relational fluents and action function symbols.

2. \( \mathcal{D}_{S_0} \) is in closed form. Let \( \mathcal{D}_{\Delta S_0}^\Delta \) be \( \mathcal{D}_{S_0} \) with all the equivalences for relational fluents removed. Therefore, \( \mathcal{D}_{\Delta S_0}^\Delta \) consists of definitions for all non-fluent predicate symbols of \( \mathcal{L}_{\text{sitcalc}} \), together with certain unique names axioms.

   - Let \( \mathcal{D}_{\text{ap}}^\Delta \) consist of the definition (see Formula 5.2 of [59]) for \( \text{Poss} \).

   - Let \( \mathcal{D}_{\text{ss}}^\Delta \) consist of the definitions (see Formula 5.3 of [59]), one for each relational fluent \( F \).
• Let $\mathcal{D}_{\text{unsit}}$ consist of the two unique names axioms for situations in $\Sigma$, namely:
  
  - $S_0 \neq \text{do}(a, s)$,
  - $\text{do}(a_1, s_1) = \text{do}(a_2, s_2) \supset a_1 = a_2 \land s_1 = s_2$,
  
  together with all instances of the schema $t[s] \neq s$, for every term of sort situation other than $s$ itself that mentions the situation variable $s$.

Then $\mathcal{D}$ and $\mathcal{D}_{S_0} \cup \mathcal{D}_{\text{ss}} \cup \mathcal{D}_{\text{ap}} \cup \mathcal{D}_{\text{una}} \cup \mathcal{D}_{\text{unsit}}$ are equivalent for regressable sentences in the following sense: Whenever $G$ is a regressable sentence of $\mathcal{L}_{\text{sitcalc}}$,

$$\mathcal{D} \models G$$

iff

$$\mathcal{D}_{S_0} \cup \mathcal{D}_{\text{ss}} \cup \mathcal{D}_{\text{ap}} \cup \mathcal{D}_{\text{una}} \cup \mathcal{D}_{\text{unsit}} \models G.$$

**Proposition 9 (Implementation Theorem)** Let $\mathcal{D}$ be a Basic Action Theory satisfying certain conditions (see the above definition), and let $\mathcal{P}$ be the Prolog program obtained from the following sentences, after transforming them by the Lloyd-Topor rules:

- For each definition of a non-fluent predicate of $\mathcal{D}_{S_0}$ of the form $P(\vec{x}) \equiv \Theta_P(\vec{x})$:
  
  $$P(\vec{x}) \supset \Theta_P(\vec{x});$$

- For each equivalence in $\mathcal{D}_{S_0}$ of the form $F(\vec{x}, S_0) \equiv \Psi_F(\vec{x}, S_0)$:
  
  $$\Psi_F(\vec{x}, S_0) \supset F(\vec{x}, S_0);$$

- For each action precondition axiom of $\mathcal{D}_{\text{ap}}$ of the form $\text{Poss}(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$:
  
  $$\Pi_A(\vec{x}, s) \supset \text{Poss}(A(\vec{x}), s);$$

- For each successor state axiom $F(\vec{x}, \text{do}(a, s)) \equiv \Phi_F(\vec{x}, a, s)$ of $\mathcal{D}_{\text{ss}}$:
  
  $$\Phi_F(\vec{x}, a, s) \supset F(\vec{x}, \text{do}(a, s)).$$

Then $\mathcal{P}$ provides a correct Prolog implementation of the Basic Action Theory $\mathcal{D}$ in the sense that whenever Prolog succeeds on a regressable sentence, that sentence is entailed by $\mathcal{D}$, and whenever it fails on such a sentence, $\mathcal{D}$ entails the negation of this sentence.
2.3 Process Specification Languages

2.3.1 Introduction

As a modular set of theories in the language of first-order logic, the Process Specification Language (PSL) [29, 31] has been designed to facilitate correct and complete exchange of process information.\(^{17}\) It has been adopted by the Semantic Web Services Language (SWSL) Committee of Semantic Web Services Initiative (SWSI)\(^ {18}\) to specify the model-theoretic semantics of Semantic Web Services Ontology (SWSO) [5], one of the two major components within Semantic Web Services Framework (SWSF) [4].

In order to formally specify a broad variety of properties and constraints on complex activities, we need to explicitly describe and quantify over complex activities and their occurrences. Within the PSL Ontology, complex activities and occurrences of activities are elements of the domain and the occurrence_of relation is used to capture the relationship between different occurrences of the same activity.

A second requirement for formalizing the queries is to specify composition of activities and occurrences. The PSL Ontology uses the subactivity relation to capture the basic intuitions for the composition of activities. Complex activities are composed of sets of atomic activities, which in turn are either primitive (i.e. they have no proper subactivities) or they are concurrent combinations of primitive activities (see Appendix A.3 and Section 5.2 of [6] for the intuitions about aggregating concurrent primitive activities as atomic activities).

Corresponding to the composition relation over subactivities, subactivity_occurrence is the composition relation over activity occurrences. Given an occurrence of a complex activity, subactivity occurrences are occurrences of subactivities of the complex activity.

Within the PSL Ontology, concurrency is represented by the occurrence of concurrent activities rather than concurrent activity occurrences, so that the interaction between these concurrent activities can be captured as the effects of a single activity (see Section 5.2 of [6] for more discussions on concurrency in PSL). For example, if activity \(a_1\) and \(a_2\) can occur concurrently, we will define a concurrent activity \(conc(a_1, a_2)\). Any occurrence of \(conc(a_1, a_2)\) thus denotes a concurrent occurrence of \(a_1\) and \(a_2\). We use the following

---

\(^{17}\) PSL has been accepted as an International Standard (ISO 18629) within the International Organization of Standardization. The full set of axioms in the Common Logic Interchange Format is available at http://www.mel.nist.gov/psl/ontology.html

\(^{18}\) http://www.swsi.org
relation to generalize the notion of occurrence to include any atomic activity that is a subactivity of the activity that occurs:

\[(\forall s, a)\ atocc(s, a) \equiv (\exists a_1)\ atomic(a_1) \land occurrence\_of(s, a_1) \land subactivity(a, a_1).\]

That is, any occurrence of activity \(a\) is a component of some occurrence \(s\) of some concurrent activity \(a_1\). Finally, we need some way to specify ordering constraints over the subactivity occurrences of a complex activity. The PSL Ontology uses the \(soo\_precedes(s_1, s_2, a)\) relation to denote that subactivity occurrence \(s_1\) precedes the subactivity occurrence \(s_2\) in occurrences of the complex activity \(a\).

The models of the axioms of the PSL Ontology have been characterized up to isomorphism [29]. A fundamental structure within these models is the occurrence tree, whose branches are equivalent to all discrete sequences of occurrences of atomic activities in the domain. Nodes of the occurrence tree are activity occurrences and are referred to as arboreal occurrences (see Appendix A.2.1 for the informal semantics of “arboreal” and Appendix A.2.2 for the axioms specifying the concept). In Situation Calculus, nodes in the situation trees denote situations (i.e., sequences of action occurrences from the initial situation \(S_0\)) instead of action occurrences. However, if we interpret activity occurrences to be situations, it is obvious that the occurrence trees are closely related to the situation trees.

Although occurrence trees characterize all sequences of activity occurrences, not all of these sequences will be physically possible within a given domain. We therefore consider the subtree of the occurrence tree that consists only of possible sequences of activity occurrences, which we refer to as the legal occurrence tree. The \(legal(o)\) relation specifies that the atomic activity occurrence \(o\) is an element of the legal occurrence tree. It should be noted that, if an activity occurrence is legal, all earlier activity occurrences in the occurrence tree are also legal (see Axiom 13 in Appendix A.2.2).

The basic structure that characterizes occurrences of complex activities within models of the ontology is the activity tree, which is a legal subtree of the occurrence tree that consists of all possible sequences of atomic subactivity occurrences;\(^{19}\) the relation \(root(s, a)\) denotes that the subactivity occurrence \(s\) is the first activity occurrence of an activity tree for \(a\). Elements of the tree are ordered by the \(soo\_precedes\) relation; each branch of an activity tree is a linearly ordered set of occurrences of subactivities of the

\(^{19}\)Nodes in activity trees also are activity occurrences, since they are subtrees of occurrence trees.
complex activity. In addition, there is a one-to-one correspondence between occurrences of complex activities and branches of the associated activity trees.

In a sense, an activity tree is a microcosm of the occurrence tree, in which we consider all of the ways in which the world unfolds in the context of an occurrence of the complex activity. Different subactivities may occur on different branches of the activity tree – different occurrences of an activity may have different subactivity occurrences or different orderings on the same subactivity occurrences. This distinction plays a key role in the specification of the entailment problems in this thesis.

To recap, the key terms in PSL core theories are:

- **activity**($a$): an activity type that can be repeated;
- **occurrence_of**($o$, $a$): an actual instantiation of activity $a$, where $o$ is an object of activity occurrence activity_occurrence($o$);
- **subactivity**($a_1$, $a_2$): $a_2$ contains $a_1$, i.e., $a_1$ is subactivity of $a_2$;
- **primitive**($a$): the minimal element in the ordering of subactivity;
- **conc**($a_1$, $a_2$): $a_1$ and $a_2$ are concurrent activities;
- **atomic**($a$): $a$ is primitive or a concurrent activity;
- **earlier**($s_1$, $s_2$): activity occurrences $s_1$ and $s_2$ are on the same branch of the tree and $s_1$ is closer to the root of the tree than $s_2$;
- **holds**($f$, $s$): the fluent $f$ is true immediately after the activity occurrence $s$;
- **prior**($f$, $s$): the fluent $f$ is true immediately before the activity occurrence $s$;
- **legal**($s$): an legal activity occurrence $s$ is an element of a legal occurrence tree;
- **min_precedes**($s_1$, $s_2$, $a$): the atomic subactivity occurrences $s_1$ and $s_2$ are atomic subactivity occurrences of a complex activity $a$, where $s_1$ is closer to some root occurrence in the minimal activity tree than $s_2$;
- **root**($s$, $a$): atomic subactivity occurrence $s$ is the root occurrence of an activity tree for $a$;
- **leaf**($s$, $a$): atomic subactivity occurrence $s$ is the leaf occurrence of an activity tree for $a$. 
In addition, relevant PSL axioms are presented in Appendix A.

2.3.2 Modularity

Within the ontology, all core theories of PSL are consistent extensions of a theory referred to as PSL-Core ($T_{pslcore}$), which introduces the basic ontological commitment to a domain of activities, activity occurrences, time points, and objects that participate in activities. Additional core theories capture the basic intuitions for the composition of activities, and the relationship between the occurrence of a complex activity and occurrences of its subactivities (Figure 2.2). More precisely, $T_{pslcore}$ axiomatizes the fundamental intuitions for describing a process: Activities in a process might not occur, or occur multiple times; Each activity occurrence or object is associated with a unique begin time point and end time point. Time points are linearly ordered. Concepts of occurrence trees are introduced in $T_{occtree}$. For a given set of activities, each branch in the corresponding occurrence tree coincides with a sequence of their occurrences. Every sequence has an initial occurrence. A branch is a legal one only if the sequence is possible with respect to the constraints on the activities in the domain. $T_{subactivity}$ captures the intuition that activities can be composed ones, whereas $T_{atomic}$ axiomatizes the intuitions about occurrence of concurrent activities. $T_{complex}$, together with $T_{actocc}$, defines the relationship between an occurrence of a complex activity and the occurrences of its subactivities.

In [30], using the notion of relative interpretations and definable models, Reiter’s version of Situation Calculus and PSL is compared. In particular, it is shown that the models of a union of the foundational axioms of Situation Calculus ($D_f$) and the action closure assumption (ACA), are elementarily equivalent to the models of the unions of $T_{pslcore}$, $T_{occtree}$, and a first-order axiom schema; meanwhile, subactivities and complex activities in PSL are not definable in Situation Calculus.

2.4 Petri Nets

Petri nets and their markings are defined in Section 2.4.1, whereas several dynamical properties of Petri nets and structurally restricted subclasses of Petri nets are introduced in Section 2.4.2.
2.4.1 Basic Definitions

**Definition 32** A Petri net (PN) is a pair \((N, M_0)\) where \(N\) is a triple \((P, T, F)\) such that

- \(P\) is a finite set of node elements called places;
- \(T\) is a finite set of node elements called transitions;
- \(F \subseteq (P \times T) \cup (T \times P)\) consists of ordered pairs;

and \(M_0\) is the initial marking, a mapping in the form \(M: P \rightarrow \mathbb{N}\), indicating the initial assignment of a non-negative integer \(k\) to each place \(p\) in \(P\). (In this case, we say that the place \(p\) is marked with \(k\) tokens.)

In general, any marking \(M\) for \(N\) in PN is defined as a vector \((M(p_1), \ldots, M(p_m))\) where \(p_1, \ldots, p_m\) is an enumeration of \(P\) and \(M(p_i)\) tokens are assigned to node \(p_i\), for all \(i\) such that \(1 \leq i \leq m\).

The elements in \(P \cup T\) are generically called nodes of PN. Given a node \(u \in PN\), the set \(\cdot u = \{v|(v, u) \in F\}\) is the pre-set of \(u\), where each \(v\) is the input of \(u\), and the set \(u^* = \{v|(u, v) \in F\}\) is the post-set of \(u\), where each \(v\) is the output of \(u\). In any marking \(M\), a place \(p\) is marked if \(M(p) > 0\). A transition \(t\) is enabled in \(M\) if every place in
**t** is marked in $M$. An enabled transition in $M$ can occur (fire) and there may exist several enabled transitions in $M$. One of them will actually fire, leading to the successor marking $M'$.

**Definition 33** A marking transition from $M$ to $M'$ due to the firing of $t$ (written as $M \xrightarrow{t} M'$) is defined by

$$M'(p) = \begin{cases} 
M(p) - 1 & \text{if } p \in \text{•}t \text{ and } p \notin \text{•}t \\
M(p) + 1 & \text{else if } p \notin \text{•}t \text{ and } p \in \text{•}t \\
M(p) & \text{otherwise}
\end{cases}$$

for every place $p$.

Graphically, a Petri Net PN can be represented by a bipartite graph, where each place is represented by a circle, each transition is represented by a rectangle, the flow relation of the Petri net $F$ is represented by arcs from places to transitions or from transitions to places, and $k$ black dots will be placed into the circle for place $p$ if it is marked with $k$ tokens at $M$. Figure 2.3 is a PN example where transitions $t_1$ and $t_2$ can be interpreted as switch on and switch off, respectively.

### 2.4.2 Petri Nets Properties

The concept of reachable marking is essential to the description and analysis of all properties (such as reachability, k-boundedness, and reversibility, as defined below) of Petri nets.
Definition 34 (Reachable marking) The transition (firing) sequence $\sigma = t_1, ..., t_m$ is a finite sequence of transition nodes in $T$; a $\sigma$ is an occurrence sequence of a given Petri net $PN = (N, M_0)$ if $M_0 \xrightarrow{t_1} M_1$, ..., $M_{m-1} \xrightarrow{t_m} M_m$ (written as $M \xrightarrow{\sigma} M_m$). Also, we write $M \xrightarrow{\sigma} M'$ and call $M'$ reachable from $M$ if there exists a firing sequence $\sigma$ such that $M \xrightarrow{\sigma} M'$. The set of all markings reachable from $M$ is denoted by $R(N, M)$.

Definition 35 (Reachability) The reachability problem for Petri nets is the problem of finding, given a Petri net $(N, M_0)$ and a marking $M$ in it, if $M \in R(N, M_0)$.

The $k$-boundedness, the liveness, and the reversibility are three of the most important basic behavioral properties of Petri nets. Informally, these properties involve checking whether the number of tokens in places in a given Petri net will never exceed $k$, whether the Petri net is always firable, and whether, from any current marking, the Petri net can always move back to its initial marking through some transition sequence. As examples, Figure 2.3 (adopted from Figure 17-h of [50]) is 1-bounded, live and reversible, Figure 2.4 (adopted from Figure 17-a of [50]) is not 1-bounded, non-live and irreversible (the
transition $t_1$ can never be fired, whereas the place $p_1$ is not bounded), whereas Figure 6.1 (adopted from Figure 17-d of [50]) is 1-bounded, non-live and irreversible.

**Definition 36 (k-boundedness)** Given a Petri net $(N, M_0)$, the k-boundedness problem involves checking whether the number of tokens in each place does not exceed the integer $k$ for any marking reachable from $M_0$. A 1-bounded Petri net is also said to be “1-safe”.

**Definition 37 (Liveness)** A Petri net $(N, M_0)$ is live if, for any marking $M \in R(N, M_0)$ and any transition $t$, there exists another marking $M'$, such that $M \xrightarrow{*} M'$, i.e., $M'$ is reachable from $M$, and $t$ is enabled in $M'$.

**Definition 38 (Reversibility/Cycleness)** A Petri net $(N, M_0)$ is reversible if $M_0$ is reachable from each $M \in R(N, M_0)$.

Extra properties of coverability and persistency are defined as follows.

**Definition 39 (Coverability)** A Petri net $(N, M_0)$ is coverable if there exists a marking $M' \in R(N, M_0)$ such that for any $p \in P$, $M'(p) \geq M_0(p)$.

**Definition 40 (Persistency)** A Petri net $(N, M_0)$ is persistent if, for any $t_1 \in T$ and $t_2 \in T$, which are both enabled at any $M \in R(N, M_0)$, $t_2$ should still be enabled in $M'$ where $M \xrightarrow{t_1} M'$.

Given that Petri nets as a modeling language have rich expressive power, evaluating properties of a Petri net is often computationally expensive, if possible. However, the difficulty is largely reduced for certain subclasses of Petri Nets where their graphical structures are highly constrained. The properties of the subclasses of Petri nets defined as follows have been thoroughly studied.

S-systems are systems whose transitions have one input place and one output place whereas T-systems are systems whose places have one input transition and one output transition:

**Definition 41** A Petri net $PN = (N, M_0)$, where $N$ is $(P, T, F)$, is an S-system if $|\bullet t| = |t^*| = 1$ for every $t \in T$.

**Definition 42** A Petri net $PN = (N, M_0)$, where $N$ is $(P, T, F)$, is a T-system if $|\bullet p| = |p^*| = 1$ for every $p \in P$. 
Conflict-free systems are generalizations of T-systems. Each place \( p \) in a conflict-free Petri net is either an input for at most one transition, or all such transitions enter the same place \( p \) to construct a so-called self-loop. As such, a token in any place of such systems will not encounter potential conflicts in choosing which transition to go.

**Definition 43** A Petri net \( PN = (N, M_0) \), where \( N \) is \( (P, T, F) \), is a conflict-free system if \( p^* \subseteq \mathcal{C} p \) for every place \( p \) with more than one output transition.

It can be shown that, given a Free-choice Petri net, for any two transitions sharing one common input place, there does not exist a marking such that enabling one transition in the Petri net will disable the another. Thus, which transition to fire can be chosen freely.

The behavioral properties of Free-choice systems are extensively studied in [13].

**Definition 44** A Petri net \( PN = (N, M_0) \), where \( N \) is \( (P, T, F) \), is a free-choice system if \((p, t) \in F\) implies \( \mathcal{C} t \times p^* \subseteq F \) for every place \( p \) and every transition \( t \).

Later we will show, in Section 6.5.1, that all of these above properties or subclasses can be represented easily in SCOPE.

### 2.5 UML Activity Diagrams

The Unified Modeling Language (UML) [54] was introduced mainly as a standard formalism for the design and analysis of software systems. In fact, ranging from enterprise information system development to distributed web service specification, broad applications of UML diagrams have been found. UML includes thirteen different types of diagrams. Among them, class diagrams, component diagrams, composite structure diagrams, deployment diagrams, package diagrams, and object diagrams are used to model structural information of systems, i.e., to specify how objects in a given system are related. Meanwhile, activity diagrams, communication diagrams, interaction overview diagrams, sequence diagrams, state machine diagrams, timing diagrams, and use case diagrams are used to capture the behavioral properties of systems.

Activity diagrams in particular are employed to exhibit the flow of control between the sequential or concurrent actions within a complicated activity. As a graphical modeling tool, activity diagrams are powerful in both constructing executable dynamic systems, and modeling dynamic aspects of existing systems. Ideally, activity diagrams can be
used as an efficient visual medium to facilitate reliable communication between different parties who are associated with the systems.

An activity diagram (see Figure 7.2 for an example of activity diagram) usually contains up to seven different nodes, which are connected by edges:

- actions, they can not be further decomposed and are represented as round boxes;
- concurrent flows of control between actions are captured by synchronization bars of horizontal or vertical dark lines to specify the forking and joining (a fork bar has one edge entering it and two or more edges leaving it, whereas a join node has one edge leaving it and two or more edges entering it);
- branching and merging of alternative choices of operations are represented as diamonds (a branch bar has one edge entering it and two or more edges leaving it, whereas a merge node has one edge leaving it and two or more edges entering it);
- Additionally, Initial nodes and final nodes are circles. In general, there exist one Initial node and one final node in an activity diagram.
Chapter 3

Dual Temporal Projection Problems

3.1 Introduction

Temporal projection problems have broad applications in a variety of different domains, from manufacturing process control in Operational Research to the Semantic Web Services in AI Knowledge Representation and Reasoning. The problems involve reasoning about the consequences of events in dynamical systems. In particular, the Possible Truth problems are those temporal projection problems that decide the possible consequences of events, (i.e., the possible settings of the literal truth value assignments representing the state of the world resulted from the occurrences of the events), whereas the Necessary Truth problems are the ones that decide the necessary consequences of events, (i.e., the necessary settings of the literal truth value assignments).

In the real world, causes and effects of events can often be observed without ambiguity. Hence the cause-and-effect relationships between events in a dynamical system model are usually characterized with complete knowledge. However, due to the uncertainty or choices in various circumstances, the order in which events will occur can only be partially specified. Since a partial order could potentially involve a set of exponentially many total orders, determining the consequence of events is thus difficult. The temporal projection problems in general are computationally intractable [12]. But in the next chapter, we will demonstrate that this intractability is built from a wide variety of sources aside from partial orderings.

One earlier framework of dynamical systems was introduced by Dean and Boddy [12] to facilitate the analysis of computational properties of temporal projection problems. In their representation, events are partially ordered, and the framework for cause-and-effect
relationships is based on the propositional STRIPS representation. The concept of Event Systems was first introduced in [53], which is a follow-up to the framework proposed in [12] and provides a refinement to the notions in [12]. An Event System is a simple logical construction of a dynamical system: it sets truth values of a set of propositional literals as initial conditions, and includes a set of partially ordered actual events. Meanwhile, in order to occur legally, an event must satisfy certain preconditions in the form of a set of propositional literals, and an occurrence of an event results in changes to the assignment of truth values in the current setting of the literals in the system. As a matter of fact, the relationship between the Possible Truth problems and the Necessary Truth problems has been investigated by several researchers, leading to numerous, but sometimes confusing, results (see [10], [19], [38], [52] and [53]). For example, the so-called Modal Truth Criterion (MTC) stated on Page 340 of [10] is claimed as necessary but insufficient in [52] and not to be necessary in [19], whereas in [53], it is believed that circular argument exists in the application of MTC to prove plan coherence (on Page 143 of [53]).

In this chapter, the Possible Truth problems, the Necessary Truth problems, and the relationship between them are investigated in the context of Situation Calculus. More precisely, by using the language of Situation Calculus to define a logical theory of Event Systems, we define a general version of goal situations, whereas the general Possible Truth problem involves checking whether a certain statement holds at some goal situation and the general Necessary Truth problem involves checking if the statement holds at all goal situations. Starting from a given problem (either on possibility or on necessity), we are thus able to further elaborate three related problems: the negation of the problem, the complement of the problem, and the complement of the negation of the problem. We establish four sets of equivalences on these problems between the version of possibility and necessity (in Section 3.3). In essence, these equivalences justify the so-called modal duality, that is, for any version of the equivalences, there exists a goal situation where the statement holds if and only if it is not the case that the statement is false in all goal states. As shown in Section 3.5, these modal equivalences clarify the results on the duality of the possible truth and the necessary truth problems introduced in [38].

In Section 3.4, we use four tables to summarize the computational complexity of four sets of dual problems derived from the work of Nebel and Bäckström [53], Chapman [10], Dean and Boddy [12], and Kambhampati and Nau [38]. Often, we obtain these results by observing that, (1) if a given problem can be solved in polynomial time, then its complementary version can be solved in polynomial time too (see page 29 of [24]); (2) if
a given problem is NP-complete, then its complementary version is co-NP-Complete (see page 156-157 of [24]). Those results which are directly adopted from the above papers will be explicitly indicated.

Using modal-logic-like operators, in [38], it is claimed that the duality does not hold for the statement in the form “\(M(p, \text{fin})\)”, which is a conjunction of the executability of Event System and the holdness of the given condition \(p\). However, when the negation of the statement is encountered (in the form “\(\neg M(p, \text{fin})\)”) in their argument, they implicitly interpret it as a negation to the holdness alone, with the executability part excluded. In Section 3.5, we show that, even using the operators introduced in [38], we still are able to establish the duality between the two problems, by adjusting the negation to be applied to the whole statement. Also in this section, we use our Situation Calculus specified concepts to explain why the Partial Truth problems, as proposed by [38], are claimed as the dual of the Possible Truth problems.

### 3.2 Motivations and Correctness of the Axiomatization

In the next section, we first show how to create an action theory \(D_{es}\) from a given Event System \(\Theta\). Observe that any causal rule in an Event System is composed of an event type, a set of preconditions, a set of added conditions, and a set of deleted conditions. It is straightforward to see how to generate Action Precondition Axioms, Effect Axioms (thus Successor State Axioms) for a Basic Action Theory \(D_{es}\), from the causal rules in \(\Theta\).

The main motivation to axiomatize Event Systems is to facilitate analytical investigation of complicated reasoning problems and the relations between these problems. For example, in the context of \(D_{es}\), we present a formal discussion on the definition of negation on expressions and thus clarify a central concept in the results of [38] that is quite confusing.

Accordingly, we focus on using \(D_{es}\) to specify particular goal situations that are generated through performing sequences of events from the initial situation \(S_0\). More precisely, we (1) enrich \(D_{es}\) by introducing the abbreviations of \(Perm(s)\), a situation reflects a permutation of actual events, \(POrder(s)\), a situation reflects that the constraints of partial orders are satisfied, and \(Admi(s)\), a situation reflects that the actual events are admissi-
ble; (2) use these abbreviations to capture the correspondence between an (admissible) complete sequence in an Event System $\Theta$ and its corresponding situation in $\mathcal{D}_{es}$ (Theorem 1 and Theorem 2); and (3) use these abbreviations to define and to prove correctness of our definition of classical temporal projection problems in $\mathcal{D}_{es}$ (Theorem 3 and Theorem 4). Multiple applications of $\mathcal{D}_{es}$ are then demonstrated in the subsequent Section 3.4 and Section 3.5.

### 3.3 A Theory of Event Systems

In this section, we build a Situation Calculus action theory $\mathcal{D}_{es}$ from a given Event System $\Theta$. For each $p \in \mathcal{P}$ in $\Theta$, a relational fluent $F_p$, which takes one argument of sort situation, is created. Notice that in $\Theta$, the initial state $I$, together with any state $State$ (recall Definition 6) that results from an actual event sequence from the initial state $I$, is a subset of $\mathcal{P}$. Given a state $State$ in $\Theta$ and a situation $s$ in $\mathcal{D}_{es}$ that corresponds to $State$ in the sense that $s$ represents a sequence of events that changes the state from $I$ to $State$, a condition $p$ holds at a state $State$ iff the fluent $F_p$ holds in the situation $s$, i.e., $p \in State$ iff $F_p(s)$ holds. The negation of $F_p(s)$ (i.e., $\neg F_p(s)$) holds, otherwise.

For each event type $t \in \mathcal{T}$, an action function $A$ is introduced in $\mathcal{D}_{es}$. In addition, a formula $\Pi_A(s)$ that is uniform in $s$ is also defined. For any $r = \langle t, \varphi, \alpha, \delta \rangle$, $\Pi_A(s)$ contains a conjunction of all conditions in the set $\varphi$. When there exist multiple applicable rules for $t$ (i.e., say there exists another $r'$ such that $r' = \langle t, \varphi', \alpha', \delta' \rangle$), $\Pi_A(s)$ is then a disjunction of conjunctions of multiple sets of preconditions. Hence, the action precondition axiom for action $A$ is that, at any situation $s$, $A$ is possible to occur iff $\Pi_A(s)$ holds:

$$Poss(A, s) \equiv \Pi_A(s).$$

For each condition $p \in \mathcal{P}$, a successor state axiom is introduced of the form:

$$F_p(do(a, s)) \equiv \Phi_{F_p}(a, s)$$

where the fluent $F_p$ corresponds to $p$ and $\Phi_{F_p}(a, s)$ is uniform in $s$. The formula $\Phi_{F_p}(a, s)$ offers a complete characterization of the value $F_p$ in the successor state $do(a, s)$, which is the result from performing action $a$ in situation $s$. The fluent $F_p$ (corresponds to a

---

1Note that, in an Event System, the set of preconditions $\varphi$, the set of added conditions $\alpha$, and the set of deleted conditions $\delta$, all contain atoms. This representation arrangement coincides with the one in the action language defined in [61], the Chapter of Planning Problem.
condition \( p \) in \( \Theta \)) is true after performing \( a \) (corresponds to an event type \( t \) in \( T \) of \( \Theta \)), iff before performing \( a \),

- there exists a causal rule \( r' = \langle t, \varphi', \alpha', \delta' \rangle \) in \( R \) such that
  - for each condition \( p_i \) in \( \varphi' \), its corresponding fluent \( F_{p_i} \) holds at situation \( s \) (i.e., \( F_{p_i}(s) \) holds), and
  - \( p \in \alpha' \).
- or,
  - \( F_p \) holds at situation \( s \) (i.e., \( F_p(s) \) holds), and
  - there does not exist a causal rule \( r'' = \langle t, \varphi'', \alpha'', \delta'' \rangle \) in \( R \) such that \( p \in \delta'' \).

In summary, we can transform an Event System \( \Theta \) into a Situation Calculus Basic Action Theory \( D_{es} \):

**Definition 45** A Basic Action theory \( D_{es} \) is a collection of several sets of axioms as follows:

\[
D_{es} = D_f \cup D_{es,ss} \cup D_{es,ap} \cup D_{es,una} \cup D_{es,S0}
\]

where \( D_{es,S0} \) is a set of first-order sentences that are uniform in \( S_0 \). In particular, if a condition \( p \in \mathcal{I} \) then \( p(S_0) \in D_{es,S0} \), otherwise, we have \( \neg p(S_0) \in D_{es,S0} \).

It should be noted that the constructed action theory \( D_{es} \) is a partial axiomatization of Event Systems as defined in Section 2.1.1: it includes the Causal Structures (Definition 1) and the set of conditions for initial state \( \mathcal{I} \), but has yet to cover the actual events (\( \mathcal{E} \) and the partial orders on these actual events \( \mathcal{O} \)). As shown later in this section, in Situation Calculus, it is more appropriate to represent them as abbreviations to characterize certain situations in \( D_{es} \).

Among all situations in \( D_{es} \), normally we are interested only in certain subsets of all situations. We can use a formula \( \text{GoalSit}(s) \), where \( s \) is the only free variable, to characterize these situations. For example, we might only be interested in all executable situations (i.e., those situations that result from performing a sequence of physically realizable actions from initial situations).\(^2\) For all of these special situations, we want to check whether certain properties in the form of \( \text{Statement}(s) \), where \( s \) is the only

---

\(^2\)See also Definition 26.
free variable, are always true or false. For example, we might want to see whether certain fluent \( F_p \) is always true in all executable situations \(((\forall s)(executable(s) \supset F_p(s)))\).

Another class of queries concerns existentially quantified sentences over these particularly specified situations. For example, one might want to see whether certain fluent \( F_p \) can be true in some executable situation \(((\exists s)((executable(s) \land F_p(s)))\).

In general,

- universally quantified problems (i.e., on necessity) include:

  **Generic Necessary Truth:** The entailment in the form of
  \[ \mathcal{D}_{es} \models (\forall s)(GoalSit(s) \supset Statement(s)), \]
  which is simply the verifications of universal quantification on \( Statement(s) \);

  **Generic Necessary Falseness:** The entailment in the form of
  \[ \mathcal{D}_{es} \models (\forall s)(GoalSit(s) \supset \neg Statement(s)), \]
  which involves deciding whether all goal situations achieve the negation of the statement;

  **Generic Complement of Necessary Truth:** The entailment in the form of
  \[ \mathcal{D}_{es} \models \neg(\forall s)(GoalSit(s) \supset Statement(s)), \]
  which involves checking the validity of the complement of the Necessary Truth problem;

  **Generic Complement of Necessary Falseness:** The entailment in the form of
  \[ \mathcal{D}_{es} \models \neg(\forall s)(GoalSit(s) \supset \neg Statement(s)), \]
  which involves checking the validity of the complement of the Necessary Falseness problem.

- Existentially quantified problems (i.e., on possibility) include:

  **Generic Possible Truth:** The entailment in the form of
  \[ \mathcal{D}_{es} \models (\exists s)(GoalSit(s) \land Statement(s)), \]
  which is simply the verifications of existential quantification on \( Statement(s) \);
**Generic Possible Falseness:** The entailment in the form of

\[ D_{es} \models (\exists s) (\text{GoalSit}(s) \land \neg \text{Statement}(s)), \]

which involves searching for a possible situation that achieves the negation of the statement;

**Generic Complement of Possible Truth:** The entailment in the form of

\[ D_{es} \models \neg (\exists s) (\text{GoalSit}(s) \land \text{Statement}(s)), \]

which involves checking the validity of the complement of the Possible Truth problem;

**Generic Complement of Possible Falseness:** The entailment in the form of

\[ D_{es} \models \neg (\exists s) (\text{GoalSit}(s) \land \neg \text{Statement}(s)), \]

which involves checking the validity of the complement of the Possible Falseness problem.

Between these two different sets of entailment problems, four equivalence relationships can be established. These equivalences can also be described as *duality between problems* as there exists a goal situation where the statement holds if and only if it is not the case that the statement is false in all goal states.

**Proposition 10** Possible Truth holds iff Complement of Necessary Falseness holds.

**Proof.**

\[ D_{es} \models (\exists s) (\text{GoalSit}(s) \land \text{Statement}(s)) \iff \]
\[ D_{es} \models (\exists s) (\neg (\neg \text{GoalSit}(s) \lor \neg \text{Statement}(s))) \iff \]
\[ D_{es} \models \neg (\exists s) (\neg (\neg \text{GoalSit}(s) \lor \neg \text{Statement}(s))) \iff \]
\[ D_{es} \models \neg (\forall s) (\neg \text{GoalSit}(s) \lor \neg \text{Statement}(s)) \iff \]
\[ D_{es} \models \neg (\forall s) (\neg \text{GoalSit}(s) \lor \neg \text{Statement}(s)). \]

□

**Proposition 11** Possible Falseness holds iff Complement of Necessary Truth holds.

**Proof.**

\[ D_{es} \models (\exists s) (\text{GoalSit}(s) \land \neg \text{Statement}(s)) \iff \]
\[ D_{es} \models (\exists s) (\neg (\neg \text{GoalSit}(s) \lor \neg \text{Statement}(s))) \iff \]
\[ D_{es} \models \neg (\exists s) (\neg (\neg \text{GoalSit}(s) \lor \neg \text{Statement}(s))) \iff \]
\[ D_{es} \models \neg (\forall s) (\neg \text{GoalSit}(s) \lor \neg \text{Statement}(s)) \iff \]
\[ D_{es} \models \neg (\forall s) (\text{GoalSit}(s) \lor \text{Statement}(s)). \]

□
**Proposition 12** Complement of Possible Truth holds iff Necessary Falseness holds.

**Proof.** \( D_{es} \models \neg(\exists s) \left( \text{GoalSit}(s) \land \text{Statement}(s) \right) \) iff
\[ D_{es} \models \neg(\exists s) \left( \neg(\neg \text{GoalSit}(s) \lor \neg \text{Statement}(s)) \right) \] iff
\[ D_{es} \models (\forall s) \left( \text{GoalSit}(s) \supset \neg \text{Statement}(s) \right). \]
\[ \square \]

**Proposition 13** Complement of Possible Falseness holds iff Necessary Truth holds.

**Proof.** \( D_{es} \models \neg(\exists s) \left( \text{GoalSit}(s) \land \neg \text{Statement}(s) \right) \) iff
\[ D_{es} \models \neg(\exists s) \left( \neg(\neg \text{GoalSit}(s) \lor \text{Statement}(s)) \right) \] iff
\[ D_{es} \models (\forall s) \left( \text{GoalSit}(s) \supset \text{Statement}(s) \right). \]
\[ \square \]

Based on \( D_{es} \), our axiomatization of an Event System \( \Theta \) can be completed by introducing several abbreviations to characterize different situations.

Without loss of generality, we require that in \( \Theta \), any event type \( t \in T \) contains one and only one actual event \( e \in E \).\(^3\) Now, a situation of event permutation is one that results from performing a permuted occurrence of each event \( e \in E \) between the initial situation \( S_0 \) and \( s \):
\[
\text{Perm}(s) \overset{\text{def}}{=} \left[ \bigwedge_{\forall t_i \in T} (\exists s_{t_0}) S_0 \subseteq \text{do}(T_i,s_{t_0}) \subseteq s \land \left( \neg(\exists s_{t_1}) (s_{t_0} \neq s_{t_1}) \land (S_0 \subseteq \text{do}(T_i,s_{t_1}) \subseteq s) \right) \right],
\]
that is,

- each event type \( t_i \) has one and only one actual event \( e_i \in E \);
- \( T_i \) is an action function corresponding to \( e_i \);
- the right hand side is a conjunction over all events \( e_i \in E \).

With reference to the set of strict partial orders \( O \) of \( \Theta \), define
\[
\text{POrder}(s) \overset{\text{def}}{=} \left[ \bigwedge_{\forall (e_1 \preceq e_2) \in O} (\exists s_{e_1}, s_{e_2}) S_0 \subseteq \text{do}(E_1,s_{e_1}) \subseteq \text{do}(E_2,s_{e_2}) \subseteq s \right]
\]
where

- \( E_1 \) and \( E_2 \) in the formula correspond to a partial order \( (e_1 \preceq e_2) \in O \);
- the right hand side is a conjunction on partial orders.

\(^3\)Otherwise, we can introduce auxiliary event types to \( \Theta \) to construct a \( \Theta' \), which is identical to \( \Theta \) from perspective of dynamical behaviors of systems.
Chapter 3. Dual Temporal Projection Problems

The admissibility requirement is equivalent to the definition of executability of situations in [59] (see Equation 4.5 on Page 53):

\[ \text{Admi}(s) \overset{\text{def}}{=} (\forall a, s^*).\text{do}(a, s^*) \sqsubseteq s \supset \text{Poss}(a, s^*). \]

Based on these abbreviations, we could match a complete sequence in an Event System to a situation in \( \mathcal{D}_{es} \), which is resulted from performing from \( S_0 \) a permutation of actions that satisfies all constraints of partial orderings:

\[ \text{CS}(s) \overset{\text{def}}{=} \text{Perm}(s) \land \text{POrder}(s), \]

whereas an admissible complete sequence is associated with a situation that is resulted from performing from \( S_0 \) a permutation of actions that satisfies the admissibility requirement and all constraints of partial orderings:

\[ \text{ACS}(s) \overset{\text{def}}{=} \text{Perm}(s) \land \text{POrder}(s) \land \text{Admi}(s). \]

Notice that while “\( \text{CS}(\Theta) \)” is a set of complete sequences with respect to the Event System \( \Theta \), “\( \text{CS}(S) \)” specifies the corresponding situations in the theory \( \mathcal{D}_{es} \). A similar distinction is made with \( \text{ACS}(\Theta) \) and \( \text{ACS}(S) \). The correctness of \( \mathcal{D}_{es} \) on axiomatizing an Event System \( \Theta \) is indicated by the following two theorems.

**Theorem 1** There exists an event sequence \( f = \langle f_1, \ldots, f_k, \ldots, f_m \rangle \) such that

\[ f \in \text{CS}(\Theta) \]

iff there exists a ground situation \( S \) in \( \mathcal{D}_{es} \) such that

\[ \mathcal{D}_{es} \models \text{CS}(S). \]

**Proof.** (\( \Rightarrow \)): From the complete sequence \( f \), we can locate a ground situation \( S \) in \( \mathcal{D}_{es} \), which results from performing a sequence of events \( (f_1 \text{ to } f_m) \) from the initial situation \( S_0 \) in \( \mathcal{D}_{es} \). Now that the sequence \( f \) contains only the actual events from \( \mathcal{E} \) and every actual event in \( \mathcal{E} \), it is the case

\[ \mathcal{D}_{es} \models \text{Perm}(S), \]

from the definition of the abbreviation \( \text{Perm}(S) \). Similarly, since \( f \) satisfies the set of strict partial orders \( \mathcal{O} \), it is also the case that

\[ \mathcal{D}_{es} \models \text{POrder}(S), \]
from the definition of the abbreviation $POrder(s)$. Hence,

$$\mathcal{D}_{es} \models Perm(S) \land POrder(S).$$

($\Leftarrow$): Assume there exists a situation $S$ such that $\mathcal{D}_{es} \models Perm(S) \land POrder(S)$, a sequence of actual events $f$ can be constructed from $S_0$ to $S$ accordingly. Since we have $\mathcal{D}_{es} \models Perm(S)$, it is the case that every event in $\mathcal{E}$ occurs some way along the path from $S_0$ to $S$, and every actual event on the path belongs to $\mathcal{E}$. From the definition of $POrder(s)$, we know that it is the case that $f$ also satisfies $O$. Hence, we have

$$f \in CS(\Theta).$$

\[\square\]

Similarly, we can prove the following theorem.

**Theorem 2** There exists an event sequence $f = \langle f_1, \ldots, f_k, \ldots, f_m \rangle$ such that

$$f \in ACS(\Theta)$$

iff there exists a particular situation $S$ in $\mathcal{D}_{es}$ such that

$$\mathcal{D}_{es} \models ACS(S).$$

**Proof.** The proof is an augmentation to the proof above by taking the issue of event admissibility into consideration. That is, we also consider that fact that all actual events in $f$ satisfy their specific causal rules in $R$ of $\Theta$, and all the actual events that lead to situation $s$ from $S_0$ will satisfy their relevant precondition axioms in $D_{ap}$ of $\mathcal{D}_{es}$ and successor state axioms in $D_{ss}$ of $\mathcal{D}_{es}$.

From the definitions on the possible/necessary truth of condition $p$, i.e., $p \in Poss^+_A(e, \Theta)$ and $p \in Nec^+_A(e, \Theta)$, we can define a goal situation as an admissible complete sequence, and the statement is about the *holdness* of the fluent $F_p$. The dual problems in this example can be specified as the following Event Systems theory entailment problems.4

**Specialized Possible Truth:** The entailment in the form of

$$\mathcal{D}_{es} \models (\exists s) (CS(s) \land Admi(s) \land F_p(s)),$$

4This example reappears in the next section as the Nebel and Bäckström’s duals.
Specialized Necessary Truth: The entailment in the form of

\[ \mathcal{D}_{es} \models (\forall s) (CS(s) \supset Admi(s) \land F_p(s)). \]

The following two theorems demonstrate the correctness of our axiomatization of Event Systems and the temporal projection problems in particular.

**Theorem 3** Given an Event System Θ and the theory for it \( \mathcal{D}_{es} \),

\[ p \in Poss^+_A(\Theta) \]

iff

\[ \mathcal{D}_{es} \models (\exists s) (CS(s) \land Admi(s) \land F_p(s)), \]

where \( p \in \mathcal{P} \) in Θ, and it corresponds to the relational fluent \( F_p \) in \( \mathcal{D}_{es} \).

**Proof.** The proof is a further augmentation to the proof for the Theorem 2. Remember that

\[ p \in Poss^+_A(\Theta) \]

iff there exists a sequence of actual events \( f = \langle f_1, \ldots, f_k, \ldots, f_m \rangle \) such that

\[ f \in ACS(\Theta) : p \in Result(\mathcal{I}, f). \]

Assume we have such a sequence \( f \) such that

\[ f \in ACS(\Theta). \]

Consequently, we have a particular situation \( S \) such that

\[ \mathcal{D}_{es} \models CS(S) \land Admi(S), \]

and \( S \) is resulted from performing the event sequence \( f \) from the initial situation \( S_0 \).

From the definition of Result in Θ and the way we construct \( \mathcal{D}_{S_0} \) from \( \mathcal{I} \), we can obtain

\[ \mathcal{D}_{es} \models F_p(S), \]

by applying the principle of induction on actual events along \( f \), and on situations along the path from \( S_0 \) to \( S \). More precisely, we have

**Base case** Any condition \( p \in \mathcal{I} \) iff \( p \in Result(\mathcal{I}, \langle \rangle) \), iff \( \mathcal{D}_{es} \models F_p(S_0); \)
Chapter 3. Dual Temporal Projection Problems

Induction step Assume \( p \in \text{Result}(\mathcal{I}, \mathcal{f} \setminus f_k) \) iff \( \mathcal{D}_{es} \models F_p(S_{k-1}) \) holds, we know it is also the case \( p \in \text{Result}(\mathcal{I}, \mathcal{f}) \) iff \( \mathcal{D}_{es} \models F_p(S_k) \), from the definition of \( \text{Result} \) (Definition 6) and the definition of \( \mathcal{D}_{es} \) (Definition 45).

\[ \square \]

The theorem regarding the Necessary Truth is stated as follows.

**Theorem 4** Given an Event System \( \Theta \) and the theory for it \( \mathcal{D}_{es} \),

\[ p \in \text{Ness}_A^+(\Theta) \]

iff

\[ \mathcal{D}_{es} \models (\forall s) (\text{CS}(s) \supset \text{Admi}(s) \land F_p(s)), \]

where \( p \in \mathcal{P} \) in \( \Theta \), and it corresponds to the relational fluent \( F_p \) in \( \mathcal{D}_{es} \).

3.4 Four Sets of Temporal Projection Duals

In this section, four sets of dual problems are introduced in order. As reflected in their names, these problems are derived from the work of Nebel and Bäckström ([53]), Chapman ([10]), Dean and Boddy ([12]), and Kambhampati and Nau ([38]).

3.4.1 Nebel and Bäckström’s Duals

Given an Event System \( \Theta \), an event \( e \in \mathcal{E} \), and a condition \( p \in \mathcal{P} \), the Possible and Necessary Truth problems as introduced in Definition 6.1 of [53] are:

\[ p \in \text{Poss}_A^+(\Theta) \text{ iff } \exists f \in \text{ACS}(\Theta) : p \in \text{Result}(\mathcal{I}, f), \]

\[ p \in \text{Nec}_A^+(\Theta) \text{ iff } \forall f \in \text{ACS}(\Theta) : p \in \text{Result}(\mathcal{I}, f). \]

Accordingly, we introduce in \( \mathcal{D}_{es} \) the entailment problems as follows and call them the Nebel and Bäckström’s Duals (which have already appeared in the previous section).

**NB’s Possible Truth:** \( \mathcal{D}_{es} \models (\exists s) (\text{Perm}(s) \land \text{POrder}(s) \land \text{Admi}(s) \land F_p(s)) \)

**NB’s Necessary Truth:** \( \mathcal{D}_{es} \models (\forall s) (\text{Perm}(s) \land \text{POrder}(s) \supset \text{Admi}(s) \land F_p(s)) \)

\[ ^5 \text{The situation } S_{k-1} \text{ is one that is resulted from performing the event sequence } f \setminus f_k \text{ from the initial situation } S_0. \]
It is shown in Corollary 6.5 of [53] that deciding Nebel and Bäckström’s Possible Truth problem is NP-complete (the source of this high complexity is discussed extensively in Section 4.2), which means that, by Proposition 10,

**Theorem 5** The Complement of the Necessary Falseness problem is also NP-complete.

It is observed that, if a given problem can be solved in polynomial time, then its complementary version can be solved in polynomial time too (see page 29 of [24]). Meanwhile, if a given problem is NP-complete, then its complementary version is co-NP-complete (see page 156-157 of [24]). Accordingly, we have that

**Theorem 6** The Complement of Possible Truth is co-NP-complete.

Then, by Proposition 12, we have

**Theorem 7** The Necessary Falseness Problem is also co-NP-complete.

We know that deciding the Necessary Truth problem is polynomial time solvable (from Theorem 6.3 of [53], also presented as Proposition 5 in Chapter 2). Based on this result, we can have the following,

**Theorem 8** Complement of Possible Falseness problem, Possible Falseness Problem, and Complement of Necessary Truth Problem, all are also polynomial-time solvable.

The complexity results can be summarized into the table as follows.

<table>
<thead>
<tr>
<th>Possible Truth</th>
<th>Complexity</th>
<th>Necessary Truth</th>
</tr>
</thead>
<tbody>
<tr>
<td>NB’s Poss. Truth</td>
<td>NP-complete</td>
<td>NB’s Comp. Nece. Falseness</td>
</tr>
<tr>
<td>NB’s Poss. Falseness</td>
<td>Polyt ime</td>
<td>NB’s Comp. Nece. Truth</td>
</tr>
<tr>
<td>NB’s Comp. Poss. Truth</td>
<td>co-NP-complete</td>
<td>NB’s Nece. Falseness</td>
</tr>
<tr>
<td>NB’s Comp. Poss. Falseness</td>
<td>Polyt ime</td>
<td>NB’s Nece. Truth</td>
</tr>
</tbody>
</table>

Table 3.1: Complexity results for Nebel and Bäckström’s Dualities

**3.4.2 Chapman’s Duals**

The paper “Planning for conjunctive goals” [10], took an important role in promoting the progress of AI partial-order planning. The main contribution in [10] is the proposal of Modal Truth Criteria (MTC). Loosely speaking, in the process of planning where the
current plan contains only a partially ordered set of partially instantiated actions, the Necessary Modal Truth Criteria, NMTC, (the Possible Modal Truth Criteria, PMTC) specifies the criterion to evaluate the necessary truth (the possible truth) of a given ground literal in all executions (some execution) of the current plan. The relationship between PMTC and NMTC should be that, as arguably reflected in the statement of MTC in [10], any completion of the plan is executable and produces literal $p$ if and only if it is not the case that some completion of the plan is nonexecutable or does not produce $p$. As pointed out by [53], [10] makes an implicit assumption in that, any completion that satisfies the set of partial orderings is also admissible (quoting [53, p.143]):

\[\ldots\text{Chapman used a similar technique to prove that deciding necessary truth in unconditional plans generated by the TWEAK planning system is a polynomial-time problem for a slightly different formalism. It should be noted, however, that Chapman’s proof of the completeness and correctness of his modal truth criterion relies on the assumption that all events he refers to in his criterion are already (or will become eventually) necessarily admissible.}\]

The assumption made by Chapman in [10] can be stated in $D_{es}$ as follows.

\[D_{es} \models AdmiAsptn \equiv (\forall s) (Perm(s) \wedge POrder(s) \supset Admi(s)),\]

i.e., all completions are admissible. Accordingly, the duality defined in [10] can be stated in Situation Calculus as

**Chapman’s Possible Truth:**

\[D_{es} \cup AdmiAsptn \models (\exists s) (Perm(s) \wedge POrder(s) \wedge F_p(s))\]

**Chapman’s Necessary Truth:**

\[D_{es} \cup AdmiAsptn \models (\forall s) (Perm(s) \wedge POrder(s) \supset F_p(s))\]

The equivalences of entailment problems and the corresponding complexity results can be summarized into the following table. It can be seen from this table that, with the assumption, all of these problems can be solved in polynomial time, with respect to the number of actual events in the system. Since checking the non-emptiness of the set $F$ can be done in polynomial time, we could answer Chapman’s Necessary Truth problem on fluent $F_p$ by querying Chapman’s Possible Truth problem on the negation of $F_p$. 
3.4.3 Dean and Boddy’s Duals

In [12], events not admissible are allowed to occur, but occur without effects. In Definition 6.1 of [53], the corresponding inadmissible Possible Truth and Necessary Truth problems are defined as:

\[
p \in Poss^+(\Theta) \text{ iff } \exists f \in CS(\Theta) : p \in Result(I, f),
\]
\[
p \in Nec^+(\Theta) \text{ iff } \forall f \in CS(\Theta) : p \in Result(I, f).
\]

Alternatively, we could interpret the occurrence of inadmissible events as that there exist two (i.e., indefinite, disjunctive) sets of preconditions for an action to occur, one of them is with empty preconditions and empty effects. Within this context, all complete sequences are actually admissible too. That is, we also have

\[
D_{es} \models (\forall s) (Perm(s) \land POrder(s) \supset Admi(s)).
\]

Note that in Chapman’s duals (as defined in the previous section), the assumption of the admissibility of all goal situations is made explicitly by introducing the abbreviation of AdmiAsptn to \(D_{es}\). Hence, the application of Chapman’s duals is restricted to cases where AdmiAsptn holds. In Dean and Boddy’s duals, however, the admissibility is instead a logical consequence, which is obtained by introducing indefinite preconditions and effects to the causal rules. This difference help explain why (shown in Table 3.2 and Table 3.3) all Chapman’s duals are poly-time solvable whereas all Dean and Boddy’s duals are intractable. Hence, the duality defined in [12] can be stated in \(D_{es}\) as

**DB’s Possible Truth:** \(D_{es} \models (\exists s) (Perm(s) \land POrder(s) \land F_p(s))\)

**DB’s Necessary Truth:** \(D_{es} \models (\forall s) (Perm(s) \land POrder(s) \supset F_p(s))\)

It is shown in Theorem 2.1 of [12] that the Possible Truth Problem is NP-complete. Similarly, we prove that
Theorem 9 \( DB’s \) Possible Falseness Problem

\[
\mathcal{D}_{es} \models (\exists s) \ (\text{Perm}(s) \land \text{POrder}(s) \land \neg F_p(s))
\]

is NP-complete.

**Proof.** It is easy to see that the problem is in NP. The NP-hard proof constructs a transformation similar to the one in the proof for Theorem 2.1 of [12]. From an arbitrary instance of 3SAT \( S = \langle U, C \rangle \), where \( U = \{u_1, u_2, \ldots, u_n\} \) is a set of variables and \( C = \{c_1, c_2, \ldots, c_m\} \) such that \(|c_j| = 3\) for \( 1 \leq j \leq m \) is the set of clauses, we construct an Event System \( \Theta = \langle P, T, R, E, O, I \rangle \) where

- \( P = \{u_1^+, \ldots, u_n^+, u_1^-, \ldots, u_n^-, c_1, \ldots, c_m, \text{unsat}\} \). That is, for each \( i \) such that \( 1 \leq i \leq n \), \( u_i^+ \) refers to the fact that \( u_i \in U \) is assigned to true in \( U \), \( u_i^- \) refers to the fact that \( u_i \in U \) is assigned to false in \( U \). For each \( j \) such that \( 1 \leq j \leq m \), \( c_j \) refers to the fact that the clause \( c_j \in C \) is satisfied. \( \text{unsat} \) refers to the claim that the 3SAT instance \( S \) is unsatisfied.

- \( T = \{t_1^+, \ldots, t_n^+, t_1^-, \ldots, t_n^-, t_1^c, \ldots, t_m^c, t_{\text{sat}}\} \), that is, for each condition in \( P \), there exists a corresponding event type in \( T \). Further, there exists a unique causal rule with respect to each \( t \in T \) (see below).

- \( \mathcal{R} \) is the union of
  - \( \{\langle t_i^+, \{\}, \{u_i^+\}, \{u_i^-\}\rangle | 1 \leq i \leq n\} \), corresponding to the truth assignment of \( u_i \) in \( S \),
  - \( \{\langle t_i^-, \{\}, \{u_i^-\}, \{u_i^+\}\rangle | 1 \leq i \leq n\} \), corresponding to the truth assignment to the negation of \( u_i \) in \( S \),
  - \( \{\langle t_i^c, \{l_{i1}^1\}, \{c_i\}, \{\}\rangle | 1 \leq i \leq m, 1 \leq j \leq 3\} \) whereas accordingly in \( S \) we have \( c_i = [l_{i1}^1, l_{i2}^2, l_{i3}^3] \) for \( 1 \leq i \leq m \), hence, condition \( c_i \) in \( P \) of \( \Theta \) holds iff the clause \( c_i \) is satisfied in \( S \),
  - \( \{\langle t_{\text{sat}}, \{c_1, \ldots, c_m\}, \{\text{unsat}\}, \{\}\rangle \} \), that is if every \( c_i \) holds, then \( \text{unsat} \) should be falsified.

- \( \mathcal{E} = \{e_1^+, \ldots, e_n^+, e_1^-, \ldots, e_n^-, e_1^c, \ldots, e_m^c, e_{\text{sat}}\} \), that is, each event type corresponds to one any only one actual event.

- \( O \) is the union of
\(- e_i^+ \prec e_j^c \text{ for } 1 \leq i \leq n \text{ and } 1 \leq j \leq m,\)
\(- e_i^- \prec e_j^c \text{ for } 1 \leq i \leq n \text{ and } 1 \leq j \leq m,\)
\(- e_i^c \prec e_{\text{sat}} \text{ for } 1 \leq i \leq m.\)

That is, all literal events (i.e., those events superscribed with “+” or “−” signs) should precede the clause events (i.e., those events superscribed with letter “c”s), whereas the event \(e_{\text{sat}}\), the one reflecting if the instance \(S\) is satisfied or not, is preceded by all other events.

- \(I = \{\text{unsat}\}.\)

It is the case that \(S\) is satisfiable iff in \(\Theta\) there exists a completion of events that will falsify \(\text{unsat}\) iff \(D_{es} \models (\exists s) (\text{Perm}(s) \land \text{POrder}(s) \land \neg F_{\text{unsat}}(s))\), where the condition \(\text{unsat}\) corresponds to the fluent \(F_{\text{unsat}}\) in \(D_{es}\).

(\(\Rightarrow\)): Assume there is an assignment to the variables \(u \in U\) with which all clauses \(c \in C\) are satisfied. A permuted sequence of all literal events in \(E\) can be constructed accordingly such that the corresponding literal conditions will hold after the sequence. But these conditions will guarantee that any subsequent (note the order constraints in \(O\)) permuted occurrence of the clause events will achieve all clause conditions. And finally, the occurrence of \(e_{\text{sat}}\) will be admissible and will falsify the condition \(\text{unsat}\).

(\(\Leftarrow\)): Assume there exists a complete sequence such that \(\text{unsat}\) will be falsified. We know that, by \(O\), \(e_{\text{sat}}\) is the only event that could possibly falsify \(\text{unsat}\), and it will occur admissibly as the last event in the sequence. But this means that all clause conditions should hold before the occurrence of \(e_{\text{sat}}\). And these clause conditions could only possibly be achieved by the holdness of literal conditions after the permuted occurrence of literal conditions (note \(O\) again). But these conditions correspond to a satisfiable assignment of the variables in \(S\).

The complexity results in Dean and Boddy’s formalism are summarized in the table as follows. Note that the co-NP-completeness of Necessary Truth problem is proved in [12] as Theorem 2.2, whereas the result can also be obtained by applying Proposition 13 and the fact that the Complement Possible Falseness problem is co-NP-complete.
Chapter 3. Dual Temporal Projection Problems

### 3.4.4 Kambhampati and Nau’s Duals (Partial Truth Duals)

In this section, we first define the following Necessary Falseness problem

\[
(\forall s) (Perm(s) \land POrder(s) \supset \neg(Admi(s) \land \neg F_p(s)))
\]

and note that it can be rewritten as

\[
(\forall s) (Perm(s) \land POrder(s) \supset (Admi(s) \supset F_p(s))).
\]

In fact, the reason this set of problems is under the name of Kambhampati and Nau is because the Necessary Falseness problem, as demonstrated above, corresponds to the definition of partial truth problems in [38]. Detailed discussion will be covered in the subsequent section.

**KN’s Possible Truth:** \( \mathcal{D}_{es} \models (\exists s) (Perm(s) \land POrder(s) \land Admi(s) \land \neg F_p(s)) \)

**KN’s Necessary Truth:** \( \mathcal{D}_{es} \models (\forall s) (Perm(s) \land POrder(s) \supset Admi(s) \land \neg F_p(s)) \)

The complexity results are the same as the ones in Nebel and Bäckström’s dualities and are summarized in the following table.

<table>
<thead>
<tr>
<th>Possible Truth</th>
<th>Complexity</th>
<th>Necessary Truth</th>
</tr>
</thead>
<tbody>
<tr>
<td>KN’s Poss. Truth</td>
<td>NP-complete</td>
<td>KN’s Comp. Nece. Falseness</td>
</tr>
<tr>
<td>KN’s Poss. Falseness</td>
<td>Polytime</td>
<td>KN’s Comp. Nece. Truth</td>
</tr>
<tr>
<td>KN’s Comp. Poss. Truth</td>
<td>co-NP-complete</td>
<td>KN’s Nece. Falseness</td>
</tr>
<tr>
<td>KN’s Comp. Poss. Falseness</td>
<td>Polytime</td>
<td>KN’s Nece. Truth</td>
</tr>
</tbody>
</table>

Table 3.4: Complexity results for Partial Dualities
3.5 On Alternative Negations

In Kambhampati and Nau’s paper with the title: On the nature and role of modal truth criteria in planning [38], a modal-logic-like formalism is applied to investigate the relationship between the Possible Truth problems and the Necessary Truth problems. In particular, the non-duality in the following form of inequivalence between these two classes of problems is formulated (as Theorem 1 in [38])

\[ \Box \mathcal{M}(p, fin) \not\equiv \neg \Diamond \neg \mathcal{M}(p, fin). \]

As will be shown in this section, in the language of Situation Calculus, with an alternative, probably more natural, way of interpreting the concepts such as negation and duality, equivalence could be demonstrated instead.

Note that in Section 2.1.4, we have reviewed the definitions introduced in [38], whereas the Possible Truth problems and the Necessary Truth problems in their formalisms are also introduced.

3.5.1 Complete Versus Partial Negation

Theorem 10 If we define the negation of truth “¬\( \mathcal{M}(p, fin) \)” as

\[ \neg \mathcal{M}(p, fin) \equiv \neg [C(p, fin) \land \bigwedge_{\forall a \in P; \forall p_a \in pre(a)} C(p_a, in(a))], \]

then

\[ \Box \mathcal{M}(p, fin) \equiv \neg \Diamond \neg \mathcal{M}(p, fin). \]

Proof. The proof is interleaved with their [38] formalisms and the terms in \( \mathcal{D}_{es} \). Again it should be noted that a condition \( p \) corresponds to a fluent \( F_p \) in \( \mathcal{D}_{es} \).

In \( \mathcal{D}_{es} \), we know that “\( \Box \mathcal{M}(p, fin) \)” corresponds to Nebel and Bäckström’s Necessary Truth problem:

\[ \mathcal{D}_{es} \models (\forall s) (Perm(s) \land POrder(s) \supset (Admi(s) \land F_p(s))), \]

whereas “\( \Diamond \mathcal{M}(p, fin) \)” corresponds to Nebel and Bäckström’s Possible Truth problem:

\[ \mathcal{D}_{es} \models (\exists s) (Perm(s) \land POrder(s) \land \neg (Admi(s) \land F_p(s))). \]

As summarized in Table 3.4, both problems can be solved in polynomial time.
Since determining whether \( p \) is necessarily true is equivalent to checking whether \( p \) is necessarily conditionally true, and whether the preconditions of each step of the plan are necessarily conditionally true, i.e., the plan is always executable, we have\(^6\)

\[
\Box M(p, fin) \equiv \Box (C(p, fin)) \land \Box (\bigwedge a \ C(p_a, in(a))).
\]

Now, since \( p \) is conditionally true in every completion iff it is not possible to have one completion such that \( p \) is conditionally false, we have,

\[
\Box (C(p, fin)) \equiv (\neg \Diamond \neg)(C(p, fin)),
\]

meanwhile, on the executability of the plan, clearly every completion is executable iff no nonexecutable completion is possible,

\[
\Box (\bigwedge a \ C(p_a, in(a))) \equiv (\neg \Diamond \neg)(\bigwedge a \ C(p_a, in(a))).
\]

Likewise, in \( D_{es} \), we have:

\[
D_{es} \models (\forall s)\left(Perm(s) \land POrder(s) \supset F_p(s)\right) \equiv (\neg \exists s)\left(Perm(s) \land POrder(s) \land \neg F_p(s)\right)
\]

and

\[
D_{es} \models (\forall s)\left(Perm(s) \land POrder(s) \supset Admi(s)\right) \equiv (\neg \exists s)\left(Perm(s) \land POrder(s) \land \neg Admi(s)\right).
\]

Hence it is the case that

\[
\Box M(p, fin) \equiv (\neg \Diamond \neg)(C(p, fin)) \land (\neg \Diamond \neg)(\bigwedge a \ C(p_a, in(a))),
\]

that is,

\[
\Box M(p, fin) \equiv (\neg \Diamond)(\neg C(p, fin)) \land (\neg \Diamond)(\neg \bigwedge a \ C(p_a, in(a))).
\]

That is, in \( D_{es} \) we have

\[
D_{es} \models (\forall s) (Perm(s) \land POrder(s) \supset (Admi(s) \land F_p(s))) \equiv \\
(\neg \exists s_1)\left(Perm(s_1) \land POrder(s_1) \land \neg Admi(s_1)\right) \land (\neg \exists s_2)\left(Perm(s_2) \land POrder(s_2) \land \neg F_p(s_2)\right).
\]

Hence, it is not possible to have a completion where \( p \) is conditionally false in its final state, or a completion which is nonexecutable,

\[
\Box M(p, fin) \equiv (\Diamond (\neg C(p, fin)) \lor (\Diamond (\neg \bigwedge a \ C(p_a, in(a))))),
\]

\(^6\)Notice that Modal Necessity distributes over conjunctions.
That is, in $\mathcal{D}_{es}$ we have

$$\mathcal{D}_{es} \models (\forall s) (\text{Perm}(s) \land P\text{Order}(s) \supset (\text{Admi}(s) \land F_p(s))) \equiv$$

$$\neg(\exists s) \left[ (\text{Perm}(s) \land P\text{Order}(s) \land \neg\text{Admi}(s)) \lor \left( \text{Perm}(s) \land P\text{Order}(s) \land \neg F_p(s) \right) \right].$$

Further, we have that it is not possible to have a completion which is nonexecutable or where $p$ is false in its final state,\footnote{The concept of Modal Possibility distributes over disjunctions is actually applied here.}

$$\Box \mathcal{M}(p, fin) \equiv \neg \lozenge \left( (\neg C(p, fin)) \lor \left( \neg \left[ \bigwedge C(p_a, in(a)) \right] \right) \right),$$

That is, in $\mathcal{D}_{es}$ we have

$$\mathcal{D}_{es} \models (\forall s) (\text{Perm}(s) \land P\text{Order}(s) \supset (\text{Admi}(s) \land F_p(s))) \equiv$$

$$\neg(\exists s) (\text{Perm}(s) \land P\text{Order}(s) \land (\neg\text{Admi}(s) \lor \neg F_p(s))).$$

Finally,

$$\Box \mathcal{M}(p, fin) \equiv \neg \lozenge \left( (\neg C(p, fin)) \land \left[ \bigwedge C(p_a, in(a)) \right] \right) \equiv \neg \lozenge \neg \mathcal{M}(p, fin).$$

That is, in $\mathcal{D}_{es}$ we have

$$\mathcal{D}_{es} \models (\forall s) (\text{Perm}(s) \land P\text{Order}(s) \supset (\text{Admi}(s) \land F_p(s))) \equiv$$

$$\neg(\exists s) (\text{Perm}(s) \land P\text{Order}(s) \land (\neg\text{Admi}(s) \land \neg F_p(s))),$$

which exactly corresponds to the Nebel and Bäckström’s duality between their Necessary Truth problem and their Complement of Possible Falseness problem (Table 3.4). \hfill \square

It is obvious that in [38], “$\neg \mathcal{M}(p, fin)$” is not defined to be

$$\neg (C(p, fin) \land \left[ \bigwedge C(p_a, in(a)) \right]),$$

otherwise, it can be concluded that

$$\Box \mathcal{M}(p, fin) \equiv \neg \lozenge \neg \mathcal{M}(p, fin),$$

which against Theorem 1 of [38]. Although never explicitly stated, it seems that instead they have the following definition

$$\neg \mathcal{M}(p, fin) \equiv \neg (C(p, fin)) \land \left[ \bigwedge C(p_a, in(a)) \right].$$
With the partial negation on \( \mathcal{M}(p, fin) \), it is stated in [38] that the possible truth is indeed the dual of the partial truth. Although a description for the partial truth is given on Page 138 of [38], a modal-theoretic definition on partial truth is not given either. Here we demonstrate that, using the formalism of Situation Calculus, and with the assumption of partial negation, the duality between the Possible Truth problems and the Partial Truth problems can be demonstrated in the following straightforward way.

**Example 1** In \( \mathcal{D}_{es} \), the entailment problem

\[
\mathcal{D}_{es} \models \neg (\forall s) \left( Perm(s) \land POrder(s) \supset (Admi(s) \supset \neg F_p(s)) \right)
\]

can be rewritten as

\[
\mathcal{D}_{es} \models \neg (\forall s) \left( (Perm(s) \land POrder(s) \land Admi(s)) \supset \neg F_p(s) \right),
\]

which equals to

\[
\mathcal{D}_{es} \models (\exists s) \left( \neg ((Perm(s) \land POrder(s) \land Admi(s)) \supset \neg F_p(s)) \right),
\]

which holds iff

\[
\mathcal{D}_{es} \models (\exists s) \left( Perm(s) \land POrder(s) \land Admi(s) \land F_p(s) \right).
\]

### 3.6 Summary

In this chapter, Event Systems, a framework for dynamical system representation that has wide use in AI planning in particular, are described as an action theory called \( \mathcal{D}_{es} \) in the language of Situation Calculus. The sequences of events from the set of actual events \( \mathcal{E} \) on event types \( \mathcal{T} \) in an Event System \( \Theta \) is captured in \( \mathcal{D}_{es} \) by introducing a particular abbreviation \( Perm(s) \) to define a particular situation \( s \), such that all these actual events will occur along the path from the initial situation \( S_0 \) to \( s \). In addition, the set of partial orderings \( \mathcal{O} \) in \( \Theta \) is also defined in \( \mathcal{D}_{es} \) by introducing another abbreviation \( POrder(s) \) for a sentence in the language of Situation Calculus. However, the abbreviation \( Admi(s) \) is in fact identical to the existing abbreviation of \( executable(s) \) in Situation Calculus.

With this formalism, we are able to specify four different varieties of temporal projection problems, on both the Possible Truth problems and the Necessary Truth problems, as action theory entailment problems. In addition, we prove that our axiomatization on
Event Systems is correct for these problems by offering rigorous argument on the one-to-one and onto relationship between a solution in the original problem and an affirmative result in its corresponding entailment problem in $\mathcal{D}_{es}$.

Additionally, we demonstrate four sets of equivalences between these problems. Perhaps more importantly, using these equivalences, we provide a unified view on existing results in the literature, authored by [10], [12], [53], and [38]. As new reasoning problems arise, their computational complexities are provided accordingly. It is also shown that, depending on the properties of the theories, a given particular entailment problem might have different computational complexity in these theories.

We in particular revisit several results presented in [38], but within the constructs of $\mathcal{D}_{es}$. We make a formal discussion on the definition of negation on expressions and thus clarify a central concept in the results of [38] that is quite confusing.

In summary, we provide in this Chapter an ontological account for Event Systems and their behaviors. The action theory $\mathcal{D}_{es}$ for Event Systems is built upon mathematical logic, offering a unifying formalism that is advantageous from both presentational and computational perspectives.

Within $\mathcal{D}_{es}$, several variants of temporal projection problems are described in a non-procedural manner (i.e., as sentences), making it clear which assumptions actually make one variant different from the other. Convincing examples drawn from the literatures are also presented to show that achieving this level of clarity is important, and not always practical for all other formalisms (for semi-formal formalisms such as STRIPS-alike presentation in [10] and modal-logic-like construction in [38]).

In addition, these ontological descriptions of the temporal projection problems in $\mathcal{D}_{es}$ also provide causal connections between logical sentences and system behaviors. As such, relations between problems, such as the *duality and equivalence between two problems*, can be investigated through studying logical equivalences between appropriate sentences, thus potentially by the direct mechanism of logical deductions. As an interesting side effect, algorithms or complexity results for one particular problem could easily be propagated to other problems through basic operations such as *negation, or complementation*.

It is worth noting that temporal projection problems, in particular, Dean and Boddy’s Necessary Truth problems, are supposed to be the basic underlying problems in checking the validity of a given nonlinear plan. While in general, propositional planning is PSPACE-complete ([16], [15]), there exist restricted planning problems which are tractable [9], whereas their corresponding Dean and Boddy’s Necessary Truth problems
are still intractable. In that regard, as studied in [53], the additional requirement of the so-called coherence (in the language of \(D_{ea}\), admissibility) in nonlinear plans contributes to this complexity difference. As such, Nebel and Bäckström’s Necessary Truth problems are proposed accordingly in [53], and it is shown that these problems are indeed tractable.

Perhaps more importantly, we believe that the ontological approach presented in this chapter potentially has a much wider applicability, in particular, to dynamical domains where problems with dual concepts are involved. For example, it might be useful to specify a theory of social choice systems [64]. Within the theory, the relationship between the problems of deciding possible winners and those of deciding necessary winners for different voting methods, and the complexity to compute these problems [84], can be illustrated in a way similar to what we have demonstrated here.
Chapter 4

Possible Truth Problems

4.1 Introduction

The previous chapter provided an investigation on the relationship between the Possible Truth problems and the Necessary Truth problems, whereas the current chapter will now focus on NP-complete Possible Truth problems. The analysis of computational properties of temporal projection problems was first studied by Dean and Boddy [12], where events are partially ordered, and the framework for cause-and-effect relationships is based on the propositional STRIPS representation.

In Dean and Boddy’s formalism, an action whose preconditions are not satisfied will still occur but without effects. Temporal projection problems with this setting in this thesis are called inadmissible temporal projection problems. Whereas, if we require that all occurrences must be admissible, i.e., must satisfy all their preconditions, we are dealing with admissible temporal projection problems. The intractability of (inadmissible) temporal projection is stated in [12]: determining the inadmissible Possible Truth problem remains NP-complete even when several severe restrictions are presented; whereas only unrealistically trivial ones are polytime solvable.

A technical follow-up written by Nebel and Bäckström [53] points out that one special case, the inadmissible Possible Truth problem in Simple Event Systems (SES), conjectured to be polynomial time solvable in [12], is indeed NP-complete. In addition, they also considered the admissible Possible Truth problems and show that such problems in general Event Systems are also NP-complete.

This chapter continues the line of work as stated above and can be divided into two parts. First, we evaluate the computational complexities of a modified Possible
Truth problem in various special cases of Event Systems. Second, we apply various constraints to the most restricted NP-complete inadmissible Possible Truth problem, in order to potentially obtain tractability. These two parts are summarized in Section 4.2 and Section 4.3, respectively.

In Section 4.2, we show that the admissible Possible Truth problem maintains its NP-completeness in an Almost-simple Event System, a Simple Event System, and even an Almost-simple Event System whose cause-and-effect relationship graph is a directed acyclic graph (DAG). However, when in a Simple Event System with DAG, the problem is tractable, whereas the corresponding inadmissible Possible Truth problem remains NP-complete in this case. This fact indicates that, in certain contexts, the role of the admissibility is critical in bringing the problem into tractable zones. In addition, new insights on the source of the complexity in the Possible Truth problems are obtained from analyzing the results. That is, in contrast to what has been claimed in [53], aside from the partial orderings, many other factors, such as the size of the preconditions lists and effects lists, the size of the initial set, the topological structure of the cause-and-effect graph, all contribute to the intractability of the problem.

In Section 4.3, two extra constraints are applied independently to the inadmissible Possible Truth problem in SESs, which insofar as we know is the most restricted intractable inadmissible Possible Truth problem. The first constraint is on the graph-theoretic representation of the cause-and-effect relationships between events. More precisely, in Section 4.3.1, we first show that the problem of the Path Avoiding Forbidden Pairs of Edges (PAFP-E) in a special class of graphs, 2-layered planar s-t-DAGs, is NP-complete. Using this result, we move on to show that the inadmissible Possible Truth problem in an SES, where the cause-and-effect graph is a 2-layered planar s-t-DAG, is also NP-complete, by a polynomial transformation constructed in exactly the same way as the one presented in [53].

The second constraint is on the partial orders of events and the results concerning this constraint are presented in Section 4.3.2, which contains three parts. First, motivated by a recent result [39], in which a polytime algorithm is proposed to solve the Path Avoiding Forbidden Pairs of Vertices (PAFP-V) problem where the forbidden pairs are with the so-called (Vertices) Hierarchical Structure (VHS), we revise the algorithm to show that PAFP-E with a similarly defined (Edges) Hierarchical Structure (EHS) can also be solved in polytime. Second, we show that, if for any partial order $a_1 \prec a_2$ there exists a path in the cause-and-effect graph $G$ from $a_2$ to $a_1$, a Possible Truth problem in SES can be
transformed into a PAFP-E problem and the existence of solution in both directions is preserved by this transformation. Third, if the transformed PAFP-E is with EHS, it can be solved in polytime, which clearly means that the original special class of inadmissible Possible Truth problem is also polytime solvable.

4.2 Admissible Possible Truth

Remember that, depending whether the admissibility is imposed, two variants of Possible Truth Problems are defined as follows.

**Definition 46** Given an Event System $\Theta$ and a condition $p \in \mathcal{P}$:

- $p \in \text{Poss}^+(\Theta)$ iff $\exists f \in \text{CS}(\Theta) : p \in \text{Result}(I, f)$,
- $p \in \text{Poss}_A^+(\Theta)$ iff $\exists f \in \text{ACS}(\Theta) : p \in \text{Result}(I, f)$.

These two problems can also be specified as the following Event Systems theory entailment problems,

**Admissible Possible Truth**: $D_{es} \models (\exists s) (\text{CS}(s) \land \text{Admi}(s) \land F(s))$

**Inadmissible Possible Truth**: $D_{es} \models (\exists s) (\text{CS}(s) \land F(s))$

It is well known that deciding $p \in \text{Poss}^+(e, \Theta)$, where $\Theta$ is a general Event System or a Simple Event System, is NP-complete. (See Theorem 2.1 of [12], and Theorem 3.3 of [53], respectively.) Meanwhile, deciding $p \in \text{Poss}^A_\Lambda(e, \Theta)$, where $\Theta$ is a general Event System, is NP-complete. (See Theorem 6.4 and Corollary 6.5 of [53].) These results are summarized in Table 4.1. The two items highlighted in **boldface** are new results introduced in Section 4.2, where the tractability line between them is analyzed in more detail. In Section 4.3, we start with the NP-complete problem at Column 1, Row 3 (in **italics**), and consider adding two extra constraints to the problem.

Finally, note that in general temporal projection problems involve evaluating the possible truth and the necessary truth of conditions. Our primary concern in this chapter is with the Possible Truth problems, whereas in the previous chapter, we discussed the relationship between these two classes of problems.

In the two sections below, five different cases of admissible possible truth problems are considered. Three of them are NP-complete (from Theorem 11 to Theorem 13), whereas two polynomial-time solvable (Theorem 14 and Theorem 15). Their relationship
Table 4.1: Existing and new complexity results on Possible Truth problems

<table>
<thead>
<tr>
<th>Event Systems</th>
<th>Inadmissible</th>
<th>Admissible</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>NP-complete (Theorem 2.1 of [12])</td>
<td>NP-complete (Corollary 6.5 of [53])</td>
</tr>
<tr>
<td>Simple</td>
<td>NP-complete (Theorem 3.3 of [53])</td>
<td>NP-complete</td>
</tr>
<tr>
<td>Simple, DAG</td>
<td>NP-complete (Theorem 3.3 of [53])</td>
<td>Polynomial time</td>
</tr>
</tbody>
</table>

is depicted in Figure 4.1, where an arrow in the Figure connects a problem to its extension (in the sense of additional restrictions). Note that when partial order is not considered, the Admissible Possible Truth problem is already poly-time solvable (Theorem 15), hence further restricted versions to the problem are also poly-time solvable, and these problems correspond to the neglected boxes below the one at the right-hand-side of Figure 4.1.

![Figure 4.1: NP-complete admissible Possible Truth Problems](image)

4.2.1 Intractable Computational Results

Proof Ideas: Before we present the NP-complete theorems and the proofs, we first informally introduce the proof ideas behind these proofs. In particular, for the hardness part of each of the three NP-complete results (Theorem 11, Theorem 12, and Theorem 13), we provide three similar polytime transformations from the NP-complete problem: directed Hamiltonian path (DHP). Each transformation converts a directed graph $G =$
$(V,E,v_x,v_y)^1$ into an Event System $\Theta(G)$, which is at least almost-simple, so that its cause-and-effect relationship can be represented as a directed graph. The conversion is done without knowing whether there exists a directed Hamiltonian path in $G$, but it turns out that there exists a directed Hamiltonian path iff a particular condition in the converted Event System is possibly true after an admissible completion of the actual events in Event System.

By splitting a node in $G$ into two conditions in $\Theta(G)$, an actual event is introduced accordingly (i.e., those events correspond to the nodes in $G$). Obviously there exists one actual event that corresponds to $v_x$, and one actual event corresponds to $v_y$. Depending on how Event System is constructed, the existence of a Hamiltonian path in $G$ should somehow correspond to the existence of an admissible subsequence of events, starting with the node event $v_x$ and finishing with the node event $v_y$. In fact, the design of the set of partial orders in the transformed $\Theta(G)$ requires that all other node events must occur between node event $v_x$ and node event $v_y$. Hence, if both events, $v_x$ and $v_y$, are admissible, all node events in between are also admissible. This subsequence then corresponds to a Hamiltonian path from $v_x$ to $v_y$ in $G$. Conversely, if there exists such a path in $G$, then, there must exist an admissible subsequence from node event $v_x$ to node event $v_y$.

The remaining issues are related to the edge events in $\Theta(G)$ (i.e., those events correspond to the edges in $G$). Every two neighboring node events in an admissible subsequence from $v_x$ and $v_y$ certainly are connected by an edge event. However, some of the edge events are not in the subsequence. But their admissible occurrences are handled by further introducing the auxiliary in-node events and out-nodes events.

**Theorem 11** Deciding $p \in \text{Poss}_A^+(\Theta)$ (i.e., determining the entailment problem $\mathcal{D}_{es} \models (\exists s) (\text{ACS}(s) \land F(s)))$, where $\Theta$ is an Almost-simple Event System, is NP-complete.

**Proof.** The problem is in NP. Given $\Theta$, an event sequence $f$, the membership of $f \in \text{ACS}(\Theta)$ and $p \in \text{Result}(\mathcal{I}, f)$ can be verified in polytime.

Let $G = (V,E,v_x,v_y)$ be a digraph, where $V = \{v_1,\ldots,v_n,v_x,v_y\}$ and $|V| = n + 2$, $E = \{e_1,\ldots,e_m\}$ and $|E| = m$, $\text{in}(v_i)$ and $\text{out}(v_i)$ are the number of edges entering and leaving the node $v_i$, respectively, and $v_x,v_y \in V$ are the start point and the end point for a DHP, respectively, we define as follows an almost-simple (as $|\mathcal{I}| > 1$ in general) Event System $\Theta = (\mathcal{P}, \mathcal{T}, \mathcal{R}, \mathcal{E}, \mathcal{O}, \mathcal{I})$ such that

---

$^1$V is the set of vertices, E is the set of edges, and $v_x$ is the starting point, and $v_y$ is the ending point.
\( \mathcal{P} = \mathcal{P}_{\text{nodes}} \cup \mathcal{P}_{\text{in\_nodes}} \cup \mathcal{P}_{\text{x\_in}} \cup \mathcal{P}_{\text{y\_in}} \) where

- \( \mathcal{P}_{\text{nodes}} = \{ v^1_{i\ left}; v^1_{i\ right} | 1 \leq i \leq n \text{ or } i = x \text{ or } i = y \} \) (generally, each node \( v_i \) (including \( v_x \) and \( v_y \)) from the set of vertices \( V \) of \( G \) is split to create two conditions, namely, \( v^1_{i\ left} \) and \( v^1_{i\ right} \), in \( \mathcal{P} \) of \( \Theta \). The superscript 1 in \( v^1_{i\ left} \) or \( v^1_{i\ right} \) indicates that the node is obtained from splitting original vertices in the given graph, so that the nodes obtained by other means will have different superscripts. They are called, respectively, as the \( i\text{-left-node condition} \) and the \( i\text{-right-node condition} \), where \( 1 \leq i \leq n \), or \( i = x \), or \( i = y \).

- \( \mathcal{P}_{\text{in\_nodes}} = \{ v^j_{i\ right} | \text{out}(v_i) = k, 1 \leq j < k, 1 \leq i \leq n \} \). Note that the number of auxiliary in-nodes \( v^j_{i\ right} \) equals to the out-degree of the node \( v_i \) minus 1. The purpose of these auxiliary nodes is to make sure that all events (represented with edges of a graph) will actually occur to contribute to a complete sequence of events.

- \( \mathcal{P}_{\text{x\_in}} = \{ v^j_{x\ right} | \text{out}(v_x) = k, 1 \leq j < k, 1 \leq i \leq n \} \) (for any node other than \( v_y \), if the out-degree of \( v_i \) is \( k \) such that \( k \geq 2 \), then \((k - 1)\) extra conditions, i.e., from \( v^2_{i\ right} \) to \( v^k_{i\ right} \), are also created in \( \mathcal{P} \), and they are called as \( i\text{-in-node conditions} \), where \( 1 \leq i \leq n \), or \( i = x \)). Although, the starting node \( x \) (with out-degree greater or equals to 1) might be included in the previous set \( \mathcal{P}_{\text{in\_nodes}} \), we treat this node as special, because it will help later to clarify some matters.

- \( \mathcal{P}_{\text{y\_in}} = \{ v^j_{y\ right} | \text{out}(v_y) = k, 1 \leq j \leq k, 1 \leq i \leq n \} \) (for the node \( v_y \), however, if the out-degree of \( v_y \) is \( k \) such that \( k \geq 1 \), then \( k \) extra conditions, i.e., from \( v^2_{i\ right} \) to \( v^{k+1}_{i\ right} \), are also created in \( \mathcal{P} \), and they are called as \( y\text{-in-node conditions} \).

Intuitively, by splitting each node in \( V \) of \( G \) to have two conditions in \( \mathcal{P}_{\text{nodes}} \), each node will have one and only one corresponding event type (see \( \mathcal{T}_{\text{nodes}} \) below) and thus will have one and only one corresponding actual event (see \( \mathcal{E}_{\text{nodes}} \) below). But the set of partial orders namely \( \mathcal{O}_1 \) (see below) will enforce that these actual events will only occur between \( e^{11}_{x\ left,x\ right} \), which corresponds to the start node \( v_x \) in \( G \) and \( e^{11}_{y\ left,y\ right} \), corresponding to the end node \( v_y \) in \( G \), setting up a potential correspondence of the existence of Hamiltonian path in \( G \) and the existence of an admissible actual events sequence in \( \Theta \) that achieves the condition \( e^{11}_{x\ left,x\ right} \).
Given that in general a Hamiltonian path in a graph \( G \) will leave some edges unvisited, the set of \( \mathcal{P}_{\text{in\_nodes}} \) is introduced accordingly in \( \Theta \). These conditions in \( \mathcal{P}_{\text{in\_nodes}} \) are set to true initially: they are included in \( \mathcal{I} \) (the only other condition in \( \mathcal{I} \) is \( v^1_{x\_left} \)), such that all of the in-node events, the ones in \( \mathcal{E}_{i\_right} \) (defined below) will occur exactly once. But the set of partial orders \( \mathcal{O}_2 \) (defined below) will ensure that these in-node events can only occur after \( e^1_{x\_left,x\_right} \) is achieved, if possible.

The sets of conditions \( \mathcal{P}_{x\_in} \) and \( \mathcal{P}_{y\_in} \), the set of event types \( \mathcal{T}_y\_right \) and actual events \( \mathcal{E}_y\_right \) are introduced separately, simply to handle the boundary case, without major new ideas involved.

- \( \mathcal{T} = \mathcal{T}_{\text{edges}} \cup \mathcal{T}_{\text{nodes}} \cup \mathcal{T}_{1\_right} \cup \mathcal{T}_{2\_right} \cup \ldots \cup \mathcal{T}_{n\_right} \cup \mathcal{T}_{x\_right} \cup \mathcal{T}_{y\_right} \) where
  - \( \mathcal{T}_{\text{edges}} = \{ t^1_{1\_right,j\_left}(v_i,v_j) \in E \} \) (that is, each edge in \( E \) corresponds to one unique event type in \( \mathcal{T} \)),
  - \( \mathcal{T}_{\text{nodes}} = \{ t^1_{1\_left,i\_right} \mid 1 \leq i \leq n \} \cup \{ t^1_{x\_left,x\_right} \} \cup \{ t^1_{y\_left,y\_right} \} \) (that is, each node in \( E \), which is conceptually split into a left node and a right node to build an edge entering from the left node to the right node, corresponds to one unique event type in \( \mathcal{T} \)),
  - for \( 1 \leq i \leq n \) or \( i = x \), \( \mathcal{T}_{i\_right} = \{ t^1_{i\_right,i\_right} \mid \text{out}(v_i) = k, k \geq 2, 1 < j \leq k \} \) (given an \( i \), for each pair of \( v^j_{i\_right} \) and \( v^1_{i\_right} \), an unique event type is created in \( \mathcal{T} \)),
  - for \( i = y \), \( \mathcal{T}_{y\_right} = \{ t^1_{y\_right,y\_right} \mid \text{out}(v_i) = k, k \geq 1, 1 < j \leq k + 1 \} \) (for each pair of \( v^j_{y\_right} \) and \( v^1_{y\_right} \), an unique event type is created in \( \mathcal{T} \));
- \( \mathcal{R} = \{ \{ q_{i,j}^{x\_left} \}, \{ q_{i,j}^{y\_right} \}, \{ q_{i,j}^1 \} \mid q_{i,j}^{x\_left}, q_{i,j}^{y\_right}, q_{i,j}^1 \in \mathcal{T} \} \);
- \( \mathcal{E} = \{ e^1_{i,j} \} \); That is, each event type \( t \) in \( \mathcal{T} \) has one and only one actual event occurrence \( e \) in \( \mathcal{E} \). Thus, the set \( \mathcal{E} \) can also be defined as a union of \( \mathcal{E}_{\text{edges}} \), \( \mathcal{E}_{\text{nodes}} \), \( \mathcal{E}_{1\_right} \), \( \mathcal{E}_{2\_right} \), \ldots , \( \mathcal{E}_{n\_right} \), \( \mathcal{E}_{x\_right} \), and \( \mathcal{E}_{y\_right} \). An \( e \in \mathcal{E}_{\text{edges}} \) is called as an edge event, an \( e \in \mathcal{E}_{\text{nodes}} \) is called as a node event, whereas an \( e \in \mathcal{E}_{i\_right} \) for \( 1 \leq i \leq n \) or \( i = x \) or \( i = y \), is called as an in-node event.
- \( \mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2 \) where
  - \( \mathcal{O}_1 = \{ e^1_{x\_left,x\_right} \prec \{ e \} \in \mathcal{E} \text{ iff } t \in \mathcal{T}_{\text{nodes}} \prec e^1_{y\_left,y\_right} \} \) (the \( x \) node event precedes all node events, which precede the \( y \) node event),
\( \mathcal{O}_2 = \{ e_{y_{left}, y_{right}}^{1,1} \prec \{ e \in \mathcal{E} \text{ iff } t \in \mathcal{T}_{x_{right}}, 1 \leq i \leq n \text{ or } i = x \text{ or } i = y \} \} \)

(the \( y \) node event precedes all in-node events);

- \( \mathcal{I} = \{ v_{x_{left}}^{1}, \{ v_{i_{right}}^{j} \mid j > 1, 1 \leq i \leq n \text{ or } i = x \text{ or } i = y \} \} \) (the left-node condition for the start point \( v_x \), and all in-node conditions, are initially set to true).

---

**Example Begin**

To illustrate the construction, a simple example is given here. Consider (Figure 4.2) a digraph with DHP \( G = (V, E, v_x, v_y) \), where

\[
V = \{ v_1, v_2, v_x, v_y \},
\]

\[
E = \{ (v_x, v_1), (v_x, v_2), (v_1, v_2), (v_1, v_y), (v_2, v_y) \}.
\]

The transformed Almost-simple Event System \( \Theta \) (depicted in Figure 4.3) is a tuple of \( \langle \mathcal{P}, \mathcal{T}, \mathcal{R}, \mathcal{E}, \mathcal{O}, \mathcal{I} \rangle \) such that

- \( \mathcal{P} = \mathcal{P}_{\text{nodes}} \cup \mathcal{P}_{\text{in nodes}} \cup \mathcal{P}_{\text{x in}} \cup \mathcal{P}_{\text{y in}} \) where
  - \( \mathcal{P}_{\text{nodes}} = \{ v_{x_{left}}^{1}, v_{x_{right}}^{1}, v_{1_{left}}^{1}, v_{1_{right}}^{1}, v_{2_{left}}^{1}, v_{2_{right}}^{1}, v_{y_{left}}^{1}, v_{y_{right}}^{1} \} \),
  - \( \mathcal{P}_{\text{in nodes}} = \{ v_{x_{right}}^{1} \} \),
  - \( \mathcal{P}_{\text{in x}} = \{ v_{x_{right}}^{2} \} \),
  - \( \mathcal{P}_{\text{in y}} = \{ \} \);  

- \( \mathcal{T} = \mathcal{T}_{\text{edges}} \cup \mathcal{T}_{\text{nodes}} \cup \mathcal{T}_{1_{right}} \cup \mathcal{T}_{2_{right}} \cup \mathcal{T}_{x_{right}} \cup \mathcal{T}_{y_{right}} \) where
  - \( \mathcal{T}_{\text{edges}} = \{ t_{x_{right}, 1_{left}}^{1,1}, t_{x_{right}, 2_{left}}^{1,1}, t_{1_{right}, 2_{left}}^{1,1}, t_{1_{right}, y_{left}}^{1,1}, t_{2_{right}, y_{left}}^{1,1} \} \),
  - \( \mathcal{T}_{\text{nodes}} = \{ t_{1_{left}, 1_{right}}^{1,1}, t_{2_{left}, 2_{right}}^{1,1}, t_{1_{left}, 2_{right}}^{1,1}, t_{1_{left}, x_{right}}^{1,1}, t_{1_{left}, y_{right}}^{1,1} \} \),
  - \( \mathcal{T}_{x_{right}} = \{ t_{2,1_{right}, x_{right}}^{1,1} \} \),
  - \( \mathcal{T}_{1_{right}} = \{ t_{2,1_{right}, 1_{right}}^{1,1} \} \),
  - \( \mathcal{T}_{2_{right}} = \{ \} \) and \( \mathcal{T}_{y_{right}} = \{ \} \);

- \( \mathcal{R} = \{ \langle t_{x_{right}, 1_{left}}^{1,1}, v_{x_{right}}^{1} \rangle, \langle v_{1_{left}}^{1}, \{ v_{x_{right}}^{1} \} \rangle, \langle v_{2_{left}}^{1}, \{ v_{x_{right}}^{1} \} \rangle, \ldots, \langle t_{1_{right}, 1_{right}}^{2,1}, v_{1_{right}}^{2} \rangle, \langle v_{2_{right}}^{1}, \{ v_{1_{right}}^{2} \} \rangle, \langle v_{2_{right}}^{2}, \{ v_{1_{right}}^{2} \} \rangle \} \);

- \( \mathcal{E} = \{ e_{x_{right}, 1_{left}}^{1,1}, \ldots, e_{1_{right}, 1_{right}}^{2,1} \} \);
• $\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2$ where
  
  $\mathcal{O}_1 = \{e_{x,\text{left},x,\text{right}}^{1,1} \prec \{e_{1,\text{left},1,\text{right}}^{1,1}, e_{2,\text{left},2,\text{right}}^{1,1}\} \prec e_{y,\text{left},y,\text{right}}^{1,1}\}$,

  $\mathcal{O}_2 = \{e_{y,\text{left},y,\text{right}}^{1,1} \prec \{e_{2,\text{right},x,\text{right}}^{2,1}, e_{1,\text{right},1,\text{right}}^{2,1}\}\};$

• $\mathcal{I} = \{v_{x,\text{left}}^{1}, v_{x,\text{right}}^{2}, v_{y,\text{right}}^{2}\}$.

Note that $|E| = |T_{\text{edges}}| = |E_{\text{edges}}| = 5$, whereas $2|V| = |P_{\text{nodes}}| = 8$. Two in-node events are introduced, corresponding to the fact that, if there ever exists a Hamiltonian path in $G$, the path will visit $v_x$ and $v_1$ and visit once, and exactly one of the two edges leaving $v_x$ and $v_1$ will left unvisited.

There exists a DHP from $v_x$ to $v_y$ in $G$ iff $v_{y,\text{right}}^{1} \in \text{Poss}_{\mathcal{A}}^+(\Theta)$.

$(\Rightarrow)$: If there exists a DHP $p_{x,y}$ from $v_x$ to $v_y$ in $G$, then, an admissible event sequence in $\Theta$ from edge event $e_{x,\text{left},x,\text{right}}^{1,1}$ to edge event $e_{y,\text{left},y,\text{right}}^{1,1}$, say $f_1$, which starts with initial condition $v_{x,\text{left}}^{1}$ and achieves condition $v_{y,\text{right}}^{1}$ in the end, can be constructed accordingly. Since $p_{x,y}$ is a DHP, all node events will occur, and occur exactly once, in $f_1$, making the order constraint $\mathcal{O}_1$ satisfied. After $f_1$, all in-node events, which are required by $\mathcal{O}_2$ to be preceded by the edge event $e_{y,\text{left},y,\text{right}}^{1,1}$, can occur, enabling the remaining edge events that have not occurred in $f_1$ to occur admissibly, and occur exactly once (note that exactly out$(v_i) - 1$ in-node events are introduced to achieve the $i$th right-node condition out$(v_i) - 1$ times). Hence, $f_1$ can be extended to an $f$ such that $f \in \text{ACS}(\Theta)$ and $v_{y,\text{right}}^{1} \in \text{Result}(\mathcal{I}, f)$.

$(\Leftarrow)$: Now, assume there exists an admissible event sequence, i.e., $f \in \text{ACS}(\Theta)$, such that $v_{y,\text{right}}^{1}$ is achieved after $f$, i.e., $v_{y,\text{right}}^{1} \in \text{Poss}_{\mathcal{A}}^+(\Theta)$. Due to the specifications of the initial conditions $\mathcal{I}$ and the partial order constraints $\mathcal{O}$, we know that, one, the edge
event $e^{1,1}_{x\_left,x\_right}$ must be the first event of $f$; two, the subsequence $f/e^{1,1}_{y\_left,y\_right}$, (from $e^{1,1}_{x\_left,x\_right}$ to $e^{1,1}_{y\_left,y\_right}$ inclusive in $f$, written as $f_1$), must include all node events and some edge events, but exclude any in-node events. Since $f_1$ is admissible, it corresponds to a Hamiltonian path from $v_x$ to $v_y$ in $G$.

\textbf{Example Begin}

Going back to the example, a Hamiltonian path in $G$ can be $\{v_x, v_1, v_2, v_y\}$, whereas its corresponding admissible sequence can be

\[
\{e^{1,1}_{x\_left,x\_right}, e^{1,1}_{x\_right,1\_left}, e^{1,1}_{1\_left,1\_right}, e^{1,1}_{1\_right,2\_left}, e^{1,1}_{2\_left,2\_right}, e^{1,1}_{2\_right,y\_left}, e^{1,1}_{y\_left,y\_right}, e^{1,1}_{x\_right,x\_right}, e^{1,1}_{x\_right,2\_left}, e^{1,1}_{1\_right,1\_right}, e^{1,1}_{1\_right,y\_left}\}
\]

\textbf{Example End}

\[\square\]

\textbf{Theorem 12} Deciding $p \in \text{Poss}^+_A(\Theta')$ (i.e., determining the entailment problem $D_{es} \models (\exists s)(\text{ACS}(s) \land F(s)))$, where $\Theta'$ is a Simple Event System, is NP-complete.
Chapter 4. Possible Truth Problems

Proof. The problem is certainly in NP. However we need to modify the hardness part proof for the Theorem above, by transforming from the same $G$ into a $\Theta'$, where $|I'| = 1$. Given $\Theta = \langle P, T, R, E, O, I \rangle$, define $\Theta' = \langle P', T', R', E', O', I' \rangle$, such that

- $P' = P \cup P_{\text{out\_nodes}} \cup P_{\text{out\_x}} \cup P_{\text{out\_y}}$ where
  - $P_{\text{out\_nodes}} = \{v_i^{j+1} | \text{in}(v_i) = k, 1 \leq j < k, 1 \leq i \leq n\}$,
  - $P_{\text{out\_y}} = \{v_y^{j+1} | \text{in}(v_y) = k, 1 \leq j < k\}$ (for any node other than $v_x$, if the in-degree of $v_i$ is $k$ such that $k \geq 2$, then $(k-1)$ extra conditions, i.e., from $v_i^2$ to $v_i^k$, are also created in $P$, and they are called as $i$-out-node conditions, where $1 \leq i \leq n$, or $i = y$),
  - $P_{\text{out\_x}} = \{v_x^{j+1} | \text{out}(v_x) = k, 1 \leq j \leq k\}$ (for the node $v_x$, however, if the in-degree of $v_i$ is $k$ such that $k \geq 1$, then $k$ extra conditions, i.e., from $v_x^2$ to $v_x^k$, are also created in $P$, and they are called as $x$-out-node conditions);

Now that in this case only the left-node condition for the start point $v_x$ is initially set to true (see $I'$ below), i.e., the conditions in $P_{\text{nodes}}$ are set to false initially, those in-node events can not occur after $e_{x, left,x,right}^{1,1}$ is achieved. The solution to the problem is to introduce out-node conditions, i.e., the ones in the set $P_{\text{out\_nodes}}$. Correspondingly, the out-node events are introduced (included in the set $E_{x, left}$) for those left-node conditions that can be achieved by more than one actual event (i.e., their corresponding nodes in the original graph $G$ have an in-degree greater than 1).

Hence, assume that $e_{x, left,x,right}^{1,1}$ can be achieved (implying the existence of a Hamiltonian path in $G$), the condition will then be falsified to achieve one in-node condition, which will be falsified to enable an edge event having not occurred yet, to occur. The occurrence of this edge event in turn will achieve an out-node event. Finally, $e_{x, left,x,right}^{1,1}$ will be achieved again. This procedure will be repeated until all left edge events have occurred, ensuring the existence of an admissible complete sequence. Note that, by extending the set of partial orders from $O_2$ to $O_2'$ (see below), even if an admissible sequence exists, all of those out-node events will have to occur after $e_{x, left,x,right}^{1,1}$ is achieved.

- $T' = T_{\text{edges}} \cup T_{\text{nodes}} \cup T_1 \cup T_2 \cup \ldots \cup T_n \cup T_x \cup T_y \cup T_{\text{from\_y\_right}} \cup T_{\text{to\_y\_right}}$ where
  - $T_{\text{edges}}$ and $T_{\text{nodes}}$ are the same as the ones in $\Theta$, 


Chapter 4. Possible Truth Problems

\[ \mathcal{T}_i = \mathcal{T}_{i,\text{left}} \cup \mathcal{T}_{i,\text{right}} \]

\[ \mathcal{T}_{x,\text{left}} \]

\[ \mathcal{T}_{y,\text{right}} = \{ t_{1,1}^1 \} \]

\[ \mathcal{T}_{x,\text{right}} = \{ t_{1,1}^1 \} \]

\[ \mathcal{T}_{i,\text{left}} = \{ t_{1,1}^1 \} \]

\[ \mathcal{T}_{i,\text{right}} = \{ t_{1,1}^1 \} \]

\[ \mathcal{R}' = \{ (t_{1,1}^1, \{ v_1^1 \}, \{ v_1^1 \}, \{ v_1^1 \}) | t_{1,1}^1 \in \mathcal{T}' \} \]

\[ \mathcal{E}' = \{ e_{1,1}^1 | t_{1,1}^1 \in \mathcal{T}' \} \]

\[ \mathcal{O}' = \mathcal{O}_1 \cup \mathcal{O}_2 \]

\[ \mathcal{I}' = \{ v_{x,\text{left}} \} \]

Example Begin

Consider the same (Figure 4.2) \( G = (V, E, v_x, v_y) \). The transformed Simple Event System \( \Theta \) (depicted in Figure 4.4) is a tuple of \( \langle P', \mathcal{T}', \mathcal{R}', \mathcal{E}', \mathcal{O}', \mathcal{I}' \rangle \) such that

\[ P' = P \cup P_{\text{out},x} \cup P_{\text{out},y} \]

\[ P_{\text{out},x} = \{ v_{2,\text{left}}^2 \} \]
Chapter 4. Possible Truth Problems

75

- \( P_{out,x} = \{ \} \),
- and \( P_{out,y} = \{ v_{y, left}^2 \} \);

- \( \mathcal{T}' = \mathcal{T}_{edges} \cup \mathcal{T}_{nodes} \cup \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_x \cup \mathcal{T}_y \cup \mathcal{T}_{from,y,right} \cup \mathcal{T}_{to,y,right} \) where

- \( \mathcal{T}_x = \mathcal{T}_{x, right} \),
- \( \mathcal{T}_1 = \mathcal{T}_{1, right} \),
- \( \mathcal{T}_2 = \{ t_2^{1,2, left, 2, left} \} \),
- \( \mathcal{T}_y = \{ t_2^{1,2, left, y, left} \} \),
- \( \mathcal{T}_{from,y,right} = \{ t_2^{1,2, y, right, x, right}, t_2^{1,2, y, right, 1, right} \} \),
- \( \mathcal{T}_{to,y,right} = \{ t_2^{2,1, 2, left, y, right}, t_2^{2,1, y, left, y, right} \} \);

- \( \mathcal{R}' = \mathcal{R} \cup \{ \langle t_2^{1,2, left, 2, left}, \{ v_2^{1, left} \}, \{ v_2^{2, left} \}, \{ v_2^{2, left} \} \rangle, \langle t_2^{1,2, y, left, y, left}, \{ v_2^{1, left} \}, \{ v_2^{2, left} \}, \{ v_2^{2, left} \} \rangle, \langle t_2^{1,2, y, right, x, right}, \{ v_2^{1, right} \}, \{ v_2^{2, right} \}, \{ v_2^{2, right} \} \rangle, \langle t_2^{1,2, y, right, 1, right}, \{ v_2^{1, right} \}, \{ v_2^{2, right} \}, \{ v_2^{2, right} \} \rangle, \langle t_2^{2,1, 2, left, y, right}, \{ v_2^{1, right} \}, \{ v_2^{2, right} \}, \{ v_2^{2, right} \} \rangle, \langle t_2^{2,1, y, left, y, right}, \{ v_2^{1, right} \}, \{ v_2^{2, right} \}, \{ v_2^{2, right} \} \rangle \} \);

- \( \mathcal{E}' = \mathcal{E} \cup \{ e_{2, left, 2, left}^{1,2}, e_{y, left, y, left}^{1,2}, e_{y, right, x, right}^{1,2}, e_{y, right, 1, right}^{1,2}, e_{2, left, y, right}^{2,1}, e_{y, left, y, right}^{2,1} \} \);

- \( \mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2' \) where

- \( \mathcal{O}_2' = \mathcal{O}_2 \cup \) \( \{ e_{y, left, y, right}^{1,1} < e | e \in \mathcal{E}_2 \cup \mathcal{E}_y \cup \mathcal{E}_{from,y,right} \cup \mathcal{E}_{to,y,right} \} \);  

- \( \mathcal{I} = \{ v_{x, left}^{1} \} \).

Note that in this example, the out-nodes \( v_{2, left}^{2} \) and \( v_{y, left}^{2} \) are introduced. In a circular manner, they are used to achieve \( v_{y, right}^{1} \), which in turn is used repeatedly to achieve \( v_{2, left}^{2} \) and \( v_{y, left}^{2} \). This is shown in Figure 4.4 by the sequence indexed from 1 to 10.

<table>
<thead>
<tr>
<th>Example End</th>
</tr>
</thead>
</table>

There exists a DHP from \( v_x \) to \( v_y \) in \( G \) iff \( v_{y, right}^{1} \in Poss_{\Lambda}^{+}(\Theta') \).

(\( \Rightarrow \)): If there exists a DHP \( p_{x,y} \) from \( v_x \) to \( v_y \) in \( G \), then, we can obtain the same \( f_1 \). Right after \( f_1 \), the y-right-node condition \( v_{y, right}^{1} \) is achieved, enabling any setting events
Chapter 4. Possible Truth Problems

Figure 4.4: The transformed Simple Event System

to occur. Randomly pick up one of them to occur will achieve its corresponding in-node condition. The occurrence of the corresponding in-node event will enable the occurrence of an edge event that has not occurred. The occurrence of this edge event must enable at least one out-node event. The occurrence of one of these out-node events will enable its corresponding resetting event, which achieve \( v_{y\_right}^1 \) again. In this iterative manner, all remaining edge events will occur and occur exactly once, indicating that there must exist a \( f \), which is an extension of \( f_1 \) and \( f \in ACS(\Theta') \) and \( v_{y\_right}^1 \in Result(\mathcal{I}, f) \).

(\( \Leftarrow \)): Now, assume there exists an admissible event sequence, i.e., \( f \in ACS(\Theta') \), such that \( v_{y\_right}^1 \) is achieved after \( f \), i.e., \( v_{y\_right}^1 \in Poss^1_\mathcal{A}(\Theta') \). Due to the specifications of the initial conditions \( \mathcal{I}' \) and the partial order constraints \( \mathcal{O}' \), we know that, one, the edge event \( e_{x\_left,x\_right}^{1,1} \) must be the first event of \( f \); two, the subsequence \( f/e_{y\_left,y\_right}^{1,1} \), (from \( e_{x\_left,x\_right}^{1,1} \) to \( e_{y\_left,y\_right}^{1,1} \) inclusive in \( f \), written as \( f_1 \)), must include all node events and some edge events, but exclude any in-node event, out-node event, setting event, or resetting event. Since \( f_1 \) is admissible, again, it corresponds to a Hamiltonian path from
v_x to v_y in G.

Example Begin

Now, a Hamiltonian path in G: \{v_x, v_1, v_2, v_y\} can have its corresponding admissible sequence
\{e_{x, x}^{1, 1}, e_{x, y}^{1, 1}, e_{1, x}^{1, 1}, e_{1, y}^{1, 1}, e_{2, x}^{1, 1}, e_{2, y}^{1, 1}, e_{x, y}^{1, 1}, e_{y, y}^{1, 1}, e_{x, x}^{2, 1}, e_{x, y}^{2, 1}, e_{1, x}^{2, 1}, e_{1, y}^{2, 1}, e_{2, x}^{2, 1}, e_{2, y}^{2, 1}, e_{x, y}^{2, 1}, e_{y, y}^{2, 1}\}

Example End

Theorem 13

Deciding \( p \in \text{Poss}^+(\Theta^\prime\prime) \) (i.e., determining the entailment problem \( \mathcal{D}_{es} \models (\exists s) (ACS(s) \land F(s)) \)), where \( \Theta^\prime\prime \) is an Almost-simple Event System and the cause-and-effect graph is a DAG, is NP-complete.

Proof. Given \( G = (V, E, v_x, v_y) \), we now construct \( \Theta^\prime\prime \), which is a modification to \( \Theta \). More specifically, the \( \mathcal{T}_{edges}' \) in \( \mathcal{T}'\prime\prime \) of \( \Theta^\prime\prime \) is defined as follows.

\[ \mathcal{T}_{edges}' = \{ t_{i, j, i, j}^{1, left}, t_{i, j, i, j}^{1, right} | (v_i, v_j) \in E \}. \]

That is, each edge in \( E \) is cut into two edges, corresponding to two unique event types in \( \mathcal{T}'\prime\prime \). Accordingly, we have

- two new conditions: \( v_{i, j}^{1, left} \) and \( v_{i, j}^{1, right} \), implying the fact that the two new nodes created by cutting the edge \((v_i, v_j)\) in \( E \) of \( G \),

- two new actual events \( e_{i, j, i, j}^{1, left} \) and \( e_{i, j, i, j}^{1, right} \) will be created accordingly in \( \mathcal{E}'\prime\prime \) of \( \Theta^\prime\prime \),

- in \( \mathcal{O}'\prime\prime \), it is now further required that \( t_{i, j, i, j}^{1, left} \prec t_{i, j, i, j}^{1, right} \),

- in \( \mathcal{T}'\prime\prime \), \( v_{i, j}^{1, right} \) is also set to true.

It can be easily verified that the transformed cause-and-effect graph is now a DAG, and the validity of the transformation still hold. The modification is illustrated using the same example (see Figure 4.5). \( \square \)
4.2.2 Tractable Computational Results

**Theorem 14** Deciding $p \in \text{Poss}^+_A(\Theta'')$ (i.e., determining the entailment problem $D_{es} = (\exists s)(\text{ACS}(s) \land F(s))$, where $\Theta''$ is a Simple Event System and the cause-and-effect graph $G$ for $T''$ is a DAG, is polytime solvable.

**Proof.** It is safe to assume that each $t \in T''$ has only one unique occurrence $e \in E''$ (hence, in this proof, it is the case that when we mention an *edge*, we are actually referring to its corresponding actual event):

- if there exist $e_1$ and $e_2$ both corresponding to a particular $t$, then there does not exist an admissible event sequence. That is, at least one of which ($e_1$ and $e_2$) can not occur (in other words, $p \not\in \text{Poss}^+_A(\Theta'')$), because $G$ is a DAG;

- if $t$ does not have actual occurrences, then simply remove its corresponding edge from $G$ to construct a $G'$, which is equivalent to the possible truth problem.

It is also safe to assume that, for any two $e_1$ and $e_2$ in $G$, there exists at least a path in between (say from $e_1$ to $e_2$), otherwise, there does not exists an Euler path for $G.$
Consequently, \( p \notin \text{Poss}^+_A(\Theta'') \). Now the problem becomes trivial. The in-degree and out-degree of any node say \( v_i \) in \( G \) are at most one: if two edges say \( e_1 \) and \( e_2 \) both are leaving \( v_i \), then there does not exist a path between them. Hence \( G \) is actually a trivial linearization and consequently the satisfiability of \( O'' \) can be checked in linear time. \( \square \)

The following theorem states that, when constraints of partial orderings are not present, admissible temporal projection problems are trivial.

**Theorem 15** Deciding \( p \in \text{Poss}^+_A(\Theta^*) \) (i.e., determining the entailment problem \( D_{es} \models (\exists s) (\text{ACS}(s) \land F(s)) \)), where \( \Theta^* \) is an Almost-simple Event System and the \( O \) in \( \Theta^* \) is empty, is polytime solvable.

**Proof.** The problem is equivalent to, for each condition initially set to true, finding a Euler path from this condition to the condition in investigation, whereas finding Euler path can be done in polynomial time [45]. \( \square \)

### 4.2.3 Remarks

The results from the previous two sections are summarized graphically in Figure 4.6.

It is observed from Figure 4.6 that

- a special Event System, whose set of partial orders is empty, is presented in the proof of Theorem 6.4 of [53], for the general Event System (illustrated in the Figure by “\( \leftarrow \)”). This is a clear indication that other constraints, not partial ordering alone, are indeed the sources of the intractability to the Possible Truth problem too;

- when without partial order, the intractability boundary occurs between general Event System and Almost-simple Event System, which means that the varying size of preconditions and effects in the causal rules contribute their own part to the intractability;

- the topological structure of the cause-and-effect graphs, and the size of the initial conditions, both contribute to the intractability, indicated by the fact that, as shown in the diagram, applying the DAG constraints and \( |I| = 1 \) (i.e., Simple Event System) alone can not bring the problem into the polytime solvable zone, whereas when applied together, the problem becomes tractable;
• however, when admissibility is removed from the tractable case at the bottom, the revised problem corresponds to the problem studied in [53], which is NP-complete (see Theorem 3.3 in [53]), but this fact means that “inadmissibility” contributes to the intractability as well. In fact, the inadmissible but effectless occurrence of event can also be interpreted as indefinite (i.e., disjunctive) preconditions for events.

To summarize, our study indicates that the constraints as follows all contribute to the intractability of the Possible Truth problem.

1. the set of partial orderings,

2. the size of the preconditions and effects,

3. the size of the initial condition,

4. the topological structure of the cause-and-effect graphs,

5. inadmissibility.

Figure 4.6: The computational hierarchy of admissible Possible Truth problems
4.3 Inadmissible Possible Truth

In this section, we consider two types of constraints that lead to tractability of inadmissible possible truth problems.\(^2\) The first constraint is on the graph-theoretic representation of the cause-and-effect relationships between events (Section 4.3.1). The second constraint is on the partial orders of events (Section 4.3.2).

### 4.3.1 2-layered Planar DAG

Now we will define the problem \(PAFP2L_{edges}\) and then prove that it is NP-complete.

**Definition 47 \((PAFP2L_{edges})\)** \(PAFP2L_{edges}\) involves finding a path from \(s\) to \(t\) with forbidden pairs of edges in a 2-Layered Planar \(s-t\)-DAG.

**Theorem 16** \(PAFP2L_{edges}\) is NP-complete.

**Proof.** It is easy to see that \(PAFP2L_{edges}\) is in NP: for any \(s-t\) path, it can be checked in polynomial time if the path satisfies the \(F_{edges}\).

To show that \(PAFP2L_{edges}\) is NP-hard, we will transform One-In-Three 3SAT (see [24] Page 259, and Section 3.2.2) to \(PAFP2L_{edges}\). From an arbitrary instance of One-In-Three 3SAT \(S = (U, C)\), where \(U = \{u_1, u_2, ..., u_n\}\) is a set of variables and \(C = \{c_1, c_2, ..., c_m\}\) such that \(|c_j| = 3\) for \(1 \leq j \leq m\) is the set of clauses, we construct a 2-Layered planar \(s-t\)-DAG \(G = (N, E)\). The node set \(N\) consists of a source \(s\), a sink \(t\), \(|m| - 1\) connectors \(\{cntr_i\} |1 \leq i \leq |m| - 1\}\), and two nodes, \(v_{ij}\) and \(d_{ij}\) \((d\) stands for “dummy”), for each literal \(p_{ij}\) in \(C\). Through application of the constraints of forbidden pairs \(F_1\) (defined below), the \(s\)-\(t\) path will pass either \(v_{ij}\) or \(d_{ij}\), but not both, reflecting the corresponding truth assignment of \(p_{ij}\) in \(C\) of \(S\). The edge set is

\[
E = \{(s, v_{i1}), (s, d_{i1})\} \cup \{(v_{i1}, d_{i2}), (d_{i1}, d_{i2}), (d_{i1}, v_{i2}) | 1 \leq i \leq m\} \cup \{(v_{i2}, d_{i3}), (d_{i2}, d_{i3}), (d_{i2}, v_{i3}) | 1 \leq i \leq m\} \cup \{(v_{i3}, cntr_i), (d_{i3}, cntr_i) | 1 \leq i < m\} \cup \{(v_{i3}, t), (d_{i3}, t) | i = m\}.
\]

The set of forbidden pairs \(F_{edges}\) is a union of \(F_1\) and \(F_2\), where

\[
F_1 = \{[(v_{i1}, d_{i2})(d_{i2}, v_{i3})], [(d_{i1}, d_{i2})(d_{i2}, d_{i3})] | 1 \leq i \leq m\}
\]

ensures a path will pass through one and only one literal node for each clause component, and \(F_2\) contains forbidden pairs

\(^2\)The work in this section is published in [74].
in which the two edges of each forbidden pair enter, respectively, two literal nodes that are negation of each other: $F_2 = \{ (node_1, v_{ij})(node_2, v_{kl}) | v_{ij} = \neg v_{kl} \}$, where $node_1$ and $node_2$ represent the unique predecessor nodes to $v_{ij}$ and $v_{kl}$, respectively.

This is obviously a polynomial time transformation. Now consider a constrained path $P$ from $s$ to $t$ in $G$, $F_1$ on $P$ makes one and only one corresponding literal in each $C_i$ of $S$ true whereas $F_2$ on $P$ ensures that the assignment is consistent. Hence, the existence of $P$ implies the One-In-Three satisfiability of $S$. The converse direction can be proved similarly.

**Example Begin**

An illustration of this construction is given in Figure 4.7. From an One-In-Three 3SAT instance $S = \{ [a, b, c], [b, \neg c, d] \}$, we construct $G = \langle N, E \rangle$ and $F_{edges}$, where

$$N = \{ v_{11}, v_{12}, v_{13}, v_{21}, v_{22}, v_{23} \} \cup \{ d_{11}, d_{12}, d_{13}, d_{21}, d_{22}, d_{23} \} \cup \{ S, T, Cntr1 \},$$

and $F_2 = \{ [(d_{12}, v_{13}),(d_{21}, v_{22})] \}$.

Elements of $E$ are shown in Figure 4.7 and the construction of forbidden pairs in $F_1$ can also be easily derived.

**Example End**

Finally, note that the general transformation in [23] produces a DAG that is not planar. While their restricted *in-and-out-degree-at-most-two* can only be further transformed into a planar DAG that is at least 3-layered.

In the proof of Theorem 3.3 of [53], NP-complete PAFP-E in general DAGs is transformed into the Possible Truth problem in a simple Event System $\Theta_{simple}$ to show that the problem is NP-hard. Similarly, we can obtain the main result of this section as follows, by transforming from the $PAFP2LP_{edges}$ problem (which is NP-complete, Theorem 16) into the current problem, leading to Theorem 17 as follows.

**Definition 48** An Event System $\Theta_{simple,2lp, stdag}$ is a simple event $\Theta_{simple}$ and its cause-and-effect graph is a 2-layered planar s-t-DAG.

**Theorem 17** The inadmissible Possible Truth problem in $\Theta_{simple,2lp, stdag}$ (i.e., determining the entailment problem $D_{es} \models (\exists s) CS(s) \land (Admi(s) \land F(s))$) is NP-complete.

It should be observed from Theorem 17 that, in principle, restricting the topological structure of cause-and-effect graphs alone can not reduce the complexity.
Figure 4.7: Transformation from an One-In-Three 3SAT Instance to a PAFP Problem
4.3.2 Partial Orders

As indicated in [39] and Proposition 7 in Chapter 2, PAFP-V in DAGs, with $F_{\text{vertices}}$ satisfying VHS, can be solved in polynomial time. The following theorem shows that the polynomial-time solvability can also be achieved on PAFP-E in DAGs, with $F_{\text{edges}}$ satisfying the EHS.

**Theorem 18** PAFP-E in DAGs, with $F_{\text{edges}}$ satisfying EHS, can be solved in polynomial time.

**Proof.** Here we present a polynomial time algorithm, which is a modification to the algorithm in Section 4 of [39].

The proof of the correctness of the algorithm consists of Lemma 19, which shows that applying Step 1, Step 2, or Step 3 will not affect the result on the existence of a PAFP-E, and Lemma 20, which indicates that when the algorithm finishes, a new problem with trivial solution is reduced from the original one.

**Lemma 19** Given $G = \langle N, E, F_{\text{edges}}, s, t \rangle$, which is an instance of PAFP-E, and $G' = \langle N', E', F'_{\text{edges}}, s, t \rangle$, which is obtained through applications of Step 1, Step 2, and Step 3 to $G$, there exists a path $P$ as a solution for $G \iff$ there exists a path $P'$ as a solution for $G'$.

"$\Rightarrow$" direction: Given $P \subseteq G$ there are three cases to consider

1. $P \subseteq G'$;
2. $P$ contains a free edge $e_{\text{free}} = (x, y)$ that is removed in $G'$ by applying Step 1;
3. $P$ contains $(x, y)(y, z)$, but by applying Step 2, $(x, y)$ is removed and $(x, z)$ is added in $G'$.

For case 1, $P$ is also a solution for $G'$. For case 2, remove $e_{\text{free}}$ in $P$ and concatenate the two segments to obtain $P'$, which is a solution for $G'$. For case 3, replacing $(x, y)(y, z)$ in $P$ with $x, z$ to obtain $P'$. Note that applying Step 3 does not affect the solution. The proof for the "$\Leftarrow$" direction is similar.

**Lemma 20** Given $G = \langle N, E, F_{\text{edges}}, s, t \rangle$, if $G = G'$ (i.e., if none of Step 1, Step 2, or Step 3 can be applied to $G$), then $N = \{s, t\}$. 
Algorithm 1 The algorithm to find path in DAGs with forbidden pairs of edges

Input: $G = (N, E, F_{edges}, s, t)$, where $s$ is the source node and $t$ is the sink node in $N$: $G' = G$

while $|N| > 2$ do

$G = G'$

Step 1: find a free edge say $e_1$ such that $e_1 \in N'$ and $e_1 = (x, y)$ (i.e., there does not exist another edge say $e_2$ such that $[e_1e_2] \in F'_{edges}$ or $[e_2e_1] \in F'_{edges}$); replace $x, y \in N'$ by a new node $u \in N'$ such that, in $E'$, any edge $(v, x)$ or $(v, y)$ is replaced by $(v, u)$ and any edge $(x, w)$ or $(y, w)$ is replaced by $(u, w)$; remove $(x, y)$ from $E'$; that is,

$N' \leftarrow (N' - \{x, y\} \cup \{u\})$,
$E' \leftarrow (E' - \{(v, x), (v, y)\} \cup \{(v, y)\} \cup \{(v, u)\})$,
$E' \leftarrow (E' - \{(x, w), (y, w)\} \cup \{(x, w)\} \cup \{(u, w)\})$,
$E' \leftarrow (E' - \{(x, y)\})$;

Step 2: find an edge pair $[(x, y)(y, z)] \in F'_{edges}$ and for any edge $(y, u) \in E'$ such that $u \neq z$ add an edge $(x, u)$ to $E'$; for any pair in $F'_{edges}$ that involves $(x, y)$, add a new pair where $(x, y)$ is replaced by $(x, u)$; remove $(x, y)$ from $E'$, remove any pairs that contains $(x, y)$ from $F'_{edges}$; that is,

$E' \leftarrow (E' \cup \{(x, u)\} \cup \{(x, y)(y, z)\} \in F'_{edges}, (y, u) \in E', u \neq z))$,
$E' \leftarrow (E' - \{(x, y)\})$,
$F'_{edges} \leftarrow (F'_{edges} \cup \{(x, u)(z_1, z_2)\} - \{(x, y)(z_1, z_2)\})$;

Step 3: remove any edge pair $[(x, y)(u, v)] \in F'_{edges}$ such that there does not exist a path from $(x, y)$ to $(u, v)$; that is,

$F'_{edges} \leftarrow (F'_{edges} - \{(x, y)(u, v)\} \cup \{(x, y) \not\in E' \ (u, v)\})$;

end while

if $E' = \emptyset$ then

print: There Does Not Exist A Path

else

print: There Exists A Path

end if
We prove this by contradiction. Suppose, on the contrary, there exists an edge \( e_1 \), \( e_1 \) with some other edge \( e_2 \) must appear in \( F_{edges} \) otherwise Step 1 can be applied. The forbidden pair \([e_1,e_2]\) must be separated by some \( e_3 \) otherwise Step 2 can be applied. Similarly, \( e_3 \) must also be in \( F_{edges} \) with some \( e_4 \). But all of the following cases

1. \( e_1 \prec_E e_3 \prec_E e_4 \prec_E e_2 \);
2. \( e_1 \prec_E e_3 \prec_E e_2 \), \( e_3 \prec_E e_4 \) and \( e_4 \not\prec_E e_2 \);
3. \( e_1 \prec_E e_4 \prec_E e_2 \), \( e_3 \prec_E e_4 \) and \( e_1 \not\prec_E e_3 \)

will not be allowed, otherwise we can start with \( e_3 \prec_E e_4 \) with two extra edges \( e_1 \) and \( e_2 \). The two remaining cases

1. \( e_3 \prec_E e_1 \prec_E e_4 \prec_E e_2 \);
2. \( e_1 \prec_E e_3 \prec_E e_2 \prec_E e_4 \)

violate EHS.

Given \(|N| = n\) and \(|E| = m\) (assume that \( n < m \)), we have \(|F_{edges}| \leq m^2\). In Step 1, searching for a free edge in \( F_{edges} \) and then removing the edge if one is found, requires \( O(m) \) time if an appropriate data structure such as an array of linked lists is applied to store the forbidden pairs in \( F_{edges} \), concatenating the two ends of the removed edge into a new node requires \( O(n) \) time, thus Step 1 is bounded by \( O(m) \). In Step 2, searching for a forbidden pair connected by a node requires \( O(m^2) \) by brute force, the subsequent operations if one such pair is found require another \( O(n) \), making Step 2 \( O(m^2) \) bounded. As for Step 3, since the redundancy check for each forbidden pair requires an \( O(m) \) time breadth-first or depth-first search, Step 3 is bounded by \( O(m^3) \). Observe that each application of Step 2 or Step 3 will reduce the size of \( F_{edges} \) by one, the number of iterations is bounded by \( O(m^2) \). The overall time complexity is thus bounded by \( O(m^4) \). Note that it is likely that a more sophisticated implementation results in a reduced complexity.

Finally, it should be noted that the algorithm can be easily modified to print out the PAEP-E, if one exists in the original graph. \( \square \)

Next we show the implication of Theorem 18 on Possible Truth problems.

**Definition 49** An Event System \( \Theta_{simple\_dag} \) is a Simple Event System and its cause-and-effect graph is an acyclic digraph.
Definition 50 An Event System $\Theta_{\text{orders simple dag}}$ is a $\Theta_{\text{simple dag}}$ that satisfies the following additional condition: if $a_1 \prec a_2$ is in its set of pairwise disjoint partial orders $\mathcal{O}$, then there exists a path in the cause-and-effect graph $G$ from $a_2$ to $a_1$ (i.e., $a_2 \prec_E a_1$).

It should be noted that, from the above definition, we have in $\Theta_{\text{orders simple dag}}$ 1) $a_2 \prec_E a_1$ implies $a_1 \not\prec_E a_2$, as $G$ is a DAG; 2) if $a_1 \prec a_2$, there does not exist any permutation of $\mathcal{A}$, in which both $a_1$ and $a_2$ are applicable, because we have $a_2 \prec_E a_1$ in $G$.

Theorem 21 The inadmissible Possible Truth problem in $\Theta_{\text{orders simple dag}}$ has a solution (i.e., achieves condition $t$ from some initial condition $s$ if and only if the PAFP-E in $\langle G, F_{\text{edges}} \rangle$, derived by the rule $a_1 \prec a_2 \in \mathcal{O} \iff [a_2, a_1] \in F_{\text{edges}}$, has a PAFP-E from $s$ to $t$.

Proof. Suppose in an inadmissible Possible Truth problem, there exists a sequence $seq$, which is a permutation $\mathcal{A}$ and achieves the goal condition $t$ from the initial condition $s$. Those applicable events in $seq$ construct a subsequence $seq_{\text{app}}$ in the form $[s, \ldots, t]$, which has a one-to-one correspondence to a path in $G$ in the form $path_{\text{app}} = [n(s), \ldots, n(t)]$ in $G$. By the above rule, and the fact that $seq_{\text{app}}$ also satisfies $\mathcal{O}$, any two edges constructing a forbidden pair cannot both appear in $path_{\text{app}}$. Hence, $path_{\text{app}}$ is a PAFP-E in $G$.

Suppose the derived $\langle G, F_{\text{edges}} \rangle$ has a PAFP-E $path_{\text{forbidden}} = [n(s), \ldots, n(t)]$. The path $path_{\text{forbidden}}$ corresponds to a sequence of applicable events in $\Theta_{\text{orders simple dag}}$, written as $seq_{\text{app}}$. Now, we can partition $\mathcal{A}$ into $\mathcal{A}_1 \cup \mathcal{A}_2$, and $\mathcal{O}$ into $\mathcal{O}_1 \cup \mathcal{O}_2$. Assume that no event in $seq_{\text{app}}$ ever appears in $\mathcal{A}_1$ or $\mathcal{O}_1$, we can obtain a sequence $seq_{\text{irrt}}$, which is irrelevant to $seq_{\text{app}}$, by running topological sorting on $\mathcal{O}_1$. Given that, first, partial orders in $\mathcal{O}$ are pairwise disjoint, and second, for any $(a_1 \prec a_2) \in \mathcal{O}_2$, exactly one of $a_1$ or $a_2$ will appear in $seq_{\text{app}}$, it is safe to extend $seq_{\text{app}}$ to $seq_{\text{ext}}$ by inserting $a_1$ before $a_2$ if $a_2$ is in $seq_{\text{app}}$, and vice versa. The concatenation $seq_{\text{ext}} + seq_{\text{irrt}}$ is a permutation of $\mathcal{A}$ that can achieve $t$ from $s$. □

Definition 51 An Event System $\Theta_{\text{orders ehs simple dag}}$ is a $\Theta_{\text{orders simple dag}}$ that satisfies the following additional condition: if an $F_{\text{edges}}$ is constructed such that $a_1 \prec a_2 \in \mathcal{O} \iff [a_2, a_1] \in F_{\text{edges}}$, $\langle G, F_{\text{edges}} \rangle$ is with EHS.

The main result of this section is stated as follows.

Theorem 22 The inadmissible Possible Truth problem in $\Theta_{\text{orders ehs simple dag}}$ (i.e., determining the entailment problem $\mathcal{D}_{\text{es}} \models (\exists s) (CS(s) \land F(s))$) can be solved in polynomial time.
Proof. By Theorem 21, it is safe to transform the inadmissible Possible Truth problem in $\Theta^{\text{orders,ehs}}_{\text{simple, dag}}$ into $\langle G, F_{\text{edges}} \rangle$ satisfying EHS, for solution. By Theorem 18, $\langle G, F_{\text{edges}} \rangle$ satisfying with EHS can be solved in polynomial time. □

4.3.3 Remarks

The most restricted inadmissible Possible Truth problem known to be NP-complete is the one in $\Theta_{\text{simple}}$. Starting from this problem, we pushed the tractability boundary by adding additional constraints on the topology of the graph indicating the cause-and-effect relationships and on the structure of the set of partial orders. In particular, we showed in this section that the inadmissible Possible Truth problem in $\Theta_{\text{simple}}$ remains NP-complete if its cause-and-effect graph is a 2-layered planar s-t-DAG, whereas it is tractable if the graph is associated in a special way with the set of partial orders, and the set is in EHS.

Layered planar digraphs are special DAGs in which efficient parallel algorithms exist for computing the shortest path [70]. From this perspective, the NP-completeness result for PAFP of edges in 2-layered planar s-t-DAGs should have significance in its own right. One interesting observation is that, 2-layered planar s-t-DAGs are not series-parallel decomposable (see Theorem 1 in [79] for recognition of series-parallel decomposable DAGs with respect to N graph as induced subgraph); meanwhile, it can be derived from the construction in [23] that PAFP of edges in series-parallel-decomposable DAGs is NP-complete.

The paper [39] also introduces a notion of halving structure that leads to NP-complete subclass of PAFP of vertices; the structure is of little use for the purposes of this current work since tractability is obtained from the EHS. Nevertheless, it seems that other structures on partial orders could also lead to tractable inadmissible Possible Truth problems: in [85], for example, Yinnone proposes a so-called skew symmetry for forbidden pairs of vertices and shows that PAFP under this restriction can be solved in polytime. We believe this result can be adopted to obtain another class of tractable inadmissible Possible Truth problems, using the same method applied as above.

Main results presented in this section are graphically represented in Fig 4.8. These results, particularly the NP-completeness one, are in favor of Nebel and Bäckström’s claim on Page 135 of [53], that is, when admissibility is not required, it seems that the structure of cause-and-effects does not contribute much to the intractability of temporal projection problems and the set of partial orderings is almost exclusively the source of complexity.
However, given that the tractability of the other result in this section depends on the close association between the set of orders and the cause-and-effect graph (see Theorem 11), the complete validity of this claim demands further comprehensive investigations (see the whole Section 3.3, for example). One obvious proposal for future work would be: keep the EHS, but relax other constraints (for example, allowing multiple preconditions and effects; multiple rules for events), check whether intractability could possibly be restored.

![Diagram of computational hierarchy of inadmissible Possible Truth problems](image-url)

Figure 4.8: The computational hierarchy of inadmissible Possible Truth problems
Chapter 5

Partially Ordered Activities in PSL

In this chapter, we will show how to define, with the PSL Ontology, the antecedents and consequents for first-order entailment reasoning problems related to the partial ordering of subactivity occurrences in occurrences of complex activities. In addition, we will define different extensions of the PSL Ontology such that the associated reasoning problems are NP-complete or tractable.\(^1\)

5.1 Introduction

Ontologies that represent complex activities (such as composite web services and manufacturing process plans) are required for many applications of automated reasoning. We typically need to specify the occurrence and ordering constraints over the different subactivities; such constraints may either be explicitly represented or they may be entailed by other properties, such as preconditions and effects.

Within domains of semantic web service discovery, resources are associated with composite web service plans, which are partially ordered sequences of processes. In general, such process plans may also be nondeterministic (that is, involve different choices of sequences of processes). At any point in a web service plan, there are multiple activities that can possibly occur next. Furthermore, different web service plans may have processes in common so that an object may participate in an activity that is part of multiple plans. This scenario motivates four queries that are relevant for the discovery and verification of semantic web service plans, and which we will formalize later in this

---

\(^1\)The work in this chapter is published in [32].
chapter:

1. Is it possible for one activity in a web service process to occur before some other activity?

2. Is one activity in a web service process required to occur before some other activity?

3. Given the occurrence of some activity in a web service process, what activities are required to occur later?

4. Given the occurrence of some activity in a web service process, what activities can possibly occur next?

In this Chapter we will focus on the formalization of these four queries as first-order entailment problems which are related to occurrences of complex activities and the ordering constraints on those occurrences. To achieve this objective, we use a first-order ontology in which complex activities and their occurrences are elements of the domain, so that web service discovery can be expressed as entailment problems that are solved by inference techniques that are sound and complete with respect to models of the ontology. Inference is done using the axioms of the ontology alone, without resorting to extralogical assumptions or special algorithms used by interpreters.

In particular, we will use the PSL Ontology to axiomatize the constraints in the antecedents, as well as the queries in the consequents, of the entailment problems. Furthermore, we use the model theory of the PSL Ontology to provide correctness theorems for the specification of the queries and the process descriptions for certain classes of activities. Finally, we will define extensions of the PSL Ontology in which two of the entailment problems are NP-complete, and additional extensions in which these entailment problems are tractable. In this way, tractable classes of the problems are explicitly axiomatized within the ontology itself, and the relationships between different assumptions can themselves be determined by first-order theorem proving.

5.2 Formalization of the Entailment Problems

We can use the PSL Ontology to specify the queries (informally posed in the introduction) as the consequents of first-order entailment problems. The antecedent of the entailment problem

\[ T \models \phi \]

Given the entailment problem \( T \models \phi \), we say that \( T \) is the antecedent and \( \phi \) is the consequent.

\[^2\text{Given the entailment problem } T \models \phi , \text{ we say that } T \text{ is the antecedent and } \phi \text{ is the consequent.}\]
problems will consist of the following sets of sentences:

- \( T_{\text{psl}} \): the axioms of the PSL Ontology, together with the following three sentences:
  
  - Activity closure (primitive and complex)
    
    \[
    \forall a \ (\text{primitive}(a) \equiv (a = A_1) \lor \ldots \lor (a = A_n))
    \]
    
    \[
    \forall a \ (\neg \text{atomic}(a) \equiv (a = P_1) \lor \ldots \lor (a = P_m))
    \]
    
    where \( A_1, \ldots, A_n, P_1, \ldots, P_m \) are constants denoting activities.
  
  - Legal Occurrence Assumption\(^3\)
    
    \[
    \forall o, a \ (\text{occurrence_of}(o, a) \land \text{atomic}(a) \supset \text{legal}(o))
    \]
  
- \( \Sigma_{\text{pd}}(P_i) \): the process description for the complex activity \( P_i \), which specifies the relationship between occurrences of the activity and its subactivities.

5.2.1 Queries

In this section, we first focus on the queries in the consequents of the problems, and then define the classes of activities and process descriptions that constitute the antecedents of the problems. The query in the consequent of one of our entailment problems is a first-order sentence that is satisfied by properties of the activity trees within the models of \( T_{\text{psl}} \) and the process descriptions. We can apply the model theory of the PSL Ontology to provide characterizations of the activity trees for each query that we consider, demonstrating the correctness of the sentence with respect to the intended properties of the activity trees. In the motivating scenarios from the introduction, process plans and composite web services are represented in the PSL Ontology as complex activities. The four particular queries that we formalize here focus on the relationship between occurrences of complex activities and their subactivities.

To formalize the first query, we want a sentence that determines whether the subactivity \( A_1 \) can possibly occur before the subactivity \( A_2 \), whenever the complex activity \( P \) occurs; such a sentence is characterized by the following result:

**Lemma 23** Suppose \( \mathcal{M} \) is a model of \( T_{\text{psl}} \cup \Sigma_{\text{pd}}(P) \).

\[
\mathcal{M} \models (\forall o) \ (\text{root}(o, P) \supset (\exists o_1, o_2)\text{occurrence_of}(o_1, A_1) \land \text{soo_precedes}(o, o_1, P) \land
\]

\(^3\)We use this assumption in the complexity analysis to focus on the intractability that arises solely from occurrence and ordering constraints, independently of preconditions and effects.
occurrence_of(o_2, A_2) \land soo\_precedes(o, o_2, P) \land soo\_precedes(o_1, o_2, P))

iff any activity tree for the complex activity P contains a branch in which a subactivity occurrence o_1 of A_1 precedes a subactivity occurrence o_2 of A_2.

**Proof.** Suppose that o is the root of an activity tree τ for P in M. By the definition of the soo\_precedes relation,

\[ o_1, o_2 \in τ \iff ⟨o, o_1, P⟩, ⟨o, o_2, P⟩ \in soo\_precedes. \]

Furthermore, \(⟨o_1, o_2, P⟩ \in soo\_precedes\) implies that there exists a branch \(B \subseteq τ\) such that \(o_1, o_2 \in B\) and that \(o_1\) precedes \(o_2\) along this branch. □

The activity tree in Figure 5.1 contains a branch in which the subactivity hotel occurs before airplane and also contains a branch in which the subactivity airplane occurs before hotel. In the same activity tree, there does not exist a branch in which the payment subactivity occurs before the register subactivity. In Figure 5.2, there does not exist any branch containing occurrences of both subactivities airplane and train. The following lemma characterizes the sentence that is satisfied when the subactivity A_1 is required to occur before the subactivity A_2 in occurrences of the complex activity P:

**Lemma 24** Suppose M is a model of T_{psl} \cup Σ_{pd}(P).

\[ M \models (∀o, o_1, o_2) (root(o, P) \land occurrence\_of(o_1, A_1) \land occurrence\_of(o_2, A_2) \land soo\_precedes(o, o_1, P) \land soo\_precedes(o_2, P)) \land ¬soo\_precedes(o_2, o_1)) \]

iff for any branch \(B\) in any activity tree for the complex activity P, either

1. every occurrence of the subactivity A_1 in \(B\) precedes every occurrence of the subactivity A_2 in \(B\), or

2. \(B\) does not contain occurrences of both A_1 and A_2.

**Proof.** Let τ be an activity tree for P in M. The sentence is logically equivalent to

\[ (∀o)(root(o, P) \supset ¬(∃o_1, o_2) occurrence\_of(o_1, A_1) \land soo\_precedes(o, o_1, P) \land occurrence\_of(o_2, A_2) \land soo\_precedes(o_2, o_1, P)) \land soo\_precedes(o, o_2, P) \land soo\_precedes(o_1, o_2, P)), \]

which is equivalent to saying that an occurrence of A_2 is never an element of a subtree of τ that is rooted in an occurrence of A_1. Thus, occurrences of A_1 and A_2 are either
on different branches, or all occurrences of $A_1$ precede all occurrences of $A_2$ whenever they are on the same branch.

For example, in Figure 5.4, the subactivity hotel occurs before payment on every branch of the activity tree. On the other hand, there is no branch in Figure 5.2, that contains occurrences of both train and airplane.

The third query from the introduction determines whether occurrences of the subactivity $A_1$ are followed by later occurrences of the subactivity $A_2$ in occurrences of the complex activity $P$. This sentence is characterized by the next result:

**Lemma 25** Suppose $M$ is a model of $T_{psl} \cup \Sigma_{pd}(P)$.

$$M \models (\forall o, o_1) (\text{root}(o, P) \land \text{occurrence of}(o_1, A_1) \land \text{soo precedes}(o, o_1, P) \supset
(\exists o_2) (\text{occurrence of}(o_2, A_2) \land \text{soo precedes}(o_1, o_2, P)))$$

iff every occurrence of the subactivity $A_1$ is the initial element of a subtree of an activity tree for $P$ that contains an occurrence of the subactivity $A_2$.

**Proof.** Every element $o_1$ of an activity tree $\tau$ for $P$ that is an occurrence of $A_1$ and that is not a leaf is the root of a subtree $\tau' \subseteq \tau$.

$$o_2 \in \tau' \text{ iff } \langle o_1, o_2, P \rangle \in \text{soo precedes}.$$ 

Thus, the subtree of $\tau$ rooted in an occurrence of $A_1$ contains an occurrence of $A_2$.  

Figure 5.2 illustrates this query, where every occurrence of the subactivity train is followed by an occurrence of the subactivity payment.

The final query determines, given the occurrence of some activity $A_1$ in a process plan $P$, which activities can possibly occur next, i.e., right after the occurrence. The following lemma characterizes the sentence that defines this query:

**Lemma 26** Suppose $M$ is a model of $T_{psl} \cup \Sigma_{pd}(P)$.

$$M \models (\forall o, o_1) (\text{root}(o, P) \land \text{occurrence of}(o_1, A_1) \land \text{soo precedes}(o, o_1, P) \supset
(\exists a, o_2) (\text{occurrence of}(o_2, a) \land \text{soo precedes}(o_1, o_2, P) \land
\neg((\exists o_3) \text{soo precedes}(o_1, o_3, P) \land \text{soo precedes}(o_3, o_2, P))))$$

iff no occurrence $o_1$ of the subactivity $A_1$ is a leaf of an activity tree for $P$.

**Proof.** An element $o_1$ of an activity tree for $P$ is a leaf of $\tau$ iff there is no element $o_4 \in \tau'$ such that $\langle o_1, o_4, P \rangle \in \text{soo precedes}$. Since the activity trees are discrete, $o_1$ is
covered by some \( o_2 \in \tau \), which is equivalent to saying that there does not exist \( o_3 \in \tau \) such that

\[
\langle o_1, o_3, P \rangle, \langle o_3, o_2, P \rangle \in \text{soo_precedes}.
\]

\[\square\]

In any proof of the above sentence with answer extraction, the variable \( a \) binds to one of the successors of the occurrence of \( A_1 \) in an activity tree for \( P \). In Figure 5.2, the next subactivity to occur after \( o_2^{\text{hotel}} \) is either \textit{airplane} or \textit{train}.

## 5.2.2 Classification of Activities

One set of sentences within the antecedent of the entailment problems is the extension of the ontology with restricted classes of activities. Within the PSL Ontology, complex activities are classified with respect to symmetries of their activity trees. Concretely, these are axiomatized by mappings between the different branches of an activity tree or between different activity trees. In this section we introduce the model-theoretic definitions for the classes of activity trees that play a prominent role in this chapter; their first-order axiomatization can be found in the PSL Ontology.\(^4\)

**Definition 52** An activity tree \( \tau \) in a model of \( T_{\text{psl}} \) is permuted iff for every two branches \( B_1, B_2 \subseteq \tau \), there exists a bijection \( \varphi : B_1 \to B_2 \) such that for any activity occurrence \( o \in B_1 \) and activity \( a \),

\[
\langle o, a \rangle \in \text{occurrence_of} \iff \langle \varphi(o), a \rangle \in \text{occurrence_of}
\]

Figure 5.1 shows an example of a permuted activity tree; there is a bijection that maps the subactivity occurrences \( o_2^{\text{hotel}}, o_3^{\text{airplane}}, o_6^{\text{payment}} \) in the branch \( B_1 \) to the subactivity occurrences \( o_5^{\text{hotel}}, o_7^{\text{airplane}}, o_4^{\text{payment}} \), respectively, in the branch \( B_2 \). Since the occurrence \( o_1^{\text{register}} \) is an element of every branch, it is mapped to itself. Intuitively, each branch of a permuted activity tree is a different permutation of the same set of subactivity occurrences; in the example, the same activities (\textit{register}, \textit{hotel}, \textit{airplane}, and \textit{payment}) occur on each branch, although they occur in a different order. On the other hand, the activity tree in Figure 5.2 is not permuted, since there is no mapping between the branch containing \( o_3^{\text{airplane}} \) and \( o_8^{\text{train}} \).

\(^4\)The axioms in the Common Logic Interchange Format are available at [http://www.mel.nist.gov/psl/ontology.html](http://www.mel.nist.gov/psl/ontology.html)
Chapter 5. Partially Ordered Activities in PSL

Definition 53 An activity tree $\tau$ in a model of $T_{psl}$ is folded iff there exists a branch $B_1 \subseteq \tau$ such that for any branch $B_2 \subseteq \tau$ there exists a surjection $\varphi : B_2 \to B_1$ such that for any activity occurrence $o \in B_1$ and activity $a$,

$$\langle o, a \rangle \in \text{atocc} \Rightarrow \langle \varphi(o), a \rangle \in \text{atocc}$$

With folded activity trees, the mappings between branches of the activity tree allow occurrences of atomic subactivities to be mapped to occurrences of concurrent subactivities. Figure 5.3 shows an example of a folded activity tree; the two subactivity occurrences $o_2^{\text{register}}$ and $o_4^{\text{payment}}$ on the branch $B_2$ are mapped to the subactivity occurrence $o_6^{(\text{register+payment})}$, which is an occurrence of the atomic activity whose primitive subactivities $\text{payment}$ and $\text{register}$ are concurrent.
Each activity tree can be associated with a partial ordering that is preserved by the mappings between branches, so that activity trees can be classified with respect to the relationship between this ordering and branches of the trees. This leads to two subclasses of permuted and folded activity trees that are particularly relevant to the specification of manufacturing process plans and semantic web services.

**Definition 54** Within a model of $T_{psl}$, a permuted activity tree is a strong poset activity tree iff there exists a partial linear order such that there is a one-to-one correspondence between its linear extensions and branches of the tree.

Figure 5.4 is an example of a strong poset activity tree. The subactivities hotel and airplane are incomparable in the partial ordering (since the ordering of the occurrences of these two activities is not preserved on each branch), so there are two branches in the activity tree, corresponding to the two possible linear extensions.

**Definition 55** Within a model of $T_{psl}$, a folded activity tree is a concurrent poset activity tree iff there is a one-to-one correspondence between branches of the tree and the weak orderings (i.e., binary relations that are irreflexive, antisymmetric, and transitive) of some set.

Figure 5.5 is an example of a concurrent poset activity tree. The subactivities hotel and register are incomparable in the partial ordering, so there are two branches in the activity tree, corresponding to the two possible linear extensions; there is also a branch in the activity tree containing the occurrence of the activity in which hotel and register are concurrent. Note that any concurrent poset tree contains a subtree that is a strong poset tree.

Strong poset and concurrent poset activity trees capture the intended semantics of constructs that are present in a wide variety of approaches to process modeling, including...
5.2.3 Process Descriptions

A process description is an axiomatization of the set of activity trees for a complex activity within models of the PSL Ontology. The syntactic form of the process description is tightly constrained by the classes of activities in the ontology.

**Theorem 27** A complex activity $P$ has a set of finite permuted activity trees iff its process description $\Sigma_{pd}(P)$ is logically equivalent to a sentence of the form

$$\forall o \left( \text{occurrence}_o(o, P) \supset (\exists o_1, ..., o_n \text{ occurrence}_o(o_1, A_1) \land ... \land \text{occurrence}_o(o_n, A_m) \right)$$
\begin{equation}
\land \text{subactivity}(A_1, P) \land \ldots \land \text{subactivity}(A_m, P) \land O(o_1, \ldots, o_n, P) \land ((\forall s) \text{arboreal}(s) \supset \\
\text{subactivity}\_\text{occurrence}(s, o) \equiv ((s = o_1) \lor \ldots \lor (s = o_n)))
\end{equation}

where \(O(o_1, \ldots, o_n, P)\) is a boolean combination of soo\_precedes literals whose only variables are \(o_1, \ldots, o_n\).

**Proof.**  \(\Rightarrow:\)

Suppose that the complex activity \(P\) has finite permuted activity trees.

Since the activity trees are finite, all of their branches are finite. Furthermore, since there is a bijection between the branches in the tree, all branches have the same cardinality \(n\). Thus, each branch consists of \(n\) occurrences of subactivities of \(P\), so that we have

\[
T_{psl} \cup \Sigma_{pd}(P) \models (\forall o) \left( \text{root}(o, P) \supset \\
[(\exists o_1, \ldots, o_n) \text{occurrence}\_of(o_1, A_1) \land \ldots \land \text{occurrence}\_of(o_n, A_m) \land \\
\text{subactivity}(A_1, P) \land \ldots \land \text{subactivity}(A_m, P) \land O(o_1, \ldots, o_n, P)] \right).
\]

Each of these is an arboreal subactivity occurrence of an occurrence of \(P\), and all arboreal subactivity occurrences within an occurrence of \(P\) are elements of a branch of an activity tree. We therefore have

\[
T_{psl} \cup \Sigma_{pd}(P) \models (\forall o) \left( \text{root}(o, P) \supset \\
[(\exists o_1, \ldots, o_n) \text{occurrence}\_of(o_1, A_1) \land \ldots \land \text{occurrence}\_of(o_n, A_m) \land \\
((\forall s) \text{arboreal}(s) \supset \text{subactivity}\_\text{occurrence}(s, o) \equiv ((s = o_1) \lor \ldots \lor (s = o_n))))\right)
\]

\(\Leftarrow:\)

Suppose that \(P\) has the process description \(\Sigma_{pd}\).

This process description entails that every occurrence of \(P\) has exactly \(n\) occurrences of atomic subactivities of \(P\). Therefore, each branch of an activity tree for \(P\) has occurrences of the same set of atomic subactivities, so that we can define a bijection between any two branches of the activity tree. Hence, the activity tree is permuted. \(\square\)

For example, the process description for the activity \(P_1\) in Figure 5.1 is

\[
(\forall o) \left( \text{occurrence}\_of(o, P_1) \supset [(\exists o_i, o_j, o_k, o_l) \text{occurrence}\_of(o_i, \text{register}) \land \\
\text{occurrence}\_of(o_j, \text{hotel}) \land \text{occurrence}\_of(o_k, \text{airplane}) \land \text{occurrence}\_of(o_l, \text{payment}) \land \\
\text{occurrence}\_of(o_m, \text{flight}) \land \text{occurrence}\_of(o_n, \text{ticket}) \land \text{occurrence}\_of(o_p, \text{payment}) ] \right)
\]
Chapter 5. Partially Ordered Activities in PSL

\[ \text{subactivity}(\text{register}, P_1) \land \text{subactivity}(\text{hotel}, P_1) \land \text{subactivity}(\text{airplane}, P_1) \land \text{subactivity}(\text{payment}, P_1) \land \text{soo\_precedes}(o_i, o_j, P_1) \land \text{soo\_precedes}(o_j, o_k, P_1) \land (\text{soo\_precedes}(o_i, o_k, P_1) \equiv \neg \text{soo\_precedes}(o_k, o_i, P_1)) \land ((\forall s) \text{arboreal}(s) \supset \\
\text{subactivity\_occurrence}(s, o) \equiv ((s = o_i) \lor (s = o_j) \lor (s = o_k) \lor (s = o_l))) \]

For folded activity trees, we have a similar, albeit weaker, result:

**Theorem 28** If the complex activity \( P \) has a set of finite folded activity trees, then its process description \( \Sigma_{pd}(P) \) entails a sentence of the form

\[
(\forall o) \left( \text{occurrence\_of}(o, P) \supset [ (\exists o_1, ..., o_n) \text{atocc}(o_1, A_1) \land ... \land \text{atocc}(o_n, A_m) \land \\
\text{subactivity}(A_1, P) \land ... \land \text{subactivity}(A_m, P) \land \mathcal{O}(o_1, ..., o_n, P) \land ((\forall s) \text{arboreal}(s) \supset \\
\text{subactivity\_occurrence}(s, o) \equiv ((s = o_1) \lor ... \lor (s = o_n))) \right)
\]

where \( \mathcal{O}(o_1, ..., o_n, P) \) is a boolean combination of \( \text{soo\_precedes} \) and equality literals whose only variables are \( o_1, ..., o_n \).

**Proof.** Suppose that the complex activity \( P \) has finite folded activity trees.

Since the activity trees are finite, all of their branches are finite. Furthermore, since there is a surjection from the branches in the tree into a unique maximal branch, this branch has the maximum cardinality \( n \). Thus, each branch consists of at most \( n \) occurrences of subactivities of \( P \).

By the definition of folded activity trees, for each element \( s \) of a branch of an activity tree that is an occurrence of the subactivity \( a \) of \( P \), there exists an element \( s' \) of the maximal branch such that

\[ (s', a) \in \text{atocc} \]

We therefore have

\[
T_{psl} \cup \Sigma_{pd}(P) \models (\forall o) \left( \text{root}(o, P) \supset [ (\exists o_1, ..., o_n) \text{atocc}(o_1, A_1) \land ... \land \text{atocc}(o_n, A_m) \land \text{subactivity}(A_1, P) \land ... \land \text{subactivity}(A_m, P) ] \right)
\]

Each element of a branch is an arboreal subactivity occurrence of an occurrence of \( P \), and all arboreal subactivity occurrence of an occurrence of \( P \) are elements of a branch of an activity tree:

\[
T_{psl} \cup \Sigma_{pd}(P) \models (\forall o) \left( \text{root}(o, P) \supset [ (\exists o_1, ..., o_n) \text{atocc}(o_1, A_1) \land ... \land \text{atocc}(o_n, A_m) \land \text{subactivity}(A_1, P) \land ... \land \text{subactivity}(A_m, P) ] \right)
\]
\[
(\exists o_1, \ldots, o_n) \atocc(o_1, A_1) \land \ldots \land \atocc(o_n, A_m) \land ((\forall s) \text{arboreal}(s) \supset \\text{subactivity}\_\text{occurrence}(s, o) \equiv ((s = o_1) \lor \ldots \lor (s = o_n)))]
\]

As indicated by the following theorem, the additional conditions in the definitions of strong poset activities and concurrent poset activities also impose restrictions on their process descriptions. The proof for the theorem is similar to the one for Theorem 28 and is skipped here.

**Theorem 29** If a complex activity \( P \) has finite strong poset or concurrent poset activity trees, then the ordering formula \( O(o_1, \ldots, o_n, P) \) in its process description is logically equivalent to a conjunction of \( \text{soo}\_\text{precedes} \) literals.

For example, the process description for the activity \( P_2 \) in Figure 5.4 is

\[
(\forall o) \left( \text{occurrence}\_o f(o, P_2) \supset [(\exists o_i, o_j, o_k, o_l) \text{occurrence}\_o f(o_i, \text{register}) \land \text{occurrence}\_o f(o_j, \text{hotel}) \land \text{occurrence}\_o f(o_k, \text{airplane}) \land \text{occurrence}\_o f(o_l, \text{payment}) \land \text{subactivity}(\text{register}, P_1) \land \text{subactivity}(\text{hotel}, P_1) \land \text{subactivity}(\text{airplane}, P_1) \land \text{subactivity}(\text{payment}, P_1) \land \text{soo}\_\text{precedes}(o_i, o_j, P_2) \land \text{soo}\_\text{precedes}(o_i, o_k, P_2) \land \text{soo}\_\text{precedes}(o_j, o_l, P_2) \land ((\forall s) \text{arboreal}(s) \supset \text{subactivity}\_\text{occurrence}(s, o) \equiv ((s = o_i) \lor (s = o_j) \lor (s = o_k) \lor (s = o_l))) \right)
\]

### 5.3 Complexity of Reasoning Problems

We can now consider the computational complexity of the entailment problems, under the assumptions that the process descriptions axiomatize the activity trees in the classes that we have presented above. In particular, we introduce additional assumptions to specify extensions to the PSL ontology, and then determine the complexity of the entailment problems in these extensions.

**Definition 56** The Permuted or Folded Occurrence Assumption (PFOA) is the sentence:

\[
(\forall o, a) (\text{occurrence}\_o f(o, a) \land \neg\text{atomic}(a) \supset (\text{permuted}(o) \lor \text{folded}(o)))
\]
The Strong or Concurrent Poset Assumption (SCPA)\(^5\) is the sentence:

\[
(\forall o, a) \ (\text{occurrence}_o(a, o) \land \neg \text{atomic}(a) \supset (\text{strong}_o \lor \text{concurrent}_o))
\]

It can be shown that \(T_{psl} \models SCPA \supset PFOA.\)\(^6\)

The following results show that these two assumptions are close to the boundary between tractability and intractability for the entailment problems that we have defined.

**Theorem 30** Suppose the complex activity \(P\) has only finite activity trees. Determining

\[
T_{psl} \cup \Sigma_{pd}(P) \cup PFOA \models (\forall o) \ (\text{root}(o, P) \supset (\exists o_1, o_2) \text{occurrence}_o(a, A_1) \land \text{soo}_o(a, o_1, P) \\
\land \text{occurrence}_o(a, A_2) \land \text{soo}_o(a, o_2, P) \land \text{soo}_o(o_1, o_2, P))
\]

is NP-complete.

**Proof.** By Theorem 27 and 28, there are \(n\) existentially quantified activity occurrence variables in \(\Sigma_{pd}(P)\). By PFOA, any branch of an activity tree contains at most \(n\) atomic activity occurrences, and the maximum number of branches in any activity tree is equal to the number of weak orderings on a set of \(n\) points. Thus, the problem is in NP, since by Lemma 23 we need to check whether the branch contains a subactivity occurrence of \(A_1\) that precedes a subactivity occurrence of \(A_2\).

For folded activity trees, the proof can be found in [71], which first provides a straightforward reduction from an instance \(I\) of Isat problem [1] in Interval Algebra (represented as a set of precedence and/or concurrency restrictions between endpoints of intervals of the instance) into \(f(I)\), an instance of the problem of determining the existence of a complex activity \(P\) (composed of subactivity occurrences with corresponding soo\(_o\) precedes and/or conc constraints) occurrences. A new subactivity occurrence \(o_i\) that precedes any other occurrences \(o_j\) is then added to construct a new complex activity \(P'\). It is obvious \(I\) is satisfiable iff soo\(_o\) precedes\((o_i, o_j, P')\).

For permuted activity trees, since all of the occurrence variables denote distinct subactivity occurrences and the ordering formulae in the process description is a boolean combination of soo\(_o\) precedes literals, NP-completeness follows from a straightforward reduction from 3SAT.

---

\(^5\)permuted\((o)\), folded\((o)\), strong\(_o\) poset\((o)\), and concurrent\(_o\) poset\((o)\) are the relations defined within the PSL Ontology to axiomatize the corresponding classes of activity trees.

\(^6\)http://www.mel.nist.gov/psl/ontology.html
Thus, this entailment problem (whose query was characterized in Lemma 23) is intractable even when we restrict the activities to have permuted or folded activity trees.

If we strengthen the assumption so that we consider only strong poset or concurrent poset activity trees, then we obtain an extension of the theory in which the entailment problem is tractable.

**Theorem 31** Suppose the complex activity $P$ has only finite activity trees. There exists an $O(n^2)$ algorithm to determine

\[
T_{psl} \cup \Sigma_{pd}(P) \cup SCPA \models \forall o \exists o_1, o_2 \text{occurrence}_o(o_1, A_1) \land \text{soo}_o \text{precedes}(o, o_1, P) \land \\
\text{occurrence}_o(o_2, A_2) \land \text{soo}_o \text{precedes}(o, o_2, P) \land \text{soo}_o \text{precedes}(o_1, o_2, P)
\]

where $n$ is the number of existentially quantified activity occurrence variables in $\Sigma_{pd}(P)$.

**Proof.** Suppose that $\Sigma_{pd}(P)$ contains $n$ activity occurrence variables and $m$ $\text{soo}_o \text{precedes}$ literals. By Theorem 29, we can construct a directed graph $G = \langle V, E \rangle$, where $V$ is the set of subactivity occurrence variables $occ_i$, and $(occ_i, occ_j) \in E$ iff the literal $\text{soo}_o \text{precedes}(occ_i, occ_j, P)$ is in $\Sigma_{pd}(P)$. $\Sigma_{pd}(P)$ is consistent (and hence there exists an occurrence of $P$) iff there exists a linear ordering on the vertices in $V$. This can be found using a topological sort algorithm, whose complexity is $O(n + m)$, where the upper bound for $m$ is $n(n - 1)/2$ (for a complete graph). Now, let $\Phi$ be the existential conjunction that is the consequent of the query. We can define another process description

\[
\Sigma_{pd}(P') = \Sigma_{pd}(P) \land \Phi
\]

Similarly, we know that checking the consistency of $\Sigma_{pd}(P')$ (which is equivalent to the existence of $P'$) can also be solved in $O(n^2)$ time (as $n$ stays unchanged). And it is straightforward to see that the existence of occurrence of $P'$ implies that it is possible that there exists a subactivity occurrence of $A_1$ before a subactivity occurrence of $A_2$ in an activity tree for $P$.

Note that the algorithm requires a process description with a fixed set of subactivity occurrence variables and ordering constraints that are equivalent to a conjunction of $\text{soo}_o \text{precedes}$ literals. Although the first condition is satisfied by all permuted and folded activities, only strong poset and concurrent poset activities have process descriptions that satisfy the second condition.
We can also consider the query that we characterized in Lemma 24:

**Theorem 32** Suppose the complex activity $P$ has only finite activity trees. Determining

$$T_{psl} \cup \Sigma_{pd}(P) \cup PFOA \models$$

$$(\forall o, o_1, o_2) \left( \text{root}(o, P) \land \text{occurrence}_o(f(o_1, A_1) \land \text{occurrence}_o(f(o_2, A_2) \land \text{soo}_\text{precedes}(o, o_1, P) \land \text{soo}_\text{precedes}(o, o_2, P) \supset \neg \text{soo}_\text{precedes}(o_2, o_1, P)) \right)$$

is NP-complete.

**Proof.** By Lemma 24, the sentence is logically equivalent to

$$(\forall o) \left( \text{root}(o, P) \supset \neg (\exists o_1, o_2) \text{occurrence}_o(f(o_1, A_1) \land \text{soo}_\text{precedes}(o, o_1, P) \land \text{occurrence}_o(f(o_2, A_2) \land \text{soo}_\text{precedes}(o, o_2, P) \land \text{soo}_\text{precedes}(o_1, o_2, P)) \right)$$

Since the activity trees for $P$ are either permuted or folded, we know that the same set of activities occur on every branch, so that the above sentence becomes

$$(\forall o) \left( \text{root}(o, P) \supset (\exists o_1, o_2) \text{occurrence}_o(f(o_1, A_1) \land \text{soo}_\text{precedes}(o, o_1, P) \land \text{occurrence}_o(f(o_2, A_2) \land \text{soo}_\text{precedes}(o, o_2, P) \land \text{soo}_\text{precedes}(o_1, o_2, P)) \right)$$

which is equivalent to the sentence in Theorem 30. □

Once again, if we use the Strong or Concurrent Poset Assumption, then we have an extension of the PSL Ontology in which the entailment problem is tractable.

**Theorem 33** Suppose the complex activity $P$ has only finite activity trees. There exists an $O(n^2)$ algorithm to determine

$$T_{psl} \cup \Sigma_{pd}(P) \cup SCPA \models$$

$$(\forall o, o_1, o_2) \left( \text{root}(o, P) \land \text{occurrence}_o(f(o_1, A_1) \land \text{occurrence}_o(f(o_2, A_2) \land \text{soo}_\text{precedes}(o, o_1, P) \land \text{soo}_\text{precedes}(o, o_2, P) \supset \neg \text{soo}_\text{precedes}(o_2, o_1, P)) \right)$$

where $n$ is the number of existentially quantified activity occurrence variables in $\Sigma_{pd}(P)$.

**Proof.** We can use the algorithm from the proof of Theorem 31 to determine whether there exists branch containing a subactivity occurrence of $A_1$ before a subactivity occurrence of $A_2$ in an activity tree for $P$ and a branch containing a subactivity occurrence
of $A_2$ before a subactivity occurrence of $A_1$ in an activity tree for $P$. If one of these branches does not exist, then the ordering satisfied by the other branch is satisfied on all branches.

The complexity of the entailment problems characterized in Lemmas 25 and 26 is still open.

### 5.4 Summary

We have shown how the PSL Ontology can be used to define the antecedents and consequents for first-order entailment problems related to the partial ordering of subactivity occurrences in occurrences of complex activities. The model theory of the PSL Ontology also allows us to prove the correctness of the formulations of the queries, as well as the correctness of the process descriptions for the classes of activities used within this chapter.

It is difficult to define these entailment problems using other process ontologies. Ontologies such as [77] lack a model theory. The ontologies in [27], [49], and [62] lack axiomatizations in their respective languages, so that we cannot formalize the queries as entailment problems. Ontologies such as [56] and [41] provide axiomatizations, but they lack an explicit and complete characterization of the models of the axiomatizations. These approaches also fail to make the distinction between the axioms in ontology and the classes of sentences in the process descriptions. As a result, it is difficult to define classes of activities such as permuted and strong poset, and we are unable to prove the correctness of the process descriptions. Finally, approaches such as [41] are unable to quantify over complex activities and their occurrences, which is required by the entailment problems that we considered. A variant to Golog called ConGolog,\(^7\) which supports concurrency, is proposed in [28]. It is notable that in [28], an encoding of programs as first-order terms is proposed and hence quantification over programs is enabled. The PSL approach adopted in this chapter is different from the ones employing ConGolog programs in the sense that

1. PSL theories are in first-order logic, hence, first-order theorem proving techniques within the fields of automated theorem proving can be applied to any reasoning task associated with PSL;

\(^7\)An example application of ConGolog can be found in [21].
2. new classes of non-regressable queries are considered, rather than the classical executability test queries and temporal reasoning queries;

3. we use a first-order axiomatization of the classes of activities for which reasoning on the queries is tractable.

In addition to providing a model-theoretic characterization of the sentences in the antecedents and consequents of the entailment problems, we have also defined extensions of the PSL Ontology in which the associated entailment problems are NP-complete and stronger extensions in which the problems are tractable. This demonstrates that the PSL Ontology can not only be used to axiomatize the assumptions that guarantee tractability, but it can also be used to reason about the logical relationships among these assumptions.

There are several avenues for future work. First, we want to provide a sharper characterization of the boundary between tractable and intractable extensions of the PSL Ontology by finding the maximal classes of ordered activity trees that contain the strong poset and concurrent poset activity trees and for which the entailment problems are still tractable.

Second, there are many other classes of activity trees and activities in the PSL Ontology which are independent of the folded and permuted activity trees; no work has yet been done to characterize the complexity of the entailment problems with these extensions of the PSL Ontology. This includes the entailment of ordering constraints from precondition and effect axioms.

Finally, we can apply the methodology of defining tractable extensions of the PSL Ontology to reasoning problems including temporal projection, plan verification, and plan recognition.
Chapter 6

SCOPE: A Situation Calculus
Ontology of Petri Nets

6.1 Introduction

Petri nets as a powerful modeling tool have long been used to describe complex dynamical systems [50]. Meanwhile, the simple formalism that Petri nets offers has enabled unambiguous graphical representations of Petri nets and facilitated studies on the properties of Petri nets with mathematical rigor. One recent application of Petri nets is within the context of the Semantic Web [49], where, in contrast to the current Web, the semantics of the content and capability are well-defined such that they are machines-interpretable, enabling automation of a variety of different operations currently performed by human-beings.

In practice, Petri nets have shown to be quite accessible in providing the operational semantics of certain class of Web services, i.e., Web-accessible service programs or devices. For example, in [51], the descriptive semantics of DARPA Agent Markup Language-Services (DAML-S), which is a logical language describing Web services, are first specified by situation calculus. In order to automatically describe, compose, simulate, and verify the Web service compositions, a Petri net formalism is then employed to describe the operational semantics of (DAML-S). Our thesis is that, Petri nets should have much broader applicability for the semantic Web. The rationale for this claim is that Situation Calculus, particularly its variant comprehensively studied by Reiter and et al. [57], [59], provides a simple but mathematically rigorous paradigm for axiomatization of dynamical systems, making it presumably an easy task to develop a situation-calculus-based Petri
nets action theory. As such, we propose in this chapter SCOPE, a situation-calculus-based ontology of Petri nets.\textsuperscript{1}

Ranging from business processes systems to flexible manufacturing control system, Petri-net-specified systems may involve concurrency, nondeterminism, asynchronism, and intense interactions between events and agents in these systems. Consequently, the specification, analysis and simulation of Petri-net-based systems often depend on the support of automated software tools. One central requirement of such tools is that they should be able to clearly specify the topological structures between the \textit{places} and the \textit{transitions} in a Petri net, as well as to correctly capture the change in the number of tokens at places upon actions of transition firing. In this chapter, we further provide a SCOPE-based Prolog implementation for Petri net systems which is notable for its conciseness – the domain-independent portion of the Prolog program only contains six clauses. Three clauses are used to specify the precondition requirements for a transition to fire, that is, a transition is enabled to fire iff all places that enter it contain at least one token; the other three clauses are used to specify the changes of the systems over the actions of transition firing; that is, the number of tokens at a place will never change upon an occurrence of a transition firing, unless this transition node enters, or leaves, the place.

Demonstrating the correctness of SCOPE and its Prolog implementation naturally is divided into two steps. The first step involves verifying that the intended interpretation of SCOPE, captured by the graph-theoretic definitions of Petri nets, is actually a model of $\mathcal{D}_{\text{scope}}$, the Basic Action Theory of SCOPE. In other words, we would show that SCOPE offers a correct axiomatization of Petri nets.\textsuperscript{2} The second step involves showing that we do provide a Prolog implementation of $\mathcal{D}_{\text{scope}}$ by justifying that whenever the Prolog program succeeds on a regressable sentence (which is, in essence, a sentence whose situation terms are rooted at the initial situation), the sentence is logically entailed by $\mathcal{D}_{\text{scope}}$; whereas whenever it fails on a regressable sentence, the negation of the sentence is entailed by $\mathcal{D}_{\text{scope}}$. Examples of use of SCOPE and its Prolog implementations are also given.

\textsuperscript{1}The first part of this work in this chapter is published in [72] and the second part of this work is summarized in [75].

\textsuperscript{2}As shown in the subsequent sections, to this end, only the satisfiability of SCOPE is proved.
6.2 Correctness and Limitations of the Axiomatization

In the next two sections, we first present SCOPE as the action theory $D_{\text{scope}}$, which is actually a set of axioms in the language of Situation Calculus, and we then show that $D_{\text{scope}}$ is satisfiable (Theorem 38).

Since $D_{\text{scope}}$ is a Basic Action Theory (Theorem 35), by Pirri and Reiter’s Relative Satisfiability Theorem [57, 59], to show that $D_{\text{scope}}$ is satisfiable, it suffices to show that, $D_{\text{scope}}_{\text{una}} \cup D_{\text{scope}}_{S_0}$, which is a subset of $D_{\text{scope}}$, is satisfiable. On the subset $D_{\text{scope}}_{\text{una}} \cup D_{\text{scope}}_{S_0}$, a statement much stronger than satisfiability can be made: any intended interpretation in the form of a Petri net structure $P^{\text{ini}}$ is a model of a particular instance of $D_{\text{scope}}_{\text{una}} \cup D_{\text{scope}}_{S_0}$ (Theorem 36), whereas any model of an instance of $D_{\text{scope}}_{\text{una}} \cup D_{\text{scope}}_{S_0}$ is isomorphic to particular Petri net structure (Theorem 37). Nevertheless, a proof for the complete correctness of the theory $D_{\text{scope}}$, which requires rigorous characterizations of dynamical behaviors of Petri nets, is expected to be mathematically involved and is left as future work.

The axiom in $D_{\text{scope}}_{\text{ss}}$, a subset of $D_{\text{scope}}$, characterizes all the conditions under which an action $\text{fire}$ can cause the fluent $\text{NumTkns}(p,s)$ to take on a particular value $n$ in the successor situation. In other words, $D_{\text{scope}}_{\text{ss}}$ is justified to be the Successor State Axiom for $\text{NumTkns}(p,s)$. We conjecture that $D_{\text{scope}}$ is a complete characterization of Petri nets and provide arguments in Appendix B to support this conjecture. We leave the formulation of a rigorous proof for future work. As shown in Section 6.5, $D_{\text{scope}}$ is adequate for (1) specifying important dynamical properties and major restricted classes of Petri nets;\(^3\) and (2) providing a foundation to perform deductive verification on Petri nets.

However, note that the theory $D_{\text{scope}}$ axiomatizes only the classical Petri nets, where, for example, tokens are not tagged with values and edges between place nodes and transition nodes are not associated with weights. Therefore in terms of logical expressiveness, $D_{\text{scope}}$ is limited. Extending $D_{\text{scope}}$ to further axiomatize these Petri net variants requires, in particular, a modification to the Successor State Axiom for the action $\text{fire}$. Depending

\(^3\)For each of the subclasses of Petri nets that we have considered, a convenient, $D_{\text{scope}}$-based axiomatization exists. Nevertheless, conflict-free Petri nets represent a subtle case that can be mistaken as a counter-example to this theorem. See Appendix B for an explanation of why conflict-free Petri nets are not a counter-example.
on how many new features are to be introduced, the modified Successor State Axiom can be complicated, potentially making it unrealistic to implement a real-world system that is based on the extended ontology.

### 6.3 SCOPE

The theory of SCOPE $D_{scope}$ is formally defined in Section 6.3.1. In Section 6.3.2, we show that indeed $D_{scope}$ constitutes a Basic Action Theory.

#### 6.3.1 The Ontology

Aside from situations, objects in SCOPE include transition nodes and place nodes. The only action function is $\text{fire}(t)$, meaning the firing of the transition $t$. The only functional fluent is $\text{NumTkns}(p, s)$, referring to the number of tokens at place $p$ in situation $s$. Situation-independent relations $\text{pre}(m, n)$ and $\text{post}(m, n)$ are defined in $D_{scope,S0}$ of SCOPE. More precisely, $\text{pre}(m, n)$ means node $m$ enters node $n$, whereas $\text{post}(m, n)$ means node $n$ enters node $m$. It is obvious that $\text{pre}(m, n) \equiv \text{post}(n, m)$.

Given a Petri net, the logical theory $D_{scope}$ is defined to include several sets of axioms as follows:

$$D_{scope} = D_f \cup D_{scope,ap} \cup D_{scope,ss} \cup D_{scope,una} \cup D_{scope,S0}$$

where

- $D_f$ is the set of the foundational axioms;
- $D_{scope,ap}$ (Action Precondition Axiom)

$$\forall s, p, t \left( \text{Poss}(\text{fire}(t), s) \equiv \text{pre}(p, t) \supset \text{NumTkns}(p, s) \geq 1 \right),$$

which simply states that $t$ is enabled to fire at situation $s$ iff each place that enters the transition node $t$ contains at least one token.

- $D_{scope,ss}$ (Successor State Axiom)

$$\forall s, p, a, n \left( \text{NumTkns}(p, \text{do}(a, s)) = n \equiv$$

---

4The union of the first four sets are independent to any Petri net instance, whereas the remaining set construct a theory on a particular Petri net instance.
\[ \gamma_f(p, n, a, s) \vee (\text{NumTkns}(p, s) = n \land \exists n' \gamma_f(p, n, a, s)) \]

where \( \gamma_f(p, n, a, s) \overset{\text{def}}{=} \gamma_{fe}(p, n, a, s) \vee \gamma_{fl}(p, n, a, s) \), referring to the two sets of firing actions that cause the number of tokens at place \( p \) on situation \( do(a, s) \) to be equal to \( n \):

- \( \gamma_{fe}(p, n, a, s) \overset{\text{def}}{=} (\exists t) (\text{pre}(t, p) \land \neg \text{post}(t, p) \land n = \text{NumTkns}(p, s) + 1 \land a = \text{fire}(t)) \) (at situation \( s \), the number of tokens at place \( p \) is \((n - 1)\), and transition \( t \), which enters \( p \), fires at \( s \));
- \( \gamma_{fl}(p, n, a, s) \overset{\text{def}}{=} (\exists t) (\text{pre}(t, p) \land \neg \text{post}(t, p) \land n = \text{NumTkns}(p, s) - 1 \land a = \text{fire}(t)) \) (at situation \( s \), the number of tokens at place \( p \) is \((n + 1)\), and transition \( t \), which leaves \( p \), fires at \( s \));

The Successor State Axiom as above summarizes all the conditions where the number of tokens at place \( p \) is \( n \) at situation \( do(a, s) \): \( n \) could be achieved by action \( a \) from situation \( s \), or at situation \( s \) it is the case that the number of tokens at \( p \) is already \( n \) and the action \( a \) that occurs in \( s \) will not change \( n \) to some other values.

- \( D_{\text{scope.una}} \) (Unique Names Axiom)\(^5\)
  \[ (\forall t_1, t_2) (\text{fire}(t_1) = \text{fire}(t_2) \supset (t_1 = t_2)) \]

- \( D_{\text{scope.S0}} \)
  - \( (\forall p, t) (\text{pre}(p, t) \equiv \text{post}(t, p)) \), which means that the place \( p \) is the input of the transition \( t \) if and only if \( t \) is the output of \( p \);
  - for each arc from transition \( t \) to place \( p \), define \( \text{pre}(p, t) \); similarly, define \( \text{pre}(t, p) \);
  - for each place \( p \) with initial marking \( k \), define \( \text{NumTkns}(k, S_0) = k \);

While modeling dynamical systems usually is a complicated task involving identifying complex networks of causal influences, axiomatizing Petri nets is performed directly on the abstract mathematical models and thus is free of this particular type of difficulty. In particular, the relationship between the set of effect axioms \( D_{\text{scope.eff}} \) for SCOPE and \( D_{\text{scope.ss}} \) in SCOPE can be addressed with respect to the Causal Completeness Assumption. In short, \( D_{\text{scope.eff}} \) is not part of \( D_{\text{scope}} \) but is entailed by \( D_{\text{scope}} \).

\(^5\)In general, for two distinct action \( a_1 \) and \( a_2 \), it is required in \( S_{\text{una}} \) that \( a_1(\vec{x}) \neq a_2(\vec{y}) \), but here they are not included as in SCOPE we have only one action “fire”.
Definition 57 The set of effect axioms \( D_{\text{scope, ef}} \) for SCOPE are defined as follows

\[
(\forall p, t, s) \ (\text{pre}(t, p) \land \neg \text{post}(t, p) \supset \text{NumTkns}(p, \text{do}(\text{fire}(t), s)) = \text{NumTkns}(p, s) + 1),
\]

\[
(\forall p, t, s) \ (\text{pre}(p, t) \land \neg \text{post}(p, t) \supset \text{NumTkns}(p, \text{do}(\text{fire}(t), s)) = \text{NumTkns}(p, s) - 1),
\]

which is logically equivalent to

\[
(\forall p, n, a, s) \ (\gamma_f(p, n, a, s) \supset \text{NumTkns}(p, \text{do}(a, s)) = n). \quad (6.1)
\]

Given the clear definition of the transition rules in Petri nets, we know that we can make a causal completeness assumption (claim) from Equation 6.1: that is, it includes all the conditions under which the only action in the domain \text{fire} can cause the fluent \text{NumTkns} to have a value \( n \) in the successor situation. The Explanation Closure Axiom corresponding to this claim is

\[
(\forall p, a, s) \ (\text{NumTkns}(p, \text{do}(a, s)) \neq \text{NumTkns}(p, s) \supset (\exists n)\gamma_f(p, n, a, s)) \quad (6.2)
\]

Theorem 34 Equation 6.1 and Equation 6.2 together, are logically equivalent to the only successor state axiom in \( D_{\text{scope}} \)

\[
(\forall p, a, n, s) \ (\text{NumTkns}(p, \text{do}(a, s)) = n \equiv \\
\gamma_f(p, n, a, s) \lor (\text{NumTkns}(p, s) = n \land \neg (\exists n')\gamma_f(p, n', a, s))).
\]

Proof. The general version of this logical equivalency is provided in [59] (as Proposition 3.2.6 for a relational fluent \( F \), whereas the version for a functional fluent \( f \) is discussed around page 33). The case here involves a particular functional fluent \text{NumTkns}. \( \Box \)

It is worth noting that the consistency of \( D_{\text{scope, ef}} \) for SCOPE relies on the consistency property in the following sentence is entailed by the theory \( D_{\text{scope}} \):

\[
\neg(\exists p, n, n', a, s) \ (\gamma_f(p, n, a, s) \land \gamma_f(p, n', a, s) \land n \neq n'), \quad (6.3)
\]

which in SCOPE states that the action \text{fire} cannot assign two different values to the fluent \text{NumTkns} in the successor situation.

6.3.2 A Basic Action Theory

Theorem 35 The theory \( D_{\text{scope}} \) is a Basic Action Theory.
Proof. It can be easily verified that the requirement of uniformity of formulas in the current situation \( s \) is satisfied in the precondition axioms, the successor state axioms, and the initial condition axioms of \( D_{\text{scope}} \).

The ontology contains one successor state axiom, for the only functional fluent \( NumTkns \). Now we show the function consistency property (see Definition 28), is satisfied with respect to the fluent \( NumTkns \).

The only successor state axiom in \( D_{\text{scope}_{ss}} \) can be written in the form

\[
(\forall p, n, a, s) \ (NumTkns(p, do(a, s)) = n \equiv \phi_f(p, n, a, s))
\]

where

\[
\phi_f(p, n, a, s) \overset{\text{def}}{=} \gamma_f(p, n, a, s) \lor (NumTkns(p, s) = n \land \neg(\exists n') \gamma_f(p, n', a, s)).
\]

As a consequence of Equation 6.3,

\[
D_{\text{scope}_{una}} \models (\forall p, n, n', a, s) \ (\gamma_f(p, n, a, s) \land \gamma_f(p, n', a, s) \supset n = n'),
\]

and from the definition of \( \phi_f \), we have

\[
D_{\text{scope}_{una}} \models (\forall p, n, n', a, s) \ (\phi_f(p, n, a, s) \land \phi_f(p, n', a, s) \supset n = n').
\]

Based on the definition of \( D_{\text{scope}_{S_0}} \), we have

\[
D_{\text{scope}_{S_0}} \models (\forall p)(\exists n) \ NumTkns(p, S_0) = n.
\]

From \( S_0 \), the firing of any action \( a = \text{fire}(t) \), where the transition \( t \) is enabled in \( S_0 \) (if \( t \) is not enabled, then \( do(a, S_0) \) is a physically unreachable, \textit{ghost}, situation), will lead to the subsequent situation \( do(a, S_0) \). In \( do(a, S_0) \), with respect to the successor state axiom \( D_{\text{scope}_{ss}} \) the number of tokens at a place \( p \) is unchanged, or increased by one if \( p \) is in the output set of \( t \), or decreased by one if \( p \) is in the input set of \( t \). But this means:

\[
D_{\text{scope}_{S_0}} \cup D_{\text{scope}_{ss}} \models (\forall p, a) \ (\exists n) \ \phi_f(p, n, a, S_0).
\]

Applying mathematical induction on actions and situations, it is the case:

\[
D_{\text{scope}_{S_0}} \cup D_{\text{scope}_{ss}} \models (\forall p, a, a', s) \ (\exists n)( \phi_f(p, n, a, s) \supset (\exists n')\phi_f(p, n', a, do(a', s))).
\]
By the principle of induction on situations, we have that
\[ D_{\text{scope},S_0} \cup D_{\text{scope},ss} \models (\forall p, a, s) (\exists n) \phi_f(p, n, a, s). \]

Hence,
\[ D_{\text{scope,una}} \cup D_{\text{scope},s_0} \models \]
\[ (\forall p, a, s) (\exists n)( \phi_f(p, n, a, s) \land (\forall n, n') (\phi_f(p, n, a, s) \land \phi_f(p, n', a, s) \supset n = n')) \]
that is, for each place \( p_i \), there exists a unique value of token numbers \( n_i \) at initial situation \( S_0 \), and this uniqueness is maintained at every successor situations. \( \square \)

### 6.4 Satisfiability of \( D_{\text{scope}} \)

The bipartite graph defined as follows is the building block for specifying the models of \( D_{\text{scope}} \).

**Definition 58** A bipartite graph is a tuple \( B = (\Omega_1, \Omega_2, \text{in}) \), where \( \Omega_1, \Omega_2 \) are disjoint sets such that \( \text{in} \subseteq (\Omega_1 \times \Omega_2) \). Two elements that are related by \( \text{in} \) are called incident.

**Definition 59** A Petri net structure \( P^{\text{ini}} \) is defined to include a 8-tuple in the form \( \langle \text{Ini}, P, T, F, N, PtoT, TtoP, \text{NumOfTkns} \rangle \) where

- \( \text{Ini} \) is a constant, intuitively standing for the initial situation \( S_0 \);
- \( P, T, F, N \) are disjoint sets, intuitively standing for places, transitions, fire actions, and natural numbers, respectively;
- \( \langle P, T, PtoT \rangle, \langle T, P, TtoP \rangle \), and \( \langle P, N, \text{NumOfTkns} \rangle \) are bipartite graphs;

In addition, \( P^{\text{ini}} \) contains,

- the unary function \( \text{fire} : T \rightarrow F \) is defined such that, for any \( \langle t_1, f_1 \rangle \) and \( \langle t_2, f_2 \rangle \) in \( \text{fire} \), \( f_1 = f_2 \) only if \( t_1 = t_2 \);
- the relation set \( \text{pre} : (P \cup T) \times (P \cup T) \) is defined such that, for any two nodes \( n_1, n_2 \in (P \cup T) \), \( \langle n_1, n_2 \rangle \in \text{pre} \) iff \( \langle n_1, n_2 \rangle \in PtoT \) or \( \langle n_1, n_2 \rangle \in TtoP \);
- the relation set \( \text{post} : (P \cup T) \times (P \cup T) \) is defined such that \( \langle n_1, n_2 \rangle \in \text{post} \) iff \( \langle n_2, n_1 \rangle \in \text{pre} \).
Theorem 36  Any structure in $P^{ini}$ is a model of the union of $D_{scope,una}$ and $D_{scope,S_0}$, where $D_{scope,S_0}$ corresponds to some particular instance of Petri net.

Proof.  Suppose we have a structure $Petri \in P^{ini}$ ($P^{ini}$ is obviously non-empty)

$$\langle Ini, P, T, F, N, PtoT, TtoP, NumOfTkns \rangle.$$ 

It is easy to construct an $D_{scope,S_0}$ such that the nodes of places, the nodes of transitions, the topological structures between places and transitions, and the initial assignments of tokens for each places in $D_{scope,S_0}$, are set with reference to $Petri$. It is easy to verify that each axiom in $D_{scope,una} \cup D_{scope,S_0}$ is satisfied by $Petri$, that is:

- $Petri \models (\forall t_1, t_2) \ (fire(t_1) = fire(t_2) \supset (t_1 = t_2))$
  
  iff for any two pairs $\langle t_1, f_1 \rangle$ and $\langle t_2, f_2 \rangle$, where $t_1, t_2 \in T$ and $f_1, f_2 \in F$,
  
  if $f_1 = f_2$ then $t_1 = t_2$.

  This holds from the definition of the function $fire$ in $P^{ini}$.

- $Petri \models pre(n_1, n_2)$ for each $pre(n_1, n_2) \in D_{scope,S_0}$

  We know that

  $Petri \models pre(n_1, n_2)$ for each $pre(n_1, n_2) \in D_{scope,S_0}$

  iff

  for any $p \in P$, $t \in T$, either $\langle p, t \rangle \in PtoT$ or $\langle t, p \rangle \in TtoP$,

  But from the way $D_{scope,S_0}$ is constructed, it is simply the case $Petri \models pre(n_1, n_2)$ for each $pre(n_1, n_2) \in D_{scope,S_0}$.

- $Petri \models (\forall p, t) \ (pre(p,t) \equiv post(t,p))$

  Since

  $Petri \models (\forall p, t) \ (pre(p,t) \equiv post(t,p))$

  iff

  for any $p \in P$, $t \in T$, $\langle p, t \rangle \in PtoT$ iff $\langle t, p \rangle \in TtoP$.  

Again from the way $D_{\text{scope},S_0}$ is constructed, we have $\text{Petri} \models (\forall p, t) \ (\text{pre}(p,t) \equiv \text{post}(t,p))$.

- $\text{Petri} \models \text{NumTkns}(p, S_0) = k$ where $\text{NumTkns}(p, S_0) = k \in D_{\text{scope},S_0}$

Similarly, from

$\text{Petri} \models \text{NumTkns}(p, S_0) = k$, where $\text{NumTkns}(p, S_0) = k \in D_{\text{scope},S_0}$

iff

for any $p \in P$, and $k \in N$, we have $\langle p, k \rangle \in \text{NumOfTkns}$.

We can conclude that $\text{Petri} \models \text{NumTkns}(p, S_0) = k$, for each place $p$.

\[ \square \]

**Theorem 37** The model $M$ of a union of $D_{\text{scope,una}}$ and $D_{\text{scope},S_0}$, where $D_{\text{scope},S_0}$ corresponds to some particular instance of Petri net, is isomorphic to a structure $\text{Petri}$ in $P^{\text{ini}}$.

**Proof.** Since $M$, the model of $D_{\text{scope,una}} \cup D_{\text{scope},S_0}$, is a model with many-sorted domains, it is easy to construct for $\text{Petri}$ the sets of $\text{Ini}$, $P$, $T$, $F$ from the disjoint elements in the domains $\text{Ini}, P, T, F$ of $M$.

Since $M \models (\forall t_1, t_2) \ (\text{fire}(t_1) = \text{fire}(t_2) \supset (t_1 = t_2))$, we can further require that the functional mappings between elements $t \in T$ and $f \in F$ satisfy the description of $\text{fire}$ in the structure $\text{Petri}$.

Similarly, since

$M \models (\forall p, t) \ (\text{pre}(p,t) \equiv \text{post}(t,p))$

and

$M \models \text{pre}(n_1, n_2) \text{ for each } \text{pre}(n_1, n_2) \in D_{\text{scope},S_0}$,

we require that the bipartite graphs $\langle P, T, \text{PtoT} \rangle$, $\langle T, P, \text{TtoP} \rangle$ in $\text{Petri}$ correspond to the pairs of $\langle p, t \rangle$ and $\langle t, p \rangle$ having the membership in the relations $\text{pre}$ and $\text{post}$ of $M$.

Finally, since

$M \models \text{NumTkns}(p, S_0) = k$ where $\text{NumTkns}(p, S_0) = k \in D_{\text{scope},S_0}$
the element \( \text{Ini} \) assigned to \( S_0 \) in the model \( M \) corresponds to \( \text{Ini} \) in \( \text{Petri} \) and this relation matches to the bipartite graph \( \langle P, N, \text{NumOfTkns} \rangle \) in \( \text{Petri} \).

Hence, we can conclude that \( M \) is isomorphic to the constructed \( \text{Petri} \), which is in membership with \( P^{\text{ini}} \).

\[ \square \]

To this end, we are able to obtain the following satisfiability theorem for \( D_{\text{scope}} \).

**Theorem 38** The theory \( D_{\text{scope}} \) is satisfiable.

**Proof.** Since \( D_{\text{scope} \cup \text{ana}} \cup D_{\text{scope} \cdot S_0} \) is satisfiable by any \( M \) corresponding to some \( P \in P^{\text{ini}} \), \( D_{\text{scope}} \) is also satisfiable, as by Pirri and Reiter’s Relative Satisfiability Theorem [57, 59], \( M \) can be extended to some \( M' \), which is a model of \( D_{\text{scope}} \).

\[ \square \]

### 6.5 Applications of SCOPE

#### 6.5.1 Axiomatization of SCOPE Subclasses

In this section, we provide an axiomatization (called \( D_{ps} \)) to the properties and subclasses of Petri nets mentioned in Section 2.4.2. With \( D_{ps} \) and \( D_{\text{scope}} \), two important tasks specified as follows can be performed: (1) to assure that the property say \( \alpha \in D_{ps} \) holds in the ontology to be developed; or (2) to check whether \( \alpha \in D_{ps} \) can be entailed from the existing ontology; that is, we can (1) check \( D \cup \alpha \) is satisfiable; or (2) check \( D \models \alpha \).

The specifications might use the following two abbreviations.\(^6\)

\[
\text{enabled}(t, s) \overset{\text{def}}{=} \text{pre}(p, t) \supset \text{NumTkns}(p, s) \geq 1
\]

\[
\text{exec}(s) \overset{\text{def}}{=} (\forall a, s^*)(\text{do}(a, s^*) \subseteq s \supset \text{Poss}(a, s^*)).
\]

Dynamical properties of Petri nets are defined as follows.

**Cycle** There exists a situation \( s \) subsequent to the initial situation \( S_0 \) such that every place contains the same number of tokens in \( s \) and in \( S_0 \)

\[
(\forall p) \ (\exists s) \ (\text{exec}(s) \land s \neq S_0 \land \text{NumTkns}(p, s) = \text{NumTkns}(p, S_0))
\]

\(^6\)The second one is introduced as Equation 4.5 in [59].
Reachability A sequence of applicable transitions firings (could be empty) will lead to certain specified marking on places $M_n$; that is, given $M_n$, we have

$$\exists s \ (exec(s) \land NumTkns(p_0, s) = n_0 \land \ldots NumTkns(p_i, s) = n_i),$$

where $n_i$ is the specified number of tokens at the place $p_i$ in the Marking $M_n$.

**K-Boundedness** The number of tokens at any place in any reachable marking will not exceed $k$

$$(\forall p, s) \ (exec(s) \supset NumTkns(p, s) \leq k)$$

**Liveness** For any transition $t$, there always exists a subsequent marking from the current marking, with which $t$ is enabled to fire.

$$(\forall s, t) \ (exec(s) \supset (\exists s_1)(s \sqsubseteq s_1 \land exec(s_1) \land Poss(\text{fire}(t), s_1)))$$

**Reversibility** From any current marking, there exists a subsequent marking that is identical to the initial marking at $S_0$.

$$(\forall s, p) \ (exec(s) \supset (\exists s_1)(s \sqsubseteq s_1 \land exec(s_1) \land NumTkns(p, s_1) = NumTkns(p, S_0))),$$

where $S_0$ is the initial situation corresponding the home marking of Petri nets.

**Coverability** ($s_1$ is covered by $s_2$) The number of tokens of any given place in the marking $s_1$ is less than or equal to its number of tokens at the marking $s_2$ Note that specifically, we can require that $s_1 = S_0$.

$$(\exists s_1, s_2, p) \ (exec(s_1) \land exec(s_2) \land NumTkns(p, s_1) \leq NumTkns(p, s_2))$$

**Persistence** If both transition $t_1$ and $t_2$ are enabled at the marking of $s$, firing of $t_2$ will not prevent the further firing of $t_1$ at the subsequent marking.

$$\neg(\exists s, t_1, t_2) \ (exec(s) \land enabled(t_1, s) \land enabled(t_2, s) \land \neg enabled(t_1, do(\text{fire}(t_2), s)))$$

Restricted classes of Petri nets are defined as follows.

**S-systems** Any transition has one input place and one output place.

$$(\forall p_1, p_2, t) \ (\text{pre}(p_1, t) \land \text{pre}(p_2, t) \supset p_1 = p_2)$$

$$(\forall p_1, p_2, t) \ (\text{post}(p_1, t) \land \text{post}(p_2, t) \supset p_1 = p_2)$$
T-systems Any place has one input transition and one output transition.

\[
(\forall t_1, t_2, p) (\text{pre}(t_1, p) \land \text{pre}(t_2, p) \supset t_1 = t_2)
\]

\[
(\forall t_1, t_2, p) (\text{post}(t_1, p) \land \text{post}(t_2, p) \supset t_1 = t_2)
\]

Conflict-Free for every place with more than one output transition (say \(t_1\) and \(t_2\)), any of its output transitions is also its input transition.

\[
(\forall p, t_3) (\exists t_1, t_2) (\text{post}(t_1, p) \land \text{post}(t_2, p) \land t_1 \neq t_2 \supset \text{post}(t_3, p) \supset \text{pre}(t_3, p))
\]

Free-Choice If the place \(p\) is the input of transition \(t\), then, all input of \(t\) is the input of all output of \(p\).

\[
(\forall p, t) (\text{pre}(p, t) \supset (\forall p_1, t_1) (\text{pre}(p_1, t) \land \text{pre}(p_1, t_1) \supset \text{pre}(p_1, t_1)))
\]

### 6.5.2 Examples

We now demonstrate the use of SCOPE by giving two examples. They are adopted from Figure 17-h of [50], which is a live, reversible, but 1-bounded Petri net instance, and Figure 17-c of [50], which is a non-live, irreversible, and 1-bounded Petri net instance. Their pictorial representations are shown in Figure 2.3 and Figure 6.1, respectively. Also, the principle of induction for executable situations as a consequence of \(\mathcal{D}_{scope}\) (see also [59]) turns out to be helpful:

\[
(\forall P)(P(S_0) \land (\forall a, s)(P(s) \land \text{exec}(s) \land \text{Poss}(a, s) \supset P(\text{do}(a, s))) \supset (\forall s).\text{exec}(s) \supset P(s)).
\]

**Example 2** \(\mathcal{D}^1_{scope} = \mathcal{D}_f \cup \mathcal{D}^1_{scope, ap} \cup \mathcal{D}^1_{scope, ss} \cup \mathcal{D}^1_{scope, unu} \cup \mathcal{D}^1_{scope, S_0}\) where

\[
\mathcal{D}^1_{scope, S_0} = \{\text{pre}(p, t) \equiv \text{post}(t, p), \text{pre}(T_1, P_1), \text{pre}(P_1, T_2), \text{pre}(T_2, P_2), \text{pre}(P_2, T_1),
\text{NumTkns}(P_1, S_0) = 1, \text{NumTkns}(P_2, S_0) = 0.\}
\]

1-boundedness: \(\mathcal{D}^1_{scope} \models (\forall p, s) (\text{exec}(s) \supset \text{NumTkns}(p, s) \leq 1)\).

**Proof.** We prove 1-boundedness by showing

\[
(\forall s) (\text{exec}(s) \supset [(\text{NumTkns}(P_1, s) = 1 \land \text{NumTkns}(P_2, s) = 0) \lor
(\text{NumTkns}(P_2, s) = 1 \land \text{NumTkns}(P_1, s) = 0)]).
\]
Figure 6.1: An examples of Petri nets: 1-bounded, non-live, and irreversible (Source: Figure 17-d of [50])

Now define

\[ \psi(s) \overset{\text{def}}{=} \left( \text{NumTkns}(P_1, s) = 1 \land \text{NumTkns}(P_2, s) = 0 \right) \lor \left( \text{NumTkns}(P_2, s) = 1 \land \text{NumTkns}(P_1, s) = 0 \right), \]

thus

\[ \psi(\text{do(fire}(t), s)) \overset{\text{def}}{=} \left( \text{NumTkns}(P_1, \text{do(fire}(t), s)) = 1 \land \text{NumTkns}(P_2, \text{do(fire}(t), s)) = 0 \right) \lor \left( \text{NumTkns}(P_2, \text{do(fire}(t), s)) = 1 \land \text{NumTkns}(P_1, \text{do(fire}(t), s)) = 0 \right). \]

The term \( \psi(S_0) \) holds trivially, as

\[ \text{NumTkns}(P_1, S_0) = 1 \land \text{NumTkns}(P_2, S_1) = 0. \]

Now assume \( \psi(s) \land \text{exec}(s) \land \text{Poss}(a, s) \) holds, we have two cases to consider

1. \( \text{NumTkns}(P_1, s) = 1 \land \text{NumTkns}(P_2, s) = 0 \)
2. \( \text{NumTkns}(P_2, s) = 1 \land \text{NumTkns}(P_1, s) = 0 \)

By using the SSA, for (1)

\[ \text{NumTkns}(P_2, \text{do(fire}(t), s)) = 1 \land \text{NumTkns}(P_1, \text{do(fire}(t), s)) = 0 \]

will be obtained, for (2)

\[ \text{NumTkns}(P_2, \text{do(fire}(t), s)) = 0 \land \text{NumTkns}(P_1, \text{do(fire}(t), s)) = 1 \]
will hold, obviously, (1) and (2) lead to the holding of $\psi(do(fire(t), s))$; that is,

$$(\forall a, s)[\psi(s) \land \text{exec}(s) \land \text{Poss}(a, s) \supset \psi(do(a, s))].$$

By the principle of induction for executable situations, we have

$$(\forall s) (\text{exec}(s) \supset \psi(s)).$$

And finally,

$$(\forall s) (\text{exec}(s) \supset \text{NumTkns}(p, s) \leq 1).$$

□

**Liveness** From the previous example, we know that

$$(\forall s) (\text{exec}(s) \supset [(\text{NumTkns}(P_1, s) = 1 \land \text{NumTkns}(P_2, s) = 0) \lor \text{NumTkns}(P_2, s) = 1 \land \text{NumTkns}(P_1, s) = 0])].$$

That is, to prove

$$(\forall s) (\text{exec}(s) \supset (\exists s_1) (s \sqsubseteq s_1 \land \text{exec}(s_1))),$$

we need to prove the following two cases

1. $$(\forall s) (\text{exec}(s) \land (\text{NumTkns}(P_1, s) = 1 \land \text{NumTkns}(P_2, s) = 0) \supset (\exists s_1) (s \sqsubseteq s_1 \land \text{exec}(s_1)));$$
2. $$(\forall s) (\text{exec}(s) \land (\text{NumTkns}(P_1, s) = 0 \land \text{NumTkns}(P_2, s) = 1) \supset (\exists s_1) (s \sqsubseteq s_1 \land \text{exec}(s_1))).$$

For (1), we will show that $do(fire(T_2), s)$ is such kind of $s_1$; similarly, for (2), we have $do(fire(T_1), s)$.

**Reversibility** We just show that any situation is the same as $S_0$ or its subsequent situation is $S_0$.

**Example 3** $D^2_{\text{scope}} = D_f \cup D^2_{\text{scope,ap}} \cup D^2_{\text{scope,ss}} \cup D^2_{\text{scope,una}} \cup D^2_{\text{scope,S0}}$ where

$$D^2_{\text{scope,S0}} = \{ \text{pre}(p, t) \equiv \text{post}(t, p), \text{pre}(T_1, P_1), \text{pre}(P_1, T_2), \text{pre}(T_2, P_2), \text{pre}(P_2, T_1), \text{pre}(T_4, P_3), \text{pre}(P_3, T_3), \text{pre}(T_3, P_4), \text{pre}(P_4, T_4), \text{pre}(P_2, T_3), \text{NumTkns}(P_1, S_0) = 1, \text{NumTkns}(P_3, S_0) = 1, \text{NumTkns}(P_2, S_0) = 0, \text{NumTkns}(P_4, S_0) = 0. \}$$
Observe, and also it is easy to see that the theory entails that

- $T_4$ is possible to execute only if $T_3$ is executed;
- $T_1$ and $T_2$ execute in turn from initial situation $S_0$ before $T_3$ can execute;
- the number of tokens at both $P_1$ and $P_2$ will be zero after the execution of $T_3$;
- no successor situation after the execution of $T_4$ is executable.

Thus the non-liveness of $D^{2\text{scope}}$ is entailed:

$$D^{2\text{scope}} \models \neg ((\forall s, t)(\text{exec}(s) \supset (\exists s_1)( s \sqsubseteq s_1 \land \text{exec}(s_1) \land \text{Poss}(\text{fire}(t), s_1)))),$$

as, for example, there exists an executable situation $do(\text{fire}(T_3), do(\text{fire}(T_2), S_0))$ and a transition $T_1$ such that $T_1$ is not possible to fire in any executable situation subsequent to $do(\text{fire}(T_3), do(\text{fire}(T_2), S_0))$. Meanwhile, the irreversibility is implied from this non-liveness, for this example in particular.

In order to show the 1-boundedness, we can show that, after $T_3$ is executed, $T_4$ must be executed and deadlock is reached, whereas all places are bounded before $T_3$ is executed. More specifically, we can use the principle of induction for executable situations to show that at any of these executable situations, we have either

$$\text{NumTkns}(P_1, S_0) = 1 \land \text{NumTkns}(P_2, S_0) = 0,$$

or

$$\text{NumTkns}(P_1, S_0) = 0 \land \text{NumTkns}(P_2, S_0) = 1.$$

Thus the example is also 1-bounded:

$$D^{2\text{scope}} \models (\forall p, s) (\text{exec}(s) \supset \text{NumTkns}(p, s) \leq 1).$$

### 6.6 Prolog Implementation of SCOPE

This section provides a Prolog implementation of $D^{\text{scope}}$. In addition, the correctness of this implementation in querying over regressable sentences is guaranteed by offering

---

7 We use the University of Amsterdam Prolog Interpreter: SWI-Prolog (Multi-threaded, 32 bits, Version 5.6.57).

8 We use a variant of SCOPE that uses a relational fluent $\text{NumTkns}(p, n, s)$, which is slightly different from the version of SCOPE presented in the previous section, where a functional fluent $\text{NumTkns}(p, s)$ is included instead. We make this logically equivalent change to enable a Prolog implementation in this current section, as, conventionally, Prolog only provides mechanisms for relations not functions.

9 We use $\text{prePT}$ and $\text{preTP}$ in the implementation to replace $\text{pre}$ in SCOPE. The only purpose for doing this is to explicitly restrict the type of the two terms in the predicates.
the Implementation Theorem for $D_{scope}$, which is an application of the general Implementation Theorem for Basic Action Theories (Corollary 5.3.6 of [59], see also Proposition 9). An example is also given.

### 6.6.1 Implementation

First, the directive as follows informs the interpreter that the definition of the predicate $numTkns$ may change during execution:

```
% Prolog Compiler Directive
:-dynamic(numTkns/3).
```

The Precondition Axiom in $D_{scope}$ is logically equivalent to

$$(\forall s,t) (Poss(fire(t),s) \equiv \neg(\exists p) \neg(pre(p,t) \supset (\exists n) NumTkns(p,n,s) \land n \geq 1)),$$

Thus: $$(\forall s,t) (Poss(fire(t),s) \equiv \neg aux_p(t,s)),$$ where the new predicate $aux_p$ is defined by

$$aux_p(t,s) \equiv (\exists p) \neg(pre(p,t) \supset (\exists n) NumTkns(p,n,s) \land n \geq 1).$$

The if-half of these two definitions are:

$$(\forall s,t) (\neg(aux_p(t,s)) \supset poss(fire(t),s)),

(\forall s,t) (\exists p)( (\neg(pre(p,t) \supset aux_q(p,s)) \supset aux_p(t,s))).
$$

By introducing

$$aux_q(p,s) \equiv (\exists n) (NumTkns(p,n,s) \land n \geq 1)$$

the second sentence become

$$(\forall s,t) (\exists p)( (\neg(pre(p,t) \supset aux_q(p,s)) \supset aux_p(t,s))).$$

Hence, the Lloyd-Topor transformations applied to the Precondition Axiom yield the following Prolog clauses:

$$(\forall s,t) (\neg(aux_p(t,s)) \supset poss(fire(t),s)),$$

$$(\forall s,t) (\exists p) (((\neg(aux_q(p,s)) \land pre(p,t)) \supset aux_p(t,s))).$$
(∀s,t) (∃n) ((numTkns(p, n, s) ∧ n ≥ 1) ⊃ aux.q(p, s)).

Replace ¬ by Prolog not, and replace ∧ and ∨ with Prolog disjunction and conjunction operators; and ,, respectively, we obtain:

-------------------------------
% Precondition for action: fire

poss(fire(T),S):- not(aux_p(T,S)).
aux_p(T,S):- prePT(P,T), not(aux_q(P,S)).
aux_q(P,S):- numTkns(P,N,S), N >=1.
-------------------------------

The sufficient condition of the Successor State Axiom (see Footnote 8) in SCOPE is

\[ γ^+_F(p, n, a, s) \lor (\text{numTkns}(p, n, s) \land \neg γ^-_F(p, n, a, s)) \supset \text{numTkns}(p, n, do(a, s)), \]

where \( γ^+_F(p, n, a, s) \) corresponds to the Prolog part achieveN, and \( \neg γ^-_F(p, n, a, s) \) corresponds to the Prolog part falsifyN. Using similar Lloyd-Topor transformation techniques, the Successor State Axiom for the fluent numTkns is stated in Prolog as follows, where \( is \) is a special predefined operator to force mathematical operations:

-------------------------------
% Fluent numTkns in Successor State

numTkns(P,N,do(A,S)):-
   achieveN(P,N,A,S);
   numTkns(P,N,S), not(falsifyN(P,A)).

achieveN(P,N, A, S):-
   (A= fire(T), preTP(T,P), numTkns(P,N0,S), N is N0+1)
   ;
   (A= fire(T), prePT(P,T), numTkns(P,N0,S), N is N0-1).
-------------------------------

In certain Prolog compilers (not SWI-Prolog that we use, where the predicate not only allows one term), the auxiliary predicates aux.p and aux.q introduced as above can be eliminated by further Lloyd-Topor transformations.
Chapter 6. SCOPE: A Situation Calculus Ontology of Petri Nets

falsifyN(P,A) :- A=fire(T), (preTP(T,P) ; prePT(P,T)).

Since $D_{scope}$ satisfies the uniform properties (discussed in the preceding section), we can obtain the following theorem from the Implementation Theorem (Proposition 9).

**Theorem 39 (Implementation Theorem for the Theory $D_{scope}$)** Given a SCOPE theory $D_{scope}$, a Prolog program $P$ containing the sentences obtained from the above transformations, and further transformations from $D_{scope,S_0}$ in the following way

- For each definition of a non-fluent predicate of $D_{scope,S_0}$ of the form $P(\vec{x}) \equiv \Theta_P(\vec{x})$:
  
  
  \[ P(\vec{x}) \supset \Theta_P(\vec{x}); \]

- For each equivalence in $D_{scope,S_0}$ of the form $F(\vec{x},S_0) \equiv \Psi_F(\vec{x},S_0)$:

  \[ \Psi_F(\vec{x},S_0) \supset F(\vec{x},S_0). \]

The program $P$ provides a correct Prolog implementation of the Basic Action Theory $D_{scope}$ in the sense that whenever Prolog succeeds on a regressable sentence, that sentence is entailed by $D_{scope}$, and whenever it fails on such a sentence, $D_{scope}$ entails the negation of this sentence.

### 6.6.2 An Example

Given a Petri net and its initial settings, a sequence of token firing is related to two major reasoning tasks on Petri nets:

- (Executability test) whether the sequence is executable;
- (Projection) what would be the setting of the Petri net resulted from the firing sequence, if it is actually executable.

We will use an example (adopted from Figure 17-c of [47], depicted in Figure 6.1) to demonstrate that our implementation does the right thing. The initial setting $D_{scope,S_0}$ is specified as follows:

\[
preTP(t,p) \equiv (t = T_1 \land p = P_1) \lor (t = T_2 \land p = P_2) \lor (t = T_3 \land p = P_4) \lor (t = T_4 \land p = P_3),
\]

\[
prePT(p,t) \equiv
\]

\[
\]
\[(p = P_1 \land t = T_2) \lor (p = P_2 \land t = T_1) \lor (p = P_3 \land t = T_3) \lor (p = P_4 \land t = T_4) \lor (p = P_4 \land t = T_3),\]

and

\[\text{numTkns}(p, n, S_0) \equiv (p = P_1 \land n = 1) \lor (p = P_2 \land n = 0) \lor (p = P_3 \land n = 1) \lor (p = P_4 \land n = 0).\]

After some simplifications, the sufficient conditions in these sentences can be expressed in the following Prolog clauses (note that these clauses are obtained from the types of transformation stated in the Theorem 39).

---

% Initial states

\text{numTkns}(p_1, 1, s_0). \text{numTkns}(p_2, 0, s_0).
\text{numTkns}(p_3, 1, s_0). \text{numTkns}(p_4, 0, s_0).

\text{preTP}(t_1, p_1). \text{preTP}(t_2, p_2). \text{prePT}(p_1, t_2). \text{prePT}(p_2, t_1).
\text{prePT}(p_2, t_3).
\text{preTP}(t_3, p_4). \text{preTP}(t_4, p_3). \text{prePT}(p_4, t_4). \text{prePT}(p_3, t_3).

---

Results from the following two queries indicate that it is possible to fire \(t_2\) from the initial setting, and after that, \(t_1\) can be fired, but not \(t_2\) again:^{11}

\[?\neg \text{poss(fire(t2),do(fire(t2),s0))}.\]
\text{fail}
\[?\neg \text{poss(fire(t1),do(fire(t2),s0))}.\]
\text{true.}

If the firing sequence: fire \(t_2\), fire \(t_3\), and then fire \(t_4\), is executable, what would be the result of this sequence?

\[?\neg \text{poss(fire(t2),s0),}
\text{poss(fire(t3),do(fire(t2),s0))},
\text{poss(fire(t4),do(fire(t3), do(fire(t2),s0))),}
\text{numTkns(P,Num,do(fire(t4), do(fire(t3),do(fire(t2),s0))))).}\]

^{11} “?\neg” is Prolog prompt for queries.
\[ \begin{align*}
P &= p3, \text{ Num } = 1 ; \\
P &= p4, \text{ Num } = 0 ; \\
P &= p2, \text{ Num } = 0 ; \\
P &= p1, \text{ Num } = 0 ;
\end{align*}\]

fail.

It is shown as above that after the sequence, only the place \( p_3 \) maintains a token. It can be further verified that, after the sequence, no transition is able to fire.

### 6.7 Summary

In this chapter, we provide a formal (mostly in first-order logic) ontology called SCOPE for Petri nets. The design of SCOPE makes the full use of the strong correspondence between the situation \( s \) in situation calculus, which is a sequence of actions starting from the initial state \( S_0 \), and the marking \( M \) in Petri nets, which results from a sequence of transition firings starting from the initial marking \( M_0 \). Consequentially, SCOPE inherits completely the simplicity of Petri nets and their powerful expressive capability.

We define SCOPE as a Basic Action Theory of situation calculus (\( D_{\text{scope}} \)) such that we can take advantage of the Relative Satisfiability Theorem for Basic Action Theories and show in an easy and straightforward manner that \( D_{\text{scope}} \) is satisfiable. These efforts endorse the qualification of SCOPE, to be an operational language describing Web service in semantic Web, or other use cases where unambiguous interpretations of operational semantics are strictly required.

Aside from the satisfiability of the theory \( D_{\text{scope}} \), we introduce a combinatorial structure and uses it to fully characterize some core subset axioms \( (D_{\text{scope,una}} \cup D_{\text{scope,S0}}) \) of \( D_{\text{scope}} \) by providing the axiomatizability theorem and the satisfiability theorem for \( D_{\text{scope,una}} \cup D_{\text{scope,S0}} \).\textsuperscript{12} Note that we often call \( D_{\text{scope,S0}} \) a domain theory, since, different from other sets in the ontology, a particular \( D_{\text{scope,S0}} \) is dependent on a particular instance of Petri nets. It is somewhat surprising how many reasoning tasks this small ontology can support. While it is the case that this benefit is largely empowered by applying second-order based foundational axioms, in reality, any concrete reasoning task can always use a replacement of a set of first-order axioms.

\textsuperscript{12}We are considering characterizing \( D_{\text{scope}} \) completely.
Major related work includes [66] and [26]. In [26], the semantics are first described by UML classes models, and then redirected into Protege, which is a logic-based ontology development tool, and finally presented in Resource Description Framework and Web Ontology Language (OWL) formalisms. Given that the semantics of UML classes models has yet to be formally specified, model-theoretic results on this complicated semantic mapping process are not given in [26] and are intrinsically difficult to obtain indeed. Similarly, the Generic Process Model (GPM) semantics for Petri nets, proposed in [66], do not include any arguments on the issues of the correctness of the semantics.

In [51], the capabilities of Web services are defined in Situation Calculus, which is further encoded in a Petri net formalism. Hence, Web service simulation, verification and composition decision procedures are provided in addition. Likewise, connection between Petri nets and the Fluent Calculus is investigated in [40], where a correspondence between models of FCPL theories (a fragment of the Fluent Calculus) and Petri nets is established such that, many problems concerning FCPL theories can be reduced to problems of the well developed Petri net theory. It is worth noting that the role of Petri nets in [51] and [40] is to provide an execution semantics for Web service procedures or the FCPL theories, whereas here we work directly on Petri nets and provide a formal ontology of Petri nets.

Based on $\mathcal{D}_{scope}$, we move on to further provide in this chapter a Prolog implementation of $\mathcal{D}_{scope}$. We show the correctness of this implementation in evaluating regressable queries on Petri nets. Although the implementation is restricted in the sense that it is designed for correct reasoning on regressable sentences only, regression is a central computing mechanism to provide automated reasoning of important problems such as executability test and projections. While this program by itself can be used as an automated reasoning tool for Petri net applications, it also provides a logical foundations to aid formal evaluations of other reasoning software. Note that, when entailment needs induction the completeness (i.e., the guarantee that whenever $\mathcal{D}_{scope}$ entails an answer, this Prolog implementation will be able to compute the answer) can not be further obtained.

Some of the trivial but tedious steps in justifying the qualification of $\mathcal{D}_{scope}$ for the Implementation Theorem are skipped in this chapter. Moreover, certain concepts, such as the closed initial database, are fully covered in Section 2.2.3.

Earlier efforts integrating Petri nets and Prolog programming include [3], [14], [22], and [25]. In these approaches, specification, simulation, or verification of systems are first modeled by Petri nets, which are then symbolically interpreted in Prolog. More recently,
similar methodology is adopted by [17], where high level Petri nets are transformed into Prolog for model checking. All of these approaches, however, take Prolog simply as an implementation environment and do not provide any justification on the correctness of their transformation from Petri nets and Prolog. During the preliminary stages, these pragmatic strategies certainly can bring insight into the relationship between these two formalisms. Further research, however, demands the establishment of theoretical foundation on the unification of these two formalisms and an extensive analysis of the concepts underlying the formalisms is necessary.
Chapter 7

UML Activity Diagrams

7.1 Introduction

Amongst all modeling diagrams in the Unified Modeling Language (UML), activity diagrams are particularly introduced for graphical specifications of dynamical aspects of systems. To this end, for each graphical notation of activity diagrams, only a textual description is provided by the Object Management Group (OMG) to define its syntax and semantics [54]. In a sense, this informality offers a flexible platform for different parties collaborating with a system to communicate effectively via activity diagrams for the system. As a result, activity diagrams have been widely used to model work flows of business processes and software systems.

Practically, activity diagrams are also used to model composite operations, many of which are service activities in nature. With the rapid growth of Web services, and the visualization of the Semantic Web, the necessity to enable web-accessibility and automatic processability of these diagrams becomes somewhat apparent. Composite service activities, however, usually are complicated. They might involve concurrency, nondeterminism, asynchronism, or even programming constructs, making open interpretations of activity diagram semantics insufficient in general (see Section 4.1 of [63] for several examples of diagrams that are subject to ambiguous interpretations). In order to illustrate the potential ambiguities in interpreting the execution semantics in UML activity diagrams, we give a simple example here: in Figure 7.1, we are not able to tell whether the execution sequence \( Ini \rightarrow B \rightarrow A \rightarrow C \rightarrow D \rightarrow Fin \) is an executable sequence, as it is not clear if buffering of tokens in the nodes Forks and Joins should be allowed.
In this chapter,\footnote{The work in this chapter is published in [73] and [76].} we define the activity diagram semantics in terms of Situation Calculus theory. More precisely, we start by providing graph-theoretic definitions for the activity diagrams and formal characterizations of activity diagram dynamics through adopting the concepts of tokens from Petri nets. Other efforts in proposing theoretical semantic foundations for activity diagrams include [7] and [67]. In particular, the line of work [67], [68] and [69] provides Petri-nets-like semantics for activity diagrams by mapping them into Petri nets. One main difference between this earlier work and the current one is that we apply the concepts of markings directly in the activity diagrams. In doing so, the dynamics of the activity diagrams can be illustrated in a clear and straightforward manner.

We move on to propose an ontology of activity diagrams called SCAD, standing for Situation-Calculus action theory for Activity Diagrams. SCAD contains a set of actions, corresponding to the firing of diagram nodes, and a set of function fluents, corresponding to the number of tokens at diagrams nodes, which are subject to change upon firing actions. In addition, we show the satisfiability of the theory of SCAD $\mathcal{D}_{\text{scad}}$, by first proving that $\mathcal{D}_{\text{scad}}$ constructs a Basic Action Theory, and then applying the Relative Satisfiability Theorem. Two examples of SCAD applications are given. First, important mathematical properties of activity diagrams are further axiomatized in $\mathcal{D}_{\text{scad}}$. Second,
Table 7.1: Four levels of UML Activities

<table>
<thead>
<tr>
<th>Levels</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fundamental</td>
<td>action nodes, object nodes</td>
</tr>
<tr>
<td>Basic</td>
<td>plus sequencing control, data flow between nodes</td>
</tr>
<tr>
<td>Intermediate</td>
<td>plus concurrent control (fork/join), decisions (branch/merge)</td>
</tr>
<tr>
<td>Complete</td>
<td>plus additional enhancement (edge weights and etc.)</td>
</tr>
</tbody>
</table>

an example diagram is presented, where the projection problem, which involves checking the executability of a sequence of firing actions in activity diagrams and the consequence of the action sequence, is investigated in particular.

One of the most essential problems associated with an activity diagram is the reachability problem: checking whether certain node can be reached from some initial settings such that all restrictions on flow of control are satisfied. Since it corresponds directly to checking the executability of the real world process that is specified visually by the diagram, dealing with the reachability problem is inevitably an important issue that has to be resolved properly to facilitate systematic design, and efficient use of activity diagrams. Consequently, a deeper understanding of the characteristics of the reachability problem is required. We approach the problem by taking a close look on the computational properties of deciding the reachability problem in two classes of activity diagrams and our results show that the problem in general is intrinsically hard to solve.

In UML, activity diagrams are divided into four levels. Starting from the simplest level, it becomes more complicated to include structured programming constructs such as loops and conditionals (see Table 7.1). At their complete level, UML activity diagrams support modeling traditional structured programming constructs such as sequences, loops and conditionals, being able to decide whether a node is reachable in these diagrams would imply being able to decide if a full programming language would terminates, thus it is clear that the reachability problem is undecidable in general. Hence, our complexity analysis is carried out at the intermediate level activity diagrams where the line of tractability boundary will be drawn. At this level, only basic control flow, such as sequencing, branching, and concurrency, is allowed.

By simulating a Nondeterministic Linear Space Automaton (NLSA) as a k-bounded (i.e., at any time, any node of the diagram can not contain more than $k$ tokens; also called as 1-safe, if $k = 1$) activity diagram, we show that the reachability problem is PSPACE-
hard for that class of activity diagrams. The concept of NLSA is covered in [24]. As a reference, one could look into [24] and [65] for the definition of general Turing machines. In addition, by transforming from the NP-complete One-in-Three 3SAT problem ([24], page 259), we show that the problem is NP-complete if certain form of acyclicity is further restricted to 1-safe diagrams.

7.2 Motivations

Consider the workflow associated with a simplified version of an on-line book sale process. The process is triggered by external customer request for certain book. Upon the request, the service provider needs to first secure a copy of the book in stock. Once an order is accepted, an internal order to ship and deliver the item is initialized and processed and the customer is billed in the same time. Aside from simple sequential control, the behaviors of many real-world systems (e.g., business processes, large scale software development procedures, or industry projects), often are complicated enough to involve synchronization, concurrency, nondeterminism and other issues such as fairness or priority.

After a dynamic system is specified by an activity diagram, the developer of the diagram has to ask “Does this diagram completely and correctly characterize all essential dynamic aspects of the system?” Meanwhile, from the perspective of diagram use, one will ask “What can be done with this diagram?” In fact, analysis problems of behavioral properties in activity diagrams could well include the ones as follows:

• Is it possible for certain action to be executed repeatedly without any interruption?

• Is deadlock possible?

• Does every possible execution of an activity terminate?

• Is there any action whose occurrence will block other actions from being executed?

Aside from the above problems, one basic problem is “From some initial settings, could an action A possibly occur in the future?” or in short, “Is A reachable?”. In the example of Figure 7.2, the node Final is not reachable from Ini, as no matter which branch Ini chooses, flow of control can only arrive at two of the three branches entering the node Join, making the occurrence of Join impossible. However, Final is certainly
reachable if the node \textit{Join} is replaced with a \textit{Merge} node. The reachability problem is a fundamental one for the study and use of activity diagrams in a sense that, most of the other problems on the behavior properties (such as the ones listed above) can be decomposed into, or closely related to, the reachability problem. For example, if deadlock is ever possible to occur, it could necessarily mean that for some particular initial settings, some node can never be reached.

7.3 UML Activity Diagrams

Graph-theoretic definitions to describe UML activity diagrams are covered in Section 7.3.1 and the concept of markings to capture the dynamics of activity diagrams are introduced in Section 7.3.2.
7.3.1 Basic Definitions

The following definition specifies the structural properties of a given UML activity diagram. Definitions of behavioral properties are reserved for the subsequent section.

**Definition 60** A UML activity diagram is a pair \((N, E)\), where \(N\) is a finite set and \(E\) is a binary relation on \(N\). The elements in \(N\) are called nodes. Each node belongs to one and only one of the following types: Ini, Final, Branch, Merge, Fork, Join, or Action. The elements in \(E\) are called edges. The edge set \(E\) consists of ordered pairs of nodes. That is, an edge is a set \(\{u, v\}\), where \(u, v \in N\) and \(u \neq v\). By convention, we use the notation \((u, v)\) for an edge, rather than the set notation \(\{u, v\}\).

If \((u, v)\) is an edge in an activity diagram, we say that \((u, v)\) is incident from or leaves node \(u\) and is incident to or enters node \(v\). Given a node \(u \in N\), the set \(\cdot u = \{v | (v, u) \in E\}\) is the pre-set of \(u\), where each \(v\) is the input of \(u\), and the set \(u^* = \{v | (v, u) \in E\}\) is the post-set of \(u\), where each \(v\) is the output of \(u\). It is required that\(^2\)

\[
|\cdot u| = \begin{cases} 
0 & \text{if } u \text{ is the Ini node} \\
1 & \text{if } u \text{ is Branch, Fork, Action, or Final} \\
2 & \text{if } u \text{ is a Merge, or a Join node}
\end{cases}
\]

and

\[
|u^*| = \begin{cases} 
0 & \text{if } u \text{ is the Final node} \\
1 & \text{if } u \text{ is Merge, Join, Action or Ini} \\
2 & \text{if } u \text{ is a Branch, or a Fork node}
\end{cases}
\]

7.3.2 Markings and Reachability Problem

As mentioned in the introduction section, the concept of tokens and the firing of tokens are adopted from Petri nets, so that the dynamical information of an activity diagram system at any stage can be easily captured by the distribution of tokens among the nodes in the diagram. This arrangement has an additional advantage in that (as will be shown in the subsequent section) an ontology-based activity diagram system can be easily built-up: intended behaviors are precisely defined to facilitate efficient system specifications and testing.

\(^2\)In general, the in-degrees of Merge and Join, and the out-degrees of Branch and Fork, all can be integers greater than 2. The Join node in the Example 7.2, for example, has an in-degree of 3.
In a Petri net, nodes of places contain tokens whereas firing of a transition node make changes to the number of tokens in the places that enter, or leave the transition node. As defined below, a node in an activity diagram by itself maintains tokens and can fire as long as it contains at least one token. In addition, the left (or right) input of a Join node accepts left (or right) tokens and, intuitively, one left token and one right token will be counted as a full token for the Join node.

**Definition 61** A marking of a diagram \((N, E)\) is a mapping in the form \(MK : N \rightarrow \mathcal{N}\), to indicate the distribution of tokens on the nodes of the diagram; it can be represented as a vector \(MK(n_1), \ldots, MK(n_m)\) where \(n_1, \ldots, n_m\) is an enumeration of the node set \(N\) and for all \(i\) such that \(1 \leq i \leq m\), \(MK(n_i)\) tokens are assigned to node \(n_i\).

A node \(n\) is marked at the marking \(MK\) if \(MK(n) > 0\). A marked node \(u\) is also enabled and is accepted by every node \(v \in u^\bullet\). The firing of an enabled node \(u\) at \(MK\) leads to the successor marking \(MK'\) (Written as \(MK \xrightarrow{u} MK'\)). More precisely,

1. if \(u\) is a Branch node, then for every node \(n \in N\),

\[
MK'(n) = \begin{cases} 
MK(n) - 1 & \text{if } n = u \\
\{MK(n) + 1 \text{ or } MK(n)\} & \text{if } n \text{ accepts } u \text{ (i.e., } n \in u^\bullet) \\
MK(n) & \text{otherwise}
\end{cases}
\]

and we also have, \(\sum_{n_i \in u^\bullet} (MK'(n_i) - MK(n_i)) = 1\) (that is, exactly one node from \(u^\bullet\) will accept the token fired by \(u\));

2. if \(u\) is a non-Branch node, then for every node \(n \in N\),

\[
MK'(n) = \begin{cases} 
MK(n) - 1 & \text{if } n = u \\
MK(n) + 1 & \text{if } n \text{ accepts } u \\
MK(n) & \text{otherwise}
\end{cases}
\]

In other words, after the firing of \(u\), a token is removed from \(u\) and

(a) if \(u\) is of type \(Ini, Action, Merge, Join\), a token is added to the only node in the post-set of \(u\);

(b) if \(u\) is of type \(Fork\), a token is added to each node in the post-set of \(u\);

(c) if \(u\) is of type \(Branch\), a token is added to one and only one node in the post-set of \(u\);
There is no need to fire a node with type $Final$.

3. (Exception of $Join$) if a token fired by $u$ is accepted by the left (right) in-edge of a $Join$ node $n$, then $MK_{\text{left}}(n)$ ($MK_{\text{right}}(n)$) is increased by 1. $MK_{\text{left}}(n) = 1$ and $MK_{\text{right}}(n) = 1$ function as one full token at $n$, i.e., $MK(n) = 1$.

The firing sequence $\sigma = n_1, \ldots, n_m$ is a sequence of nodes in $N$. For particular $\sigma$ and $MK$, $\sigma$ is legal at $MK$ if there are marking sequence $MK_0, MK_1, \ldots, MK_m$ such that $MK = MK_0$, $M_0 \xrightarrow{n_1} MK_1$, $\ldots$, $MK_{m-1} \xrightarrow{n_m} MK_m$ (written as $MK \xrightarrow{\sigma} MK_m$). Also, we write $MK \xrightarrow{*} MK'$ if there exists a firing sequence $\sigma$ such that $MK \xrightarrow{\sigma} MK'$.

Given an activity diagram instance in the form of a 3-tuple $(N, E, MK_0)$, where $(N, E)$ is the diagram and $MK_0$ is its initial marking, the reachability set contains markings such that, for any $MK'$ in the set, $MK_0 \xrightarrow{*} MK'$ in $(N, E)$. The reachability problem for an $(N, E, MK_0)$ is to decide whether some arbitrary final marking $MK_{\text{final}}$ is in the reachability set of the instance. An instance $(N, E, MK_0)$ is cyclic, or reversible, if there exists a firing sequence $\sigma$ in the reachability set such that some node, say $n_i$, appears more than once in $\sigma$. It is acyclic, or irreversible, otherwise. An instance $(N, K, MK_0)$ is $k$-bounded if the number of tokens of any node $n \in N$ at any $MK$ in the reachability set is bounded by $k$; it is 1-safe if $k = 1$. The example of Figure 7.2 is 4-bounded and acyclic.

Without loss of generality, we can assume that REACH problems are to test, in an activity diagram with an unique $Ini$ node and $Final$ node where the initial $MK_0$ contains a unique token in $Ini$, if the token can reach $Final$. Three $k$-bounded variants are defined as follows.

**Definition 62** $O\text{Safe}-\text{REACH}$ is the reachability problem for 1-safe activity diagrams.

**Definition 63** $K\text{Bounded}-\text{REACH}$ is the reachability problem for $k$-bounded activity diagrams.

**Definition 64** $A-O\text{Safe}-\text{REACH}$ is the reachability problem for acyclic, 1-safe activity diagrams.

Figure 7.2 now is a pictorial representation of an instance of $(N, E, MK_0)$, where

$$N = \{Ini, Fin, A, B, C, D, Fork_1, Fork_2, Branch, Merge, Join\},$$
\[ E = \{(Ini, Branch), (Branch, Fork_0), (Branch, Fork_1), (Fork_0, A), (Fork_0, B), (Fork_1, C), \\
(Fork_1, D), (A, Join), (B, Merge), (C, Merge), (D, Join), (M, Join), (Join, Fin)\}, \]

and

\[ \{MK_0(Ini) = 1, MK_0(n_i) = 0 \text{ for all other nodes}\}. \]

In this example, the REACH problem involves finding a firing sequence from node \( Ini \) that will deliver a token into the node \( Fin \). Note that \( Fin \) is not reachable from \( Ini \), as no matter which branch \( Ini \) chooses, flow of control can only arrive at two of the three branches entering the node \( J \), making the occurrence of \( J \) impossible. However, \( Fin \) is certainly reachable if the node \( J \) is replaced with a node of type \( Merge \).

In addition, we provide a reachable example (Figure 7.3, where initially, there exists two tokens in the initial node \( Ini \)), where all sequences that end up to the node \( Fin \) correspond to the branches, from the root node \( Ini \) to all leaf nodes \( Fins \), in the execution tree depicted in Figure 7.4. For example, one of the branches is

\[ Ini \rightarrow A \rightarrow A \rightarrow D \rightarrow C \rightarrow D \rightarrow C \rightarrow Fin, \]

and the following node firing sequence will achieve this:

\[ Ini \rightarrow Branch \rightarrow Branch \rightarrow A \rightarrow A \rightarrow Fork \rightarrow \\
Fork \rightarrow D \rightarrow C \rightarrow D \rightarrow C \rightarrow Merge \rightarrow Fin \]

That is, initially, one token is branched into \( A \), enabling \( A \) to occur. Then, another token is also branched into \( A \), making \( A \) to contain two tokens. And then \( A \) occurs, twice, moving the two tokens in \( A \) to the \( Fork \) node. Each firing of the two tokens at the \( Fork \) node will deliver one token to both \( C \) and \( D \). Now the system contains four tokens, with two in \( C \) and two in \( D \). Firing one token at \( D \) will send a left token to the \( Join \) node. The node \( C \) then fires, delivering one token to the node \( Merge \). After that, the remaining token at \( D \) also fires, delivering another left token to the \( Join \) node. After that, the remaining token at \( C \) fires to put one more token into \( Merge \). Finally \( Merge \) fires, which puts one token into \( Fin \), which finishes the process.
7.4 Correctness and Limitations of the Axiomatization

In the next two sections, we first present SCAD as an action theory $D_{scad}$ and then we show that $D_{scad}$ is satisfiable (Theorem 42).

Similar to what we have achieved in the previous chapter for Petri nets, in this current chapter, with respect to UML activity diagrams, we show that $D_{scad}$ is satisfiable through the fact that $D_{scad}$ is a Basic Action Theory (Theorem 40) and a subset of $D_{scad}$ is satisfiable (Theorem 41), but we do not have a proof for the claim that the intended interpretation for a UML activity diagram is the only model of its corresponding theory $D_{scad}$.

Also note that the theory $D_{scad}$ axiomatizes only the intermediate level UML activity diagrams and it remains to be investigated whether concise axiomatization is still possible for activity diagrams at their complete level, where enhancement such as structured programming constructs (e.g., loops, and conditionals) are included.

It needs to be emphasized that $D_{scad}$ axiomatizes a particular version of the execution semantics for UML activity diagrams, which are based on a set of graph-theoretic definitions proposed in Section 7.3.
Figure 7.4: The action sequences that reach the $Fin$ node
Chapter 7. UML Activity Diagrams

7.5 SCAD

This section introduces a Situation-Calculus-based action theory for Activity Diagrams. The theory of SCAD $\mathcal{D}_{scad}$ is formally defined in Section 7.5.1. In Section 7.5.2, we show that any SCAD theory $\mathcal{D}_{scad}$ is a Basic Action Theory.

7.5.1 The Ontology

Aside from situations and actions, objects in SCAD include diagram nodes: Ini, Final, Action, Branch, Merge, Fork, and Join. Action functions in SCAD include

- $\text{fireJ}(j)$: firing a Join node;
- $\text{fireB}_l(b)$: firing a Branch node to its left edge;
- $\text{fireB}_r(b)$: firing a Branch node to its right edge;
- $\text{fire}(p)$: firing a node other than Join and Branch.

Functional fluents includes:

- $\text{TknsJ}_l(j,s)$: the number of left tokens at a Join node $j$ in situation $s$;
- $\text{TknsJ}_r(j,s)$: the number of right tokens at a Join node $j$ in $s$;
- $\text{Tkns}(p,s)$: the number of tokens at a non-Join node $p$ in $s$.

Situation-independent relations are defined in $\mathcal{D}_{scad,S_0}$ of SCAD, which specifies the structure of an activity diagram and include:

- $\text{LpreL}(m,n)$: the left output of a (Fork or Branch) node $m$ enters the left input of a Merge or Join node $n$, $\text{LpreR}(m,n)$, $\text{RpreL}(m,n)$, and $\text{RpreR}(m,n)$ are defined in a similar way;
- $\text{Lpre}(m,n)$: the left output of a (Fork or Branch) node $m$ enters the only input of node $n$ that is not of type Merge or Join, $\text{Rpre}(m,n)$ is defined similarly;
- $\text{preL}(m,n)$: the only output of a (non-Fork and non-Branch) node $m$ enters the left input of of a type Merge or Join node $n$, $\text{preR}(m,n)$ is defined similarly;
- $\text{pre}(m,n)$: all the other cases (i.e., the only input of the node $m$ enters the only input of the node $n$), and all of the above cases;
• post\((m, n)\): \(m\) leaves \(n\) if and only if \(n\) enters \(m\).

**Definition 65**  Given an activity diagram, a logical theory \(\mathcal{D}_{\text{scad}}\) can be built accordingly as a union of the ontology for activity diagrams \(\mathcal{D}_{\text{scad ontology}}\) and a specific domain theory for the diagram \(\mathcal{S}_{\text{scad domain}}\):

\[
\mathcal{D}_{\text{scad}} = \mathcal{S}_{\text{scad ontology}} \cup \mathcal{S}_{\text{scad domain}}.
\]

The ontology \(\mathcal{D}_{\text{scad ontology}}\) consists several sets of axioms as follows:

\[
\mathcal{D}_{\text{scad ontology}} = \mathcal{D}_{f} \cup \mathcal{S}_{\text{scad ap}} \cup \mathcal{S}_{\text{scad ss}}
\]

where

• \(\mathcal{D}_{f}\) is the set of the foundational axioms;

• \(\mathcal{D}_{\text{scad ap}}\)

  \[
  (\forall s, j) (Poss(fireJ(j), s) \equiv \text{TknsJ}(j, s) \geq 1 \land \text{TknsJ}_{r}(j, s) \geq 1) \text{ (a Join node } j \text{ is enabled to fire iff both the number of left tokens and the number of right tokens are greater than, or equal to 1)};
  \]

  \[
  (\forall s, b) (Poss(fireB_l(b), s) \equiv \text{Tkns}(b, s) \geq 1) \text{ (a Branch node } b \text{ is enabled to fire to its left iff the number of tokens it contains is greater than or equals to 1)};
  \]

  \[
  (\forall s, b) (Poss(fireB_r(b), s) \equiv \text{Tkns}(b, s) \geq 1) \text{ (a Branch node } b \text{ is enabled to fire to its right iff the number of tokens it contains is greater than or equals to 1)};
  \]

  \[
  (\forall s, p) (Poss(fire(p), s) \equiv \text{Tkns}(p, s) \geq 1) \text{ (a node } p \text{ is enabled to fire iff the number of tokens it contains is greater than or equals to 1)};
  \]

• \(\mathcal{D}_{\text{scad ss}}\)

  \[
  (\forall s, p, a, n) (\text{Tkns}(p, do(a, s)) = n \equiv \gamma_{f}(p, n, a, s) \lor (\text{Tkns}(p, s) = n \land \nexists n' \gamma_{f}(p, n', a, s)))
  \]

where \(\gamma_{f}(p, n, a, s) \overset{\text{def}}{=} \gamma_{f_{l}}(p, n, a, s) \lor \gamma_{f_{r}}(p, n, a, s)\), referring to the two sets of firing actions that makes the number of tokens at the non-Join node \(p\) on
situation $do(a,s)$ to $n$, that is:

1. $\gamma_{f_1}(p,n,a,s) \overset{\text{def}}{=} (n = Tkns(p,s) - 1 \land (a = fire(p) \lor a = fire_{B_l}(p) \lor a = fire_{B_r}(p)))$ (at situation $s$, the number of tokens at the node $p$, which is of any type but $Join$, is $(n + 1)$, and $p$ fires at situation $s$, making the number of tokens it contains at the subsequent situation $do(a,s)$ decreased by 1);

2. $\gamma_{f_1}(p,n,a,s) \overset{\text{def}}{=} (\exists q)(\ pre(q,p) \land n = Tkns(p,s) - 1 \land (a = fire(q) \lor a = fire_{B_l}(q) \lor a = fire_{B_r}(q) \lor a = fire_{J}(q)))$ (at situation $s$, the number of tokens at place $p$, which is of any type but $Join$, is $(n - 1)$, and the node $q$, which is of any type and enters $p$, fires at $s$, making the number of tokens at $do(a,s)$ also to $n$);

$$\forall, s, p, a, n\ (Tkns_{J_l}(p,do(a,s)) = n \equiv$$

$$\gamma_{f}(p,n,a,s) \land (Tkns_{J_l}(p,s) = n \land \neg(\exists n') \gamma_{f_1}(p,n',a,s)))$$

where $\gamma_{f}(p,n,a,s) \overset{\text{def}}{=} \gamma_{f_{l1}}(p,n,a,s) \lor \gamma_{f_{r1}}(p,n,a,s)$, referring to the two sets of firing actions that makes the number of left tokens at the $Join$ node $p$ on situation $do(a,s)$ to $n$, that is:

1. $\gamma_{f_{l1}}(p,n,a,s) \overset{\text{def}}{=} (n = Tkns_{J_l}(p,s) - 1 \land a = fire_{J}(p))$ (at situation $s$, the number of left tokens at the node $p$, which is of type $Join$, is $(n + 1)$, and $p$ fires at situation $s$, making the number of tokens it contains at the subsequent situation $do(a,s)$ decreased by 1);

2. $\gamma_{f_{l1}}(p,n,a,s) \overset{\text{def}}{=} (\exists q)(\ pre_{L}(q,p) \land n = Tkns(p,s) + 1 \land (a = fire(q) \lor a = fire_{B_l}(q) \lor a = fire_{B_r}(q) \lor a = fire_{J}(q)))$ (at situation $s$, the number of tokens at place $p$, which is of type $Join$, is $(n - 1)$, and the node $q$, which is of any type and enters the left edge of $p$, fires at $s$, making the number of left tokens at $do(a,s)$ also to $n$);

$$\forall, s, p, a, n\ (Tkns_{J_r}(p,do(a,s)) = n \equiv$$

$$\gamma_{f}(p,n,a,s) \land (Tkns_{J_r}(p,s) = n \land \neg(\exists n') \gamma_{f}(p,n',a,s)))$$

where $\gamma_{f}(p,n,a,s) \overset{\text{def}}{=} \gamma_{f_{l1}}(p,n,a,s) \lor \gamma_{f_{r1}}(p,n,a,s)$ such that

1. $\gamma_{f_{l1}}(p,n,a,s) \overset{\text{def}}{=} (n = Tkns_{J_r}(p,s) + 1 \land a = fire_{J}(p))$, and

2. $\gamma_{f_{l1}}(p,n,a,s) \overset{\text{def}}{=} (\exists q)(\ pre_{R}(q,p) \land n = Tkns(p,s) - 1 \land (a = fire(q) \lor a = fire_{B_l}(q) \lor a = fire_{B_r}(q) \lor a = fire_{J}(q)))$\(^3\)

\(^3\)The argument for the right tokens of the $Join$ node $p$ is similar to the argument of the left tokens.
The domain theory $D_{\text{scad\_domain}}$ consists several sets of axioms as follows:

$$D_{\text{scad\_domain}} = D_{\text{scad\_una}} \cup D_{\text{scad\_S0}}$$

where

- $D_{\text{scad\_una}}$
  - $(\forall t_1, t_2) (\text{fireJ}(t_1) = \text{fireJ}(t_2) \supset (t_1 = t_2))$
  - $(\forall t_1, t_2) (\text{fireB}_l(t_1) = \text{fireB}_l(t_2) \supset (t_1 = t_2))$
  - $(\forall t_1, t_2) (\text{fireB}_r(t_1) = \text{fireB}_r(t_2) \supset (t_1 = t_2))$
  - $(\forall t_1, t_2) (\text{fire}(t_1) = \text{fire}(t_2) \supset (t_1 = t_2))$
  - $(\forall t_1, t_2) (\text{fire}(t_1) \neq \text{fireJ}(t_2))$
  - $\ldots$\(^4\)

- $D_{\text{scad\_S0}}$
  - $L_{\text{preL}}(p_1, p_2), L_{\text{pre}}(p_3, p_4), \ldots$ (i.e., the specifications of the connections between nodes in the diagram);

- On the subsumption of connection concepts:
  * $(\forall p, q) (L_{\text{preL}}(p, q) \supset L_{\text{pre}}(p, q))$ (which means that if the left output of $p$ enters the left input of $q$, then $p$ enters the left input of $q),
  * $(\forall p, q) (L_{\text{preL}}(p, q) \supset L_{\text{pre}}(p, q))$ (which means that if the left output of $p$ enters the left input of $q$, then the left output of $p$ enters $q),
  * $(\forall p, q) (L_{\text{pre}}(p, q) \supset p_{\text{pre}}(q))$, (which means that if the left output of $p$ enters $q$, then $p$ enters $q),
  * $\ldots$\(^5\);
  - $(\forall p, q) (p_{\text{pre}}(q) \equiv p_{\text{post}}(q, p))$, which means that the node $p$ enters node $q$ if and only if $q$ leaves $p$;
  - for each node $p$ with initial marking $k$, define $T_{\text{knS}}(p, S_0) = k$;

\(^4\)That is, any two distinct firing action names are two distinct actions.

\(^5\)That is, specific relations on node connections imply more general relations on node connections.
7.5.2 A Basic Action Theory

Theorem 40 The theory $\mathcal{D}_{scad}$ is a Reiter’s Basic Action Theory.

Proof. It can be easily verified that the uniform requirement on the current situation $s$ is satisfied in the precondition axioms, the successor state axioms, and the initial condition axioms of $\mathcal{D}_{scad}$.

The theory $\mathcal{D}_{scad}$ contains three successor state axioms, for the functional fluent $Tkns$, $TknsJ_l$, and $TknsJ_r$. Now we show the function consistency property as defined in Section 3 is satisfied with respect to the $Tkns$. The argument for $TknsJ_l$, and $TknsJ_r$ are similar.

The successor state axiom in $\mathcal{D}_{scad}$ for $Tkns$ can be written in the form

$$(\forall p, n, a, s) \left( \text{NumTkns}(p, do(a, s)) = n \equiv f(p, n, a, s) \right)$$

where

$$f(p, n, a, s) \overset{\text{def}}{=} g_f(p, n, a, s) \lor \left( \text{NumTkns}(p, s) = n \land \neg(\exists n') g_f(p, n', a, s) \right).$$

As a consequence of Equation 6.3,

$$\mathcal{D}_{scad,una} \models (\forall p, n, n', a, s) \left( g_f(p, n, a, s) \land g_f(p, n', a, s) \supset n = n' \right),$$

and from the definition of $f$, we have

$$\mathcal{D}_{scad,una} \models (\forall p, n, n', a, s) \left( f(p, n, a, s) \land f(p, n', a, s) \supset n = n' \right).$$

Based on the definition of $\mathcal{D}_{scad,S_0}$, we have

$$\mathcal{D}_{scad,S_0} \models (\forall p)(\exists n) \left( \text{NumTkns}(p, S_0) = n \right).$$

From $S_0$, the firing of any action $a = \text{fire}(t)$, where the node $t$ is enabled in $S_0$ (if $t$ is not enabled, then $do(a, S_0)$ is a physically unreachable, ghost, situation), will lead to the subsequent situation $do(a, S_0)$. In $do(a, S_0)$, with respect to the successor state axiom $S_{ss}$ the number of tokens at a place $p$ is unchanged, or increased by one if $p$ is in the output set of $t$, or decreased by one if $p$ is in the input set of $t$. But this means:

$$\mathcal{D}_{scad,S_0} \cup \mathcal{D}_{scad,ss} \models (\forall p, a)(\exists n) f(p, n, a, S_0).$$
Applying mathematical induction on actions and situations, it is the case:

\[ D_{scad,S_0} \cup D_{scad,ss} \models (\forall p, a, a', s)(\exists n)(\phi_f(p, n, a, s) \supset (\exists n')(\phi_f(p, n', a, do(a', s))). \]

By the principle of induction on situations, we have that

\[ D_{scad,S_0} \cup D_{scad,ss} \models (\forall p, a, s)(\exists n)\phi_f(p, n, a, s). \]

Hence,

\[ D_{scad,una} \cup D_{scad,S_0} \models (\forall p, a)(\exists n)(\phi_f(p, n, a, s) \land (\forall n, n', s) (\phi_f(p, n, a, s) \land \phi_f(p, n', a, s) \supset n = n')). \]

that is, for each place \( p_i \), there exists a unique value of token numbers \( n_i \) at initial situation \( S_0 \), and this uniqueness is maintained at every successor situations. □

### 7.6 Satisfiability of \( D_{scad} \)

In this section, we show that, given an activity diagram instance, we can build a structure from the graph-theoretic specifications of the diagram and prove that the structure can be extended to a model of the \( D_{scad} \) representation of the diagram. In other words, \( D_{scad} \) is satisfiable. We need additional steps, however, to show that \( D_{scad} \) is correct: any statement entailed by \( D_{scad} \) is asserted by the diagram.

**Definition 66 (Initial Establishment of Interpretation)** Given an activity diagram \( AD \), which is a tuple in the form \( ((N, E), MK_0) \), and its corresponding SCAD theory \( D_{scad} \), which is a union of \( D_f, D_{scad,ap}, D_{scad,ss}, D_{scad,una}, \) and \( D_{scad,S_0} \), the initial establishment of the intended interpretation of \( AD \), written as \( I \), is defined to include

- a domain \( Domain \), in the form \( \langle Nd, F_J, F_{BL}, F_{BR}, F, N, [ ] \rangle \) where \( Nd, F_J, F_{BL}, F_{BR}, F, N, \) and \( [ ] \) are disjoint and
  - \( Nd \) is the set of nodes (of the type Ini, Fin, Branch, Merge, Fork, Join, or Act), that appear in \( N \) of \( AD \);
  - \( F_J \) is the set containing actions of firing the Join nodes in \( AD \);
  - \( F_{BL} \) is the set containing actions of firing the Branch nodes to their left in \( AD \);
- $F_{BR}$ is the set containing actions of firing the Branch nodes to their right in $AD$,
- $F$ is the set of other firing actions,
- $\mathcal{N}$ is the set of natural numbers;
- $[]$ is an empty sequence.

• $\delta$, denotations of terms that appear in $\mathcal{D}_{scad\_una} \cup \mathcal{D}_{scad\_S_0}$, i.e.,
  - the symbols of typed nodes are denoted accordingly by the elements in $\mathcal{N}_{d}$;
  - number symbols are denoted accordingly by the numbers in $\mathcal{N}$;
  - the constant $S_0$ is denoted by the empty sequence $[]$.

• and $\pi$, mappings for functions and predicates in $\mathcal{S}_{una} \cup \mathcal{S}_{S_0}$ with respect to the graph-theoretic definitions of $AD$, that is,
  - for each function symbol $\text{fire}J(t)$, map the node denoted by $t$ into the corresponding element in the set $F_J$;
  - the mappings for $\text{fire}B_l(t)$, $\text{fire}B_r(t)$, and $\text{fire}(t)$ are constructed in a similar manner;
  - for predicate symbol $\text{pre}$ (similarly, its variants), $\text{pre}(p, q)$ holds iff the pair of the denotations for $p$ and $q$, written as $(p^T, q^T)$, corresponds to an edge in $E$ of $AD$;
  - $\text{post}(q, p)$ holds iff $\text{pre}(p, q)$ holds;
  - $\text{Tkns}(p, S_0) = k$ holds iff $p^T$ contains $k$ tokens initially.

The following theorem holds straightforwardly through the definition above.

**Theorem 41** The interpretation $\mathcal{I}$ is a model of $\mathcal{D}_{scad\_una} \cup \mathcal{D}_{scad\_S_0}$, i.e.,

$$\mathcal{I} \models \mathcal{D}_{scad\_una} \cup \mathcal{D}_{scad\_S_0}.$$ 

**Proof.** The domain $\text{Domain}$ in $\mathcal{I}$ is constructed in disjoint way such that all of the unique names axioms in $\mathcal{D}_{scad\_una}$ will be satisfied. The terms $\delta$ and the set for the
mappings for functions and relations \( \pi \) are defined with respect to the topological relationships between the nodes edges in the activity diagrams, hence axioms in will also be satisfied.

The major theorem on the satisfiability of \( D_{scad} \), now can be stated.

**Theorem 42** The theory \( S_{scad} \) is satisfiable.

**Proof.** Since \( D_{scad_{una}} \cup D_{scad_{Sa}} \) is satisfiable by \( I \), \( S_{scad} \) is also satisfiable, as by Pirri and Reiter’s Relative Satisfiability Theorem ([57], [59]), \( I \) can be extended to some \( I' \), which is a model of \( S_{scad} \). \( \square \)

In the following, we attempted to build the intended interpretation \( I' \) that is a model of \( S \). However, we have yet to provide a proof. To come up with such a proof, techniques used in the proof for Theorem 1 of [57] might be helpful, where it is shown that the predicates and functions holds for the action precondition axioms and successor state axioms in the initial situation. Using the induction principles, the axioms also hold for subsequent situations.

**Definition 67 (Establishment of Intended interpretation)** The establishment of the intended interpretation of \( AD \), written as \( I' \), is an extension of \( I \). It includes

- a domain Domain', which is a union of Domain and \( \text{Sit} \), the set of all finite sequences of actions in \( F_J, F_{BL}, F_{BR}, \) and \( F \);
- \( \delta' \), which is the same as \( \delta \);
- \( \pi' \) mappings for functions and predicates in \( D_{scad} \) that are not included in \( \pi \), that is
  - for each function symbol \( \text{do}(\alpha, [\alpha_1, \ldots, \alpha_n]) \), map the sequence \( [\alpha_1, \ldots, \alpha_n] \) in \( \text{Sit} \) to the sequence \( [\alpha_1, \ldots, \alpha_n, \alpha] \) in \( \text{Sit} \);
  - for predicate symbol \( \sqsubseteq, \sigma \sqsubseteq I' \sigma' \) iff the sequence \( \sigma \) is a proper initial subsequence of \( \sigma' \);

### 7.7 Applications

In Section 7.7.1, a few subclasses of \( D_{scad} \) are defined and we show that the reachability problems in a particular subclass is PSPACE-complete. An abbreviation as follows is
used for the specification.\footnote{The abbreviation is first introduced as Equation 4.5 in [59].}

\[\text{exec}(s) \overset{\text{def}}{=} (\forall a, s^*)(do(a, s^*) \sqsubseteq s \supset Poss(a, s^*)).\]

A SCAD instance as depicted in Figure 7.5 is also introduced in Section 7.7.1. Example use of SCAD is demonstrated by several projection problems. (i.e., the problems of deciding whether a formula is true in the situation resulting from performing a sequence of ground actions.) Note that the principle of induction for executable situations as a consequence of $\mathcal{S}$ (see also [59]) is helpful for the discussion in the section

\[(\forall P)(P(S_0) \land (\forall a, s)[P(s) \land \text{exec}(s) \land Poss(a, s) \supset P(do(a, s))] \supset (\forall).\text{exec}(s) \supset P(s)).\]

the principle of induction for executable situations

Complexity results on the reachability problems in UML activity diagrams are introduced in 7.7.2.

7.7.1 Axiomatization of SCAD Subclasses

Three important dynamical properties of activity diagrams defined in Section 7.3.2 are formally characterized in SCAD as follows.

**Reversibility (To the initial state $S_0$, for any node $p_i$)** There exists a situation $s$ subsequent to the initial situation $S_0$ such that every node contains the same number of tokens in $s$ and in $S_0$

\[Q_{\text{rev}}(s, \vec{p}) \overset{\text{def}}{=} (\exists s') (\text{exec}(s) \land (S_0 \sqsubseteq s) \land (\bigwedge_{1 \leq i \leq k} \text{Tkns}(p_i, s) = \text{Tkns}(p_i, S_0))),\]

where $\vec{p} = (p_1, \ldots, p_k)$.

**Reachability (Given specified markings $M_n$)** A sequence of applicable transitions firings (could be empty) will lead to certain specified marking on places $M_n$;

\[Q_{\text{reach}}(s, \vec{p}) \overset{\text{def}}{=} (\exists s') (\text{exec}(s) \land (S_0 \sqsubseteq s) \land (\bigwedge_{1 \leq i \leq k} \text{Tkns}(p_i, s) = n_i)),\]

where $\vec{p} = (p_1, \ldots, p_k)$ and $n_i$ is the specified number of tokens at the nodes $p_i$ in the marking $M_n$.  


K-Boundedness The number of tokens at any node in any reachable marking will not exceed $k$

$$Q_{kbound}(s, p, k) \overset{def}{=} (\forall p, s, k) (\text{exec}(s) \supset \text{Tkns}(p, s) \leq k)$$

**Example 4** A SCAD instance $\mathcal{D}_{\text{scad}}^1$ is defined such that $\mathcal{D}_{\text{scad}}^1 = \mathcal{D}_f \cup \mathcal{D}_{\text{scad,ap}}^1 \cup \mathcal{D}_{\text{scad,as}}^1 \cup \mathcal{D}_{\text{scad,una}}^1 \cup \mathcal{D}_{\text{scad,S0}}^1$ where

$$\mathcal{D}_{\text{scad,S0}}^1 = \{\text{pre}(\text{Ini}, B), \text{Lpre}(B, F), \text{RpreR}(B, J), \text{RpreL}(F, J), \text{LpreL}(F, M), \text{preR}(j, M), \text{pre}(M, \text{Final}), \text{Tkns}(\text{Ini}, S_0) = 2, \text{Tkns}(B, S_0) = 0, \text{Tkns}(F, S_0) = 0, \text{Tkns}(\text{Final}, S_0) = 0, \text{Tkns}_{J_r}(J, S_0) = 0, \text{Tkns}(J, S_0) = 0, \text{Tkns}(M, S_0) = 0, \text{Tkns}(\text{Final}, S_0) = 0\}$$

Figure 7.5 is its pictorial presentation. A activity diagram reachability problem can now be stated as a SCAD entailment problem:

$$\mathcal{D}_{\text{scad}}^1 \models (\exists s) (\text{exec}(s) \land (S_0 \sqsubseteq s) \land \text{Tkns}(\text{Final}, s) \geq 1)?$$
That is, is there a sequence of executable actions such that at least one token is delivered to the Final node? Define
\[ \overrightarrow{a} = \{\text{fire}(\text{Ini}), \text{fireB}(B), \text{fire}(F), \text{Fire}(M)\}, \]
it can be seen that, let \( s_a = \text{do}(\overrightarrow{a}, S_0) \),\(^7\)
\[ D_{\text{scad}}^1 \models \text{exec}(S_a) \land (S_0 \sqsubseteq S_a) \land \text{Tkns}(\text{Final}, S_a) = 1. \]

From the fourth foundational axioms, it is obvious that \( S_0 \sqsubseteq s_a \). The executability of \( \overrightarrow{a} \) and \( \text{Tkns}(\text{Final}, S_a) = 1 \) can be verified by sequentially applying the four precondition axioms in \( D_{\text{scad,ap}}^1 \) and the three successor state axioms in \( D_{\text{scad,ss}}^1 \). For example, \( \text{Poss}(\text{fire}(\text{Ini}), S_0) \), together with \( \text{exec}(S_0) \), leads to \( \text{exec}(\text{do}(\text{fire}(\text{Ini}), S_0)) \); whereas \( \text{Tkns}(\text{Ini}, \text{do}(\text{fire}(\text{Ini}), S_0)) = 1 \) and \( \text{Tkns}(B, \text{do}(\text{fire}(\text{Ini}), S_0)) = 1 \).

Let
\[ \overrightarrow{b} = \{\text{fire}(\text{Ini}), \text{fire}(\text{Ini}), \text{fireB}(B), \text{fireB}(B), \text{fire}(F), \text{Fire}(J), \text{Fire}(M), \text{Fire}(M)\} \]
It is obvious that \( \text{Tkns}(\text{Final}, S_b) = 2 \).

The K-boundedness of \( D_{\text{scad}}^1 \) meanwhile, can be stated as a SCAD entailment problem in the form:
\[ D_{\text{scad}}^1 \models (\forall s, p) (\text{exec}(s) \supset \text{Tkns}(p, s) \leq 2) \]
The proof is based on the observation that all executable sequences have finite length and the principle of induction for executable situations is applied thereafter to each of these executable ones.

### 7.7.2 Computational Complexity

Three intractable theorems on \( k \)-bounded reachability problems are provided in this section. If cycles are allowed, a \( k \)-bounded reachability problem is PSPACE-complete, otherwise it is NP-complete.

**Theorem 43** The OSafe-REACH problem, i.e., given an \( S_{\text{scad}}' \), deciding
\[ D_{\text{scad}}' \cup Q_{\text{kbound}}(s, p, 1) \models Q_{\text{reach}}(s, p) \]
is PSPACE-complete.

\(^7s_a = \text{do}(\overrightarrow{a}, S_0) \) is an abbreviation for \( \text{do}(\text{Fire}(M), \text{do}(\text{fire}(F), \text{do}(\text{fireB}(B), \text{do}(\text{fire}(\text{Ini}), S_0)))) \).
Proof. We first prove OSafe-REACH is in PSPACE by showing that it is in NPSPACE. Suppose we have an instance of OSafe-REACH in the form of 4-tuple \((G, MK_0, Ini, Final)\), where \(G = (N, E)\) is an activity diagram, \(MK_0\) is the initial marking of \(G\), and \(Ini\) and \(Final\), respectively, are the initial and final node in \(G\). Since \(G\) is 1-safe, the maximal number of different markings of \(G\) is \(2^m\), where \(m = |N|\). Hence, if there ever exists a firing sequence \(\sigma\) such that \(MK_0 \xrightarrow{\sigma} MK'\) and \(MK'\) marks \(Final\), we must have \(|\sigma| \leq 2^m\), i.e., \(MK'\) shall be reached in \(2^m\) steps. Otherwise, some marking must be repeated in \(\sigma\) and the subsequence between any two repeated markings can be removed in \(\sigma\) to obtain a shorter \(\sigma'\) whereas \(MK_0 \xrightarrow{\sigma'} MK'\) also holds.

As such, we propose a nondeterministic algorithm that decides OSafe-REACH as follows:

On input \((G, MK_0, Final)\):

1. Nondeterministically select a firing sequence \(\sigma\), where \(|\sigma| \leq 2^m\).

2. Firing \(\sigma\) from \(MK_0\) to obtain a \(MK'\), i.e., \(MK_0 \xrightarrow{\sigma} MK'\). (Simply reject if \(\sigma\) is not completely firable from \(MK_0\).)

3. Accept if the \(Final\) node is marked in \(MK'\), reject otherwise.

The algorithm hence is bounded by nondeterministic linear space. As it precedes, a size \(m\) stack is required to maintain the current marking. But by virtue of Savitch’s theorem, OSafe-REACH is also in PSPACE.

To show that OSafe-REACH is PSPACE-hard, we will transform a PSPACE-complete Nondeterministic Linear Space Acceptance problem (see Page 175 in [24]) to OSafe-REACH. Suppose we have a linear bounded nondeterministic Turing machine \(T\) in the form \((Q, \Gamma, \Sigma, \delta, q_{ini}, Q_{accept}, Q_{reject}, \$)\), which is a particular Turing machine bounded by the size of its input string, whereas \(\{Q = q_1, q_2, \ldots, q_m\}\) is the set of states and \(q_1 = q_{ini}\) and \(q_{accept}, q_{reject} \in Q\), \(\Sigma\) is the set of input symbols, \(\Gamma = \{a_1, a_2, \ldots, a_p\}\) is the set of tape symbols, \(\Gamma \supseteq \Sigma\), and \(\$ = a_1\). Given a sentence \(w = \$w_1w_2\ldots w_n\$\) in \(\$\Sigma^*\$\), we will construct in polynomial time an instance of activity diagram \(OSafe(T) = (N, E, MK_0, Ini, Final)\) as follows:

1. \(N_{table} \subseteq N\) and \(N_{table} = \{LocationSymbol_{i,j}|0 \leq i \leq n + 1, 1 \leq j \leq p\} \cup \{LocationState_{i,j}|0 \leq i \leq n + 1, 1 \leq j \leq m\} \cup \{Ini, Final\}\), that is,
(a) The 2-dimensional table LocationSymbol marked with $w$ tokens is used to maintain the current content in the tape of $T$ and the 2-dimensional table LocationState with one token is used to indicate the current state. In other words, the automaton $T$ is in state $q_j$ scanning location $i$ with the symbol $k$ on $i$ if and only if there are one token at LocationSymbol$_{i,k}$ and LocationState$_{i,j}$, respectively.

(b) a token at node LocationSymbol$_0,1$, and at LocationSymbol$_{n+1,1}$, corresponds to the symbol “$\$" at the left end, and the right end of the input string of $T$.

(c) each $n_i \in N_{table}$ is an action node.

2. $E_{ini} \subseteq E$ and $E_{ini} = \{(Fork_{ini}, LocationSymbol_{i,j})|\text{input has symobl } j \text{ at } i\}$
   \[ \cup \{ (Fork_{ini}, LocationSymbol_{0,1}) \} \cup \{ (Fork_{ini}, LocationSymbol_{n+1,1}) \} \cup \{ (Ini, Fork_{ini}) \}, \]
   that is, a token in Ini will be forked into elements of the table, reflecting the input for $T$.

3. The initial marking $MK_0(T)$ simply assigns one token at Ini.

4. If a state is in $Q_{accept}$, then, it will enter the node Final. (See Fig. 7.6 for an example of the first four steps).

5. For each transition function $\delta_i \in \delta$ such that $\delta_i = \{(q_s \times a_t) \rightarrow (q_u \times a_v \times L), (q_s \times a_t) \rightarrow (q_w \times a_r \times R)\}$ and for each location $j$ such that $1 \leq j \leq n$, we construct
   
   (a) a subgraph $CpntLSS_{i,j,s,t} = (CpntN_i, CpntE_i)$, where $CpntN_i \subseteq N$ such that $CpntN_i = \{Fork_i, Join_i\}$ and $CpntE_i \subseteq E$ such that $E_i = \{(Fork_i, LocationState_{j,s}), (Fork_i, LocationSymbol_{j,t}), (LocationState_{j,s}, Join_i), (LocationSymbol_{j,t}, Join_i)\}$, constructs a component corresponding to: $T$ is at state $s$, reading symbol $t$ at location $j$.

   (b) The subgraphs $CpntLSS_{i,j-1,s,t}$, and $CpntLSS_{i,j+1,s,t}$, are constructed similarly.

   (c) A branch node $Branch_i \in N$ is introduced to simulate the nondeterministic transition of $\delta_i$. More specifically, $CpntLSS_{i,j,q,u,a_v}$ enters $Branch_i$ and $Branch_i$ leaves for $CpntLSS_{i,j-1,q,u,a_r}$ and $CpntLSS_{i,j+1,q,w,a_r}$ (see Fig. 7.7 for an example).
6. Transition functions dealing with boundary cases can be treated similarly as in the step above.

7. In $T$, transition function for a state $q_i$ might include reading different symbols, say $a_j$ or $a_k$, for example. Accordingly, the construction will have to deal with two components, say $C_{\text{ptLSS}}_{i,j,k,l}$ and $C_{\text{ptLSS}}_{i,j,k,m}$, which share the node $LocationState_{j,k}$. In this case, a $Merge$ node is added before the node to take a token from either component and a $Branch$ node is added after the node to return the token to the component. Since any $T$ can not in the same time at different state, or location, it is impossible for this $Merge$ or $Fork$ node to have multiple tokens thus the 1-safe property is maintained. (see Fig. 7.8 for an illustration).

8. $N$ (and $E$) does not contain any other nodes (or edges, respectively).

Since any nondeterministic move in $T$ corresponds precisely to a nondeterministic firing of a token in $OSafe$ and vice versa, it is obvious that $T$ accepts $w$ if and only if $OSafe(T)$ can reach the $Final$ node. Finally, we complete this proof by showing that the transformation is bounded by $O(n|Q|\cdot|\Gamma|^2)$. Since in the worst case, each location involves a nondeterministic transition function that deals with every symbol in $\Gamma$ for every state in $Q$ and the number of nodes created is bounded thus by $n \cdot |Q| \cdot |\Gamma| \cdot |Q| \cdot |\Gamma| \cdot 2$. □

Similarly we can prove

**Corollary 44** The $KBounded$-$\text{REACH}$ problem, i.e., given an $S'_{\text{scad}}$, deciding

$$D'_{\text{scad}} \cup Q_{\text{kbound}}(s,p,k) = Q_{\text{reach}}(s,p)$$

is $\text{PSPACE}$-complete.

**Proof.** Based on the fact that, the total number of markings in a k-bounded activity diagram is bounded by $\left[ \left( \sum_{i=1}^{k} i \right) + 1 \right]^m$, a nondeterministic algorithm can be proposed accordingly, ensuring that the problem is also in $\text{PSPACE}$. The same $\text{PSPACE}$-hard proof from Theorem 43 can be applied here as an 1-safe diagram is certainly k-bounded. □

In addition, we can obtain that

**Theorem 45** Given an $S'_{\text{scad}}$ where its nodes are free of $Branch$ and $Merge$ types, deciding

$$D'_{\text{scad}} \cup Q_{\text{kbound}}(s,p,k) = Q_{\text{reach}}(s,p)$$

is $\text{PSPACE}$-complete.
Figure 7.6: An example of an initial marking in $OSafe(T)$
Figure 7.7: An example of a nondeterministic transition function in $OSafe(T)$

Figure 7.8: An example of a $LocationState$ node corresponding to multiple Components in $OSafe(T)$
Proof. (Sketch.) The same nondeterministic algorithm for the general case appeared in [73] can still be applied here to show that the problem is in PSPACE.

To show that it is PSPACE-hard, we transform the PSPACE-complete Deterministic Linear Space Acceptance (DLSA) problem (see [24]) into the current problem. The proof is similar to the PSPACE-hard proof for Theorem 1 in [73], where a PSPACE-complete nondeterministic Linear Space Acceptance (NLSA) problem is applied. Here the transition function is easier as no Branch node is needed. In case a node belongs to multiple components, instead of using Merge, Branch nodes and poly-bounded duplicates of the nodes are introduced, and the proof then can be carried out in essentially the same way as in the proof for Theorem 1 of [73]. □

Theorem 46 The A-OSafe-REACH problem, i.e., deciding

\[ \mathcal{D}'_{\text{scad}} \cup Q_{k\text{bound}}(s, p, k) \cup \lnot Q_{\text{rev}}(s, p) \models Q_{\text{reach}}(s, p) \]

is NP-complete.

Proof. It is easy to see that A-OSafe-REACH is in NP. Since we know that, if the Final node in such an acyclic diagram is reachable, the firing sequence starting from Ini and reaching it is bounded by the size of the nodes in the diagram \(|N|\). A nondeterministic algorithm need only guess a firing sequence and check in polynomial time if it satisfies all constraints and delivers the token to the Final node.

To show that A-OSafe-REACH is NP-hard, we will transform One-In-Three 3SAT to A-OSafe-REACH. Suppose we have an arbitrary instance of One-In-Three 3SAT, where \(U = \{u_1, u_2, ..., u_n\}\) is a set of variables and \(C = \{c_1, c_2, ..., c_m\}\) such that \(|c_j| = 3\) for \(1 \leq j \leq m\) is the set of clauses. We construct an activity diagram

\[ \text{AOSafeReach}(\text{OneInThree3SAT}) = (N, E, MK_0, Ini, Final) \]

with the following steps.

1. For each variable \(u_i\) and its negation \(\bar{u}_i\) we construct an auxiliary variable component in which a Branch node called \(B_{u_i}\) enters the Action node \(u_i\) and the Action node \(\bar{u}_i\) before merging into a Merge node \(M_{u_i}\).

2. For each clause \(C_j = [p_{j1}, p_{j2}, p_{j3}]\) we construct an auxiliary clause component in which a Branch node called \(B_{C_j}\) enters three branches corresponding to actions \(p_{j1}, p_{j2}, p_{j3}\) before merging into a Merge node called \(M_{C_j}\).

3. The diagram contains a Fork node Fork and a Join node Join, where Ini enters Fork, Join enters Final. Also, Fork enters every node \(B_{u_i}\) for \(1 \leq i \leq n\) and
Figure 7.9: Polynomial time transformation from an One-In-Three 3SAT instance to an activity diagram

every node $B_{C_j}$ for $1 \leq j \leq m$, whereas every $M_{u_i}$ for $1 \leq i \leq n$ and every $M_{C_j}$ for $1 \leq j \leq m$ enters $Join$.

4. In the constructed diagram, we also need to indicate which literals occur in which clauses in the One-In-Three 3SAT instance. Assume literal $p_{ij}$ represents a negation of a variable $\bar{u}_k$ (or a variable $u_k$, but the construction is similar), we now add an extra fork node $Fork_{\bar{u}_k}$ such that $B_{u_k}$ enters $Fork_{\bar{u}_k}$ instead (not directly to $\bar{u}_k$) and $Fork_{\bar{u}_k}$ enters $\bar{u}_k$. Additionally, on the corresponding clause component where $p_{ij}$ belongs to, $Fork_{\bar{u}_k}$ and $p_{ij}$ enters a new $Join$ node $Join_{p_{ij}}$ and $Join_{p_{ij}}$ enters $M_{C_j}$.

5. The initial marking $MK($OneInThree3SAT$)$ simply assigns one token at $Ini$.

---

**Example Begin**

An illustration of this construction is given in Figure 7.9, where the One-In-Three 3SAT instance is $C = \{[a, b, c], [b, \neg c, d]\}$.

**Example End**
We now show that the node Final is reachable from Ini in the diagram AOSafeReach(OneInThree3SAT), if and only if, OneInThree3SAT, the instance of the problem is one-in-three satisfiable. Suppose first that \( t : U \rightarrow \{ T, F \} \) is a one-in-three truth assignment satisfying \( C \). Based on \( t \), we can construct a firing sequence \( \delta \) such that, 1) for each variable \( u_i \), if \( t(u_i) = True \) (or \( t(\bar{u}_i) = True \)), the node \( u_i \) (or \( \bar{u}_i \)) should be included in the first segment of \( \delta \) (relative order between \( u_i \)'s does not matter); 2) for each literal \( p_{ij} \), if \( p(i_j) = True \), the node \( p_{ij} \) should be included in the second segment of \( \delta \) (relative order between \( u_i \)'s does not matter). It is easy to verify that firing of \( \delta \) constructed in this way can reach Final from Ini in the constructed diagram. Conversely, if \( \delta \) is a reachable firing sequence in the diagram, for each \( i \) such that \( 1 \leq i \leq n \), \( B_{u_i} \) and \( M_{u_i} \) will ensure that only one of \( u_i \) or \( \bar{u}_i \) is chosen where as \( B_{C_i} \) and \( M_{C_i} \) will ensure that only one of \( p_{ij} \) in \( C_j \) is chosen, but this constructs a One-In-Three truth assignment \( t \) for the 3SAT problem. It is required that each of the clause component should merge to the Fork node, which means that \( t \) in the same time is actually a satisfiable assignment. The transformation can be verified as a polynomial time transformation without difficulty, since the size of constructed diagram is bounded by a polynomial \( mn \), which is the size of the 3SAT instance.

7.8 Summary

Description, analysis, or simulation of composite Web services with UML activity diagrams requires the availability of formal semantics of these diagrams. In this chapter, a graph-theoretic specification for the structural properties of activity diagrams, with adoptions of the concept of tokens in Petri nets to model the dynamics of these diagrams, is proposed.

A second-order axiomatization of this semantics of UML activity diagrams is further provided as a Basic Action Theory \( D_{scad} \). The design of the theory makes use of the strong correspondence between the situation \( s \) in situation calculus, which is a sequence of actions starting from the initial state \( S_0 \), and the marking \( M \) in activity diagrams, which is resulted from a sequence of node firings starting from the firing of the initial node at initial marking. In the previous chapter and in [72], similar methodology is taken to build SCOPE, which is a straightforward axiomatization of the dynamics of Petri nets. Major distinction between SCAD and SCOPE is that, in SCAD, a node can both store
tokens and fire them. It is observed that in [6], a mechanism for axiomatizing general flow models using the Process Specification Language, a logic language with richer process semantics, is suggested.

We have shown that the reachability problem in activity diagrams is at least PSPACE-hard. This result is significant since the problem is one of the most important problems associated with the design and use of activity diagrams. Given the popular use of the diagrams, an extensive study of the complexity of all typical problems associated with activity diagrams is obviously necessary. Although there exist extensive complexity results of reachability problems for 1-safe Petri nets (see [11] for example), those results cannot directly be used to obtain the complexity results presented here, since there does not exist a trivial way to correctly map a Petri net into an activity diagram.
Chapter 8

Conclusions

8.1 Contributions

This thesis has demonstrated powerful applications of Situation Calculus and the PSL as semantic foundations for process modeling formalisms. Event Systems, Petri nets, and UML activity diagrams are axiomatized as Basic Action Theories in Situation Calculus, whereas a complex activity with partially-ordered, non-deterministic component action occurrences is axiomatized as a PSL domain theory. More precisely,

- We have axiomatized Event Systems as logical theories in Situation Calculus, within which we defined three essential concepts that are related to a linear completion of actual events in an event system: order, admissibility, and achievability (Section 3.3);

- We have specified queries about complex activities as the consequents of first-order entailment problems from the axioms of the PSL ontology together with the process description for complex activities (Chapter 5.2);

- We provided a second-order axiomatization $D_{scope}$ of the semantics of Petri nets. The axiomatization is accurate in the sense that we prove that the intended interpretation is a model of the theory that is associated with the Petri net (Section 6.3);

- By adopting the concepts of tokens in Petri nets, we were able to provide a graph-theoretic specification of the dynamical behaviors of UML activity diagrams. These definitions share the flexibility and conciseness in specifying dynamical aspects of
domain systems as demonstrated in the use of Petri nets. This approach offers an excellent alternative to the earlier frameworks on the semantics of UML activity diagrams that have been proposed in the literature (Section 7.3). In addition, a second-order axiomatization $D_{scad}$ of the semantics of UML activity diagrams was also provided (Section 7.5).

Throughout the thesis, these ontologies were used to facilitate analytical investigation of complicated reasoning problems and the relations between these problems, to characterize subclasses of graphical models, to perform deductive reasoning on verifying properties of dynamical systems, to provide foundation for programming languages, and to support analysis of computational complexity of querying.

Through the application of Situation Calculus to Event Systems, we provided a persuasive case study that demonstrates how the introduction of formal axiomatizations can both help identify new problems in a domain and facilitate efficient analysis of the properties of these problems (Chapter 3). We considered the version of temporal projection problems where admissability of events is required. Our results show that while this version is still intractable in many restricted cases, it is simpler than the original temporal projection problems under certain settings. Given the correspondence between admissability and the coherence in plans, our results bring significant new insights into the source of computational complexity of planning as well. We demonstrated that a wide variety of different constraints – partial orderings, the number of the preconditions and effects, the number of the initial conditions, the topological structure of the cause-and-effect diagrams, and admissability – all contribute to the inherent computational intractability of temporal projection problems. For the classical inadmissible temporal projection problems, we demonstrated two distinct but also related cases where tractability is obtained. This work brings significant new insights to the development of algorithms to tackle these intractable problems (Chapter 4).

Important dynamical and structural properties of Petri nets were also axiomatized (Section 6.5.1), and the closely related concepts of reversibility, reachability, and k-boundedness were introduced to UML activity diagrams (7.7.1). We provided several examples of deductive reasoning to verify the properties of Petri nets (Section 6.5.2) and of activity diagrams (Section 7.7.1), particularly liveness, reversibility and 1-boundedness. We provided a Prolog-based interpreter for Petri nets that guarantees the preservation of the semantics from the transformation between these two formalisms. This work ex-
emphasizes the theoretical integration between Petri nets and Prolog (Section 6.6).

We defined the class of *reachability problems* in UML activity diagrams, which are of great importance in studying the behavior and structural properties of UML activity diagrams. Based on the axiomatization, we initiated the computational complexity study on the problem by providing the first set of intractable complexity results on these problems. Frequently, the proofs are enlightened by the extensive work on the complexity results of Petri nets reachability problems in the literature. On the other hand, although several efforts (see [67, 68, 69]) have been taken to transform a UML activity diagram into a Petri net diagram, the converse transformation (i.e., from a Petri net diagram to a UML activity diagram) is quite nontrivial. Hence, there does not exist a straightforward way to obtain these complexity results on the reachability problems of UML activity diagrams by one-step transformation from intractable problems in Petri nets (Section 7.7.2).

### 8.2 Future Research

One of the major limitations of our approach to axiomatizing existing formalisms is that, although we are able to show the satisfiability of $D_{es}$, $D_{scope}$ and $D_{scad}$, the full correctness of our axiomatizations of these formalisms remains to be verified: we have shown that these theories are satisfiable, but technically speaking, we cannot make a full statement that all the intended interpretations are the only models of these axiomatizations.

As such, in Situation Calculus, we are interested in looking into the categoricity of Basic Action Theories. We are going to define the conditions (such as uniformity of initial states) where the model of a Basic Action Theory is unique, i.e., the theory is categorical. In [59], only the relative satisfiability is explicitly addressed, although implicitly, the categorical property is one of the major reasons Reiter chose a second-order logic to represent the situation tree. The categoricity properties of Basic Action Theories have impact on the regular ontology engineering practice that is based on Situation Calculus in the sense that, as long as one is able to show that the intended interpretation does construct a model of the built theory, it follows that any other models of the theory are isomorphic to the intended interpretation. In other words, the axiomatization is not only sound, but also complete.

At the end of each chapter of this thesis, we have discussed the possible future work that is related to the content of the chapter. To recap, future research beyond this thesis is suggested as follows:
**Temporal Projection Problems**

- Partially ordered events are called nonlinear plans, or simply plans. All actions in a plan must be admissible, that is, have their preconditions satisfied. Partially ordered events with initial conditions are called Event Systems. Hence temporal projection problems are also called plan validation problems. Since new insights are obtained on plan validation problems, it is interesting to see what impacts these results on the admissible Possible Truth problems would have on the complexity of plan existence problems (for example, we might start by comparing and integrating our results to the results presented in [33] and [34]).

- The set of the 2-layered DAGs is a kind of highly restricted diagrams that could be used to describe relationships (see a recent example [42] on connectedness between complex regions). It might be interesting to see if our NP-completeness results on the inadmissible temporal projection problems could be extended into that domain of Region Connection Calculus (RCC) or other domains.

- Approximate algorithms for the intractable inadmissible temporal projection problems are proposed both in [12] and [53], however, neither of them are able to provide quantitative evaluations of the quality of their approximate solutions by offering upper bounds on the ratio of violated constraints in their solutions. In fact, it might be possible to investigate the hardness of approximation of these temporal projection problems by applying the Probabilistically Checkable Proofs (PCP) theorem [2, 81].

**Partially Ordered Activities**

Two of the queries that are formally specified as entailment problems in PSL are:

- Is it possible for an activity in a process plan to occur before some other activity?

- Is one activity in a process plan required to occur before some other activity?

The complexity results are obtained via defining four different classes of activity trees: Folded, Permutated, Strong Poset, and Concurrent Poset. This line of work can be extended to include formal definitions of additional queries such as:
Given the occurrence of some activity in a process plan, what activities are required to occur later?

Also, more classes of activity trees could be defined to provide a tighter tractability boundary for computation.

• Graphical Models

Given that we have built a solid correspondence between Prolog and the Petri nets, one obvious possible avenue of future research is to develop Prolog based software systems to simulate, specify, evaluate, compile, or model check Petri nets and their extended variants such as the weighted Petri nets, the colored Petri nets ([36], [37]), or the object Petri nets (which are Petri nets that use Petri nets as tokens, see also [80]).

All complexity results of the reachability problems of UML activity diagrams obtained so far are about intractability. However, there exist subclasses of activity diagrams that behave in similar ways to the conflict-free and free-choices diagrams in Petri nets [13], such that the reachability problems can be solved in polynomial time. At the same time, the complexity of other classes of problems, such as liveness, coverability, can also be investigated.

Complexity results similar to the ones in UML activity diagrams can be obtained in a variety of cross-discipline domains where work-flow representation and analysis is essential. Possible examples include:

* SAP’s WebFlow Engine [60], which is the SAP tool for managing business processes over the internet as well as internally within and between SAP components;

* workflow in the projects of Collaborative Adaptive Sensing of the Atmosphere (CASA) and Linked Environments for Atmospheric Discovery (LEAD), which are complementary projects for real-time multi-scale forecasting [58];

* ArcGIS ModelBuilder, which is a development tool for solving spatial analysis problems [55] in geography.
8.3 Conclusions

This thesis demonstrates convenient applications of Situation Calculus and the Process Specification Language (PSL) to organizing process information:

- Event Systems, Petri nets, and UML activity diagrams are axiomatized as Basic Action Theories in Situation Calculus, whereas a complex activity with partially-ordered, non-deterministic component action occurrences is axiomatized as a PSL domain theory.

- In addition, these ontologies are used
  - to facilitate analytical investigation of complicated reasoning problems and the relations between these problems;
  - to characterize subclasses of graphical models;
  - to perform deductive reasoning on verifying properties of dynamical systems, and
  - to support analysis of computational complexity of querying.
Appendix A

Process Specification Language

All materials in this appendix are adopted from the official website of PSL at:

A.1 PSL-Core

The purpose of PSL-Core\textsuperscript{1} is to axiomatize a set of intuitive semantic primitives that is adequate for describing the fundamental concepts of manufacturing processes. Consequently, this characterization of basic processes makes few assumptions about their nature beyond what is needed for describing those processes, and the Core is therefore rather weak in terms of logical expressiveness.

The basic ontological commitments of PSL-Core are based on the following intuitions:

\textbf{Intuition 1} There are four kinds of entities required for reasoning about processes – activities, activity occurrences, timepoints, and objects.

\textbf{Intuition 2} Activities may have multiple occurrences, or there may exist activities that do not occur at all.

\textbf{Intuition 3} Time points are linearly ordered, forwards into the future, and backwards into the past.

\textbf{Intuition 4} Activity occurrences and objects are associated with unique timepoints that mark the begin and end of the occurrence or object.

\textsuperscript{1}The material in this section is adopted from http://www.mel.nist.gov/psl/psl-ontology/psl_core.html.
A.1.1 Informal Semantics for PSL-Core

- (activity ?a) is TRUE in an interpretation of PSL-Core if and only if ?a is a member of the set of activities in the universe of discourse of the interpretation. Intuitively, activities can be considered to be reusable behaviors within the domain.

- (activity_occurrence ?occ) is TRUE in an interpretation of PSL-Core if and only if ?occ is a member of the set of activity occurrences in the universe of discourse of the interpretation. An activity occurrence is associated with a unique activity and begins and ends at specific points in time. Although there may exist activities that have no activity occurrence, all activity occurrences must be associated with an activity.

Activities are not classes within PSL; consequently, an activity occurrence is not an instance of an activity.

For example, the activity denoted by the term (paint House1 Paintcan1) is an instance of the class of Painting activities:

\[(\text{Painting (paint House1 Paintcan1)})\]

There may be multiple distinct occurrences of this instance:

\[(\text{occurrence_of Occ1 (paint House1 Paintcan1)})\]
\[(\text{occurrence_of Occ2 (paint House1 Paintcan1)})\]
\[(= (\text{beginof Occ1}) 1100)\]
\[(= (\text{endof Occ1}) 1200)\]
\[(= (\text{beginof Occ2}) 1500)\]
\[(= (\text{endof Occ2}) 1800)\]

There may be another instance of the class of Painting activities

\[(\text{Painting (paint House1 Paintcan2)})\]

that has no occurrences.

- (timepoint ?t) is TRUE in an interpretation of PSL-Core if and only if ?t is a member of the set of timepoints in the universe of discourse of the interpretation. Timepoints form an infinite linear ordering with endpoints at infinity.
• (object ?x) is TRUE in an interpretation of PSL-Core if and only if ?x is a member of the set of objects in the universe of discourse of the interpretation. An object is anything that is not a timepoint, nor an activity nor an activity-occurrence.

Intuitively, an object is a concrete or abstract thing that can participate in an activity. Objects can come into existence (be created) and go out of existence (be “used up” as a resource) at certain points in time. In such cases, an object has a begin and an end point. In some contexts it may be useful to consider some ordinary objects as having no such points either.

• (= ?t inf-) is TRUE in an interpretation of PSL-Core if and only if ?t is the unique timepoint that is before all other timepoints in the linear ordering over timepoints in the universe of discourse of the interpretation.

inf- plays the role of negative infinity. It is needed to specify objects that have not been created.

• (= ?t inf+) is TRUE in an interpretation of PSL-Core if and only if ?t is the unique timepoint that is after all other timepoints in the linear ordering over timepoints in the universe of discourse of the interpretation.

inf+ plays the role of positive infinity. It is needed to specify objects that are never destroyed.

• (before ?t1 ?t2) is TRUE in an interpretation of PSL-Core if and only if the timepoint ?t1 is earlier than ?t2 in the linear ordering over timepoints in the interpretation.

In PSL-Core, the set of timepoints is not dense (between any two distinct timepoints there is a third timepoint), although in PSL-Core the set of timepoints is infinite. Denseness can be added by a user as an additional postulate. Time intervals are not included among the primitives of PSL-Core; intervals can be defined with respect to timepoints and activities.

• (occurrence_of ?occ ?a) is TRUE in an interpretation of PSL-Core if and only if ?occ is a particular occurrence of the activity ?a. This is the basic relation between activities and activity occurrences. Every activity occurrence is associated with a unique activity. An activity may have no occurrences or multiple occurrences.
• \((\text{participates_in} \ ?x \ ?occ \ ?t)\) is TRUE in an interpretation of PSL-Core if and only if \(?x\) plays some role in an occurrence of the activity occurrence \(?occ\) at the timepoint \(?t\) in the interpretation. An object can participate in an activity occurrence only at those timepoints at which both the object exists and the associated activity is occurring.

• \((= \ ?t \ (\text{beginof} \ ?occ))\) is TRUE in an interpretation of PSL-Core if and only if \(?t\) is the timepoint at which the activity occurrence \(?occ\) begins.

• \((= \ ?t \ (\text{beginof} \ ?x))\) is TRUE in an interpretation of PSL-Core if and only if \(?t\) is the timepoint at which the object \(?x\) becomes possible to participate in an activity.

• \((= \ ?t \ (\text{endof} \ ?occ))\) is TRUE in an interpretation of PSL-Core if and only if \(?t\) is the timepoint at which the activity occurrence \(?occ\) ends.

• \((= \ ?t \ (\text{endof} \ ?x))\) is TRUE in an interpretation of PSL-Core if and only if \(?t\) is the timepoint at which the object \(?x\) becomes no longer possible to participate in an activity.

A.1.2 Axioms of PSL-Core

**Axiom 1** The before relation only holds between timepoints.

\[
(\forall (?t1 \ ?t2)
(if (before \ ?t1 \ ?t2)
(and (timepoint \ ?t1)
(timepoint \ ?t2)))
\]

**Axiom 2** The before relation is a total ordering.

\[
(\forall (?t1 \ ?t2)
(if (and (timepoint \ ?t1)
(timepoint \ ?t2))
(or (= \ ?t1 \ ?t2)
(before \ ?t1 \ ?t2)
(before \ ?t2 \ ?t1)))
\]

**Axiom 3** The before relation is irreflexive.
(forall (?t1)
  (not (before ?t1 ?t1)))

**Axiom 4** The before relation is transitive.

(forall (?t1 ?t2 ?t3)
  (if (and (before ?t1 ?t2)
            (before ?t2 ?t3))
    (before ?t1 ?t3)))

**Axiom 5** The timepoint inf- is before all other timepoints.

(forall (?t)
  (if (and (timepoint ?t)
            (not (= ?t inf-)))
    (before inf- ?t)))

**Axiom 6** Every other timepoint is before inf+.

(forall (?t)
  (if (and (timepoint ?t)
            (not (= ?t inf+)))
    (before ?t inf+)))

**Axiom 7** Given any timepoint t other than inf-, there is a timepoint between inf- and t.

(forall (?t)
  (if (and (timepoint ?t)
            (not (= ?t inf-)))
    (exists (?u)
      (between inf- ?u ?t))))

**Axiom 8** Given any timepoint t other than inf+, there is a timepoint between t and inf+.
(forall (?t)
  (if (and (timepoint ?t)
           (not (= ?t inf+)))
   (exists (?u)
    (between ?t ?u inf+))))

Axiom 9  Everything is either an activity, activity occurrence, timepoint, or object.

(forall (?x)
  (or (activity ?x)
       (activity_occurrence ?x)
       (timepoint ?x)
       (object ?x)))

Axiom 10  Objects, activities, activity occurrences, and timepoints are all distinct kinds of things.

(forall (?x)
  (and (if (activity ?x)
          (not (or (activity_occurrence ?x) (object ?x) (timepoint ?x))))
       (if (activity_occurrence ?x)
           (not (or (object ?x) (timepoint ?x))))
       (if (object ?x)
           (not (timepoint ?x))))

Axiom 11  The occurrence relation only holds between activities and activity occurrences.

(forall (?a ?occ)
  (if (occurrence_of ?occ ?a)
      (and (activity ?a)
           (activity_occurrence ?occ))))

Axiom 12  Every activity occurrence is the occurrence of some activity.
Appendix A. Process Specification Language

(forall (?occ)
  (if (activity_occurrence ?occ)
      (exists (?a)
        (and (activity ?a)
            (occurrence_of ?occ ?a))))))

**Axiom 13** An activity occurrence is associated with a unique activity.

(forall (?occ ?a1 ?a2)
  (if (and (occurrence_of ?occ ?a1)
           (occurrence_of ?occ ?a2))
      (= ?a1 ?a2)))

**Axiom 14** The begin and end of an activity occurrence or object are timepoints.

(forall (?a ?x)
  (if (or (occurrence_of ?x ?a)
           (object ?x))
      (and (timepoint (beginof ?x))
           (timepoint (endof ?x))))))

**Axiom 15** The begin point of every activity occurrence or object is before or equal to its end point.

(forall (?x)
  (if (or (activity_occurrence ?x)
           (object ?x))
      (beforeEq (beginof ?x) (endof ?x))))

**Axiom 16** The participates_in relation only holds between objects, activity occurrences, and timepoints, respectively.

(forall (?x ?occ ?t)
  (if (participates_in ?x ?occ ?t)
      (and (object ?x)
           (activity_occurrence ?occ)
           (timepoint ?t))))
**Axiom 17** An object can participate in an activity occurrence only at those timepoints at which both the object exists and the activity is occurring.

\[
\text{(forall } (\?x \ ?occ \ \?t) \\
\text{(if (participates_in } \?x \ ?occ \ \?t) \\
\text{(and (exists_at } \?x \ \?t) \\
\text{(is_occuring_at } ?occ \ \?t))))\]

**A.1.3 Supporting Definitions of PSL-Core**

**Definition 1** Timepoint \(\?t_2\) is between timepoints \(\?t_1\) and \(\?t_3\) if and only if \(\?t_1\) is before \(\?t_2\) and \(\?t_2\) is before \(\?t_3\).

\[
\text{(forall } (\?t_1 \ \?t_2 \ \?t_3) \ (\text{iff (between } \?t_1 \ \?t_2 \ \?t_3) \\
\text{(and (before } \?t_1 \ \?t_2) \ (before \?t_2 \ \?t_3)))})\]

**Definition 2** Timepoint \(\?t_1\) is beforeEq timepoint \(\?t_2\) if and only if \(\?t_1\) is before or equal to \(\?t_2\).

\[
\text{(forall } (\?t_1 \ \?t_2) \ (\text{iff (beforeEq } \?t_1 \ \?t_2) \\
\text{(and (timepoint } \?t_1) \ (timepoint \?t_2) \\
\text{(or (before } \?t_1 \ \?t_2) \ (= \?t_1 \ ?t_2)))})\]

**Definition 3** Timepoint \(\?t_2\) is betweenEq timepoints \(\?t_1\) and \(\?t_3\) if and only if \(\?t_1\) is before or equal to \(\?t_2\), and \(\?t_2\) is before or equal to \(\?t_3\).

\[
\text{(forall } (\?t_1 \ \?t_2 \ \?t_3) \ (\text{iff (betweenEq } \?t_1 \ \?t_2 \ \?t_3) \\
\text{(and (beforeEq } \?t_1 \ ?t_2) \ (beforeEq \?t_2 \ ?t_3))))\]

**Definition 4** An object exists at a timepoint \(\?t\) if and only if \(\?t\) is betweenEq its begin and end points.

\[
\text{(forall } (\?x \ \?t) \ (\text{iff (exists_at } \?x \ \?t) \\
\text{(and (object } \?x) \\
\text{(betweenEq (beginof } \?x) \ ?t \ (endof \?x)))})\]
Definition 5 An activity is occurring at a timepoint $t$ if and only if $t$ is betweenEq the begin and end points of the activity occurrence.

$$(\forall (\text{occ} \; t) \; (\text{iff} \; (\text{is}_\text{occurring}_\text{at} \; \text{occ} \; t) \; (\text{betweenEq} \; (\text{beginof} \; \text{occ}) \; t \; (\text{endof} \; \text{occ}))))$$

A.2 Occurrence Trees

An occurrence tree\(^2\) is the set of all discrete sequences of activity occurrences. They are isomorphic to substructures of the situation tree from situation calculus, the primary difference being that rather than a unique initial situation, each occurrence tree has a unique initial activity occurrence. As in the situation calculus, the poss relation is introduced to allow the statement of constraints on activity occurrences within the occurrence tree. Since the occurrence trees include sequences that modelers of a domain will consider impossible, the poss relation “prunes” away branches from the occurrences tree that correspond to such impossible activity occurrences. It should be noted that the occurrence tree is not the structure that represents the occurrences of subactivities of an activity. The occurrence tree is not representing a particular occurrence of an activity, but rather all possible occurrences of all activities in the domain.

The basic ontological commitments of the Occurrence Tree Theory are based on the following intuitions:

Intuition 1 An occurrence tree is a partially ordered set of activity occurrences, such that for a given set of activities, all discrete sequences of their occurrences are branches of the tree.

An occurrence tree contains all occurrences of all activities; it is not simply the set of occurrences of a particular (possibly complex) activity. Because the tree is discrete, each activity occurrence in the tree has a unique successor occurrence of each activity.

Intuition 2 There are constraints on which activities can possibly occur in some domain.

This intuition is the cornerstone for characterizing the semantics of classes of activities and process descriptions. Although occurrence trees characterize all sequences

\(^2\)The material in this section is adopted from http://www.mel.nist.gov/psl/psl-ontology/part12/occtree_page.html.
of activity occurrences, not all of these sequences will intuitively be physically possible within the domain. We will therefore want to consider the subtree of the occurrence tree that consists only of possible sequences of activity occurrences; this subtree is referred to as the legal occurrence tree.

The definitional extensions of the PSL Ontology use different constraints on possible activity occurrences as a way of classifying activities.

Intuition 3 Every sequence of activity occurrences has an initial occurrence (which is the root of an occurrence tree).

This intuition is closely related to the properties of occurrence trees. For example, one could consider occurrences to form a semilinear ordering (which need not have a root element) rather than a tree (which must have a root element). However, we are using occurrence trees to characterize the semantics of different classes of activities, rather than using the occurrence tree to represent history (which may not have an explicit initial event). In our case, it is sufficient to consider all possible interactions between the set of activities in the domain, and we lose nothing by restricting our attention to initial occurrences of the activities. For example, given the query “Can the factory produce 1000 widgets by Friday?”, one can take the initial state to be the current state, and the initial activity occurrences being the activities that could be performed at the current time.

Intuition 4 The ordering of activity occurrences in a branch of an occurrence tree respects the temporal ordering.

Within the theory of occurrence trees, the ordering over activity occurrences and the ordering over timepoints are distinct. The set of activity occurrences is partially ordered (hence the intuition about occurrence trees), but timepoints are linearly ordered (since this theory is an extension of PSL-Core). However, every branch of an occurrence tree is totally ordered, and the intuition requires that the beginof timepoint for an activity occurrence along a branch is before the beginof timepoints of all following activity occurrences on that branch.

A.2.1 Informal Semantics for Occurrence Trees

- (initial ?occ) is TRUE in an interpretation of the Occurrence Tree Theory if and only if the activity occurrence ?occ is a root of the occurrence tree.
• \((\text{earlier} \ ?\text{occ1} \ ?\text{occ2})\) is TRUE in an interpretation of the Occurrence Tree Theory if and only if the two activity occurrences \(\text{occ1}\) and \(\text{occ2}\) are on the same branch of the tree and \(\text{occ1}\) is closer to the root of the tree than \(\text{occ2}\). In other words, the earlier relation specifies the partial ordering over the activity occurrences in this tree.

• \((= (\text{successor} \ ?a \ ?\text{occ}) \ ?\text{occ2})\) is TRUE in an interpretation of the Occurrence Tree Theory if and only if \(\text{occ2}\) denotes the occurrence of \(?a\) that follows consecutively after the activity occurrence \(\text{occ}\) in the occurrence tree.

• \((\text{arboreal} \ ?s)\) is TRUE in an interpretation of the Occurrence Tree Theory if and only if \(?s\) is an element of the occurrence tree.

• \((\text{generator} \ ?a)\) is TRUE in an interpretation of the Occurrence Tree Theory if and only if \(?a\) is an activity whose occurrences are elements of the occurrence tree.

• \((\text{legal} \ ?\text{occ})\) is TRUE in an interpretation of the Occurrence Tree Theory if and only if the activity occurrence \(\text{occ}\) is an element of the legal occurrence tree.

• \((\text{poss} \ ?a \ ?\text{occ})\) is TRUE in an interpretation of the Occurrence Tree Theory if and only if the activity \(?a\) can possibly occur after the activity occurrence \(\text{occ}\).

• \((\text{precedes} \ ?\text{occ1} \ ?\text{occ2})\) is TRUE in an interpretation of the Occurrence Tree Theory if and only if the activity occurrence \(\text{occ1}\) is earlier than the activity occurrence \(\text{occ2}\) in the occurrence tree and such that all activity occurrences between them correspond to activities that are possible. This relation specifies the sub-tree of the occurrence tree in which every activity occurrence is the occurrence of an activity that is possible.

### A.2.2 Axioms of Occurrence Trees

**Axiom 1** The earlier relation is restricted to arboreal activity occurrences (that is, activity occurrences that are elements of the occurrence tree).

\[
(\forall \ ?s1 \ ?s2 \ (\text{earlier} \ ?s1 \ ?s2) \\
(\text{if} \ (\text{earlier} \ ?s1 \ ?s2) \\
(\text{and} \ (\text{arboreal} \ ?s1) \\
(\text{arboreal} \ ?s2))))
\]
Appendix A. Process Specification Language

**Axiom 2** The ordering relation over occurrences is irreflexive.

\[
\text{(forall (?s1 ?s2))} \\
\text{(if (earlier ?s1 ?s2) } \\
\text{(not (earlier ?s2 ?s1))})
\]

**Axiom 3** The ordering relation over occurrences is transitive.

\[
\text{(forall (?s1 ?s2 ?s3)) } \\
\text{(if (and (earlier ?s1 ?s2) } \\
\text{(earlier ?s2 ?s3))} \\
\text{(earlier ?s1 ?s3))}
\]

**Axiom 4** A branch in the occurrence tree is totally ordered.

\[
\text{(forall (?s1 ?s2 ?s3))} \\
\text{(if (and (earlier ?s1 ?s2) } \\
\text{(earlier ?s3 ?s2))} \\
\text{(or (earlier ?s1 ?s3) } \\
\text{(earlier ?s3 ?s1) } \\
\text{(= ?s3 ?s1))})
\]

**Axiom 5** No occurrence in the occurrence tree is earlier than an initial occurrence.

\[
\text{(forall (?s))} \\
\text{(iff (initial ?s) } \\
\text{(and (arboreal ?s) } \\
\text{(not (exists (?sp) } \\
\text{(earlier ?sp ?s))))})
\]

**Axiom 6** Every branch of the occurrence tree has an initial occurrence.

\[
\text{(forall (?s1 ?s2))} \\
\text{(if (earlier ?s1 ?s2) } \\
\text{(exists (?sp) } \\
\text{(and (initial ?sp) } \\
\text{(earlierEq ?sp ?s1))))})
\]
Axiom 7 There is an initial occurrence of each activity.

\[
\text{(forall } (?s \ ?a) \\
\quad \text{(if } \text{(occurrence_of } ?s \ ?a) \\
\quad \text{(iff } \text{(arboreal } ?s) \\
\quad \text{(generator } ?a))))
\]

Axiom 8 No two initial activity occurrences in the occurrence tree are occurrences of the same activity.

\[
\text{(forall } (?s1 \ ?s2 \ ?a) \\
\quad \text{(if } \text{(and } \text{(initial } ?s1) \\
\quad \text{(initial } ?s2) \\
\quad \text{(occurrence_of } ?s1 \ ?a) \\
\quad \text{(occurrence_of } ?s2 \ ?a)) \\
\quad (= ?s1 ?s2)))
\]

Axiom 9 The successor of an arboreal activity occurrence is an occurrence of a generator activity.

\[
\text{(forall } (?a \ ?o) \\
\quad \text{(iff } \text{(occurrence_of } \text{(successor } ?a \ ?o) ?a) \\
\quad \text{(and } \text{(generator } ?a) \\
\quad \text{(arboreal } ?o))))
\]

Axiom 10 Every non-initial activity occurrence is the successor of another activity occurrence.

\[
\text{(forall } (?s1 \ ?s2) \\
\quad \text{(if } \text{(earlier } ?s1 \ ?s2) \\
\quad \text{(exists } (?a \ ?s3) \\
\quad \quad \text{(and } \text{(generator } ?a) \\
\quad \quad \quad (= ?s2 \text{(successor } ?a \ ?s3))))))
\]

Axiom 11 An occurrence $s_1$ is earlier than the successor occurrence of $s_2$ if and only if the occurrence $s_2$ is later than $s_1$. 
(forall (?a ?s1 ?s2)
  (if (generator ?a)
      (iff (earlier ?s1 (successor ?a ?s2))
           (earlierEq ?s1 ?s2)))))

Axiom 12 The legal relation restricts arboreal activity occurrences.

(forall (?s)
  (if (legal ?s)
      (arboreal ?s)))

Axiom 13 If an activity occurrence is legal, all earlier activity occurrences in the occurrence tree are also legal.

(forall (?s1 ?s2)
  (if (and (legal ?s1)
           (earlier ?s2 ?s1))
      (legal ?s2)))

Axiom 14 The endof an activity occurrence is before to the beginof the successor of the activity occurrence.

(forall (?s1 ?s2)
  (if (earlier ?s1 ?s2)
      (before (endof ?s1) (beginof ?s2))))

A.2.3 Supporting Definitions of Occurrence Trees

Definition 1 An activity occurrence ?s1 precedes another activity occurrence ?s2 if and only if ?s1 is earlier than ?s2 in the occurrence tree and ?s2 is legal.

(forall (?s1 ?s2) (iff (precedes ?s1 ?s2)
                          (and (earlier ?s1 ?s2)
                               (legal ?s2)))))

Definition 2 An activity occurrence ?s1 is EarlierEq than an activity occurrence ?s2 if and only if it is either earlier than ?s2 or it is equal to ?s2.
(forall (?s1 ?s2) (iff (earlierEq ?s1 ?s2)
(and (arboreal ?s1)
(arboreal ?s2)
(or (earlier ?s1 ?s2)
(= ?s1 ?s2))))))

**Definition 3** An activity is poss at some occurrence if and only if the successor occurrence of the activity is legal.

(forall (?a ?s) (iff (poss ?a ?s)
(legal (successor ?a ?s))))

**Definition 4** An activity is a generator iff it has an initial occurrence in the occurrence tree. ;;

(forall (?a) (iff (generator ?a)
(exists (?s)
(and (initial ?s)
(occurrence_of ?s ?a)))))

**Definition 5** An activity occurrence is arboreal iff it is an element of an occurrence tree. ;;

(forall (?s) (iff (arboreal ?s)
(exists (?sp)
(earlier ?s ?sp)))))

### A.3 Atomic Activities

The core theory of Atomic Activities\(^3\) provides axioms for intuitions about the concurrent aggregation of primitive activities. Although the Subactivity Theory can represent arbitrary composition of activities, the composition of atomic activities is restricted to concurrency; to represent complex, or nonatomic, activities requires the Complex Activity Theory.

\(^3\)The material in this section is adopted from [http://www.mel.nist.gov/psl/psl-ontology/part12/atomic_page.html](http://www.mel.nist.gov/psl/psl-ontology/part12/atomic_page.html).
The basic ontological commitments of the Atomic Activity Theory are based on the following intuitions:

**Intuition 1** Concurrency is represented by the occurrence of one concurrent activity rather than multiple concurrent occurrences. Since concurrent activities may have preconditions and effects that are not the conjunction of the preconditions and effects of their activities, this core theory takes the following approach:

**Intuition 2** Every concurrent activity is equivalent to the composition of a set of primitive activities. Atomic activities are either primitive or concurrent (in which case they have proper subactivities). The Atomic Activities core theory introduces the function conc that maps any two atomic activities to the activity that is their concurrent composition. Essentially, what we call an atomic activity corresponds to some set of primitive activities.

### A.3.1 Informal Semantics for the Atomic Activity Theory

- (atomic ?a) is TRUE in an interpretation of the Atomic Activity Theory if and only if either ?a is primitive or it is the concurrent superposition of a set of primitive activities.

- (= ?a (conc ?a1 ?a2)) is TRUE in an interpretation of the Atomic Activity Theory if and only if ?a3 is the atomic activity that is the concurrent superposition of the two atomic activities ?a1 and ?a2.

- (primitive ?a) is TRUE in an interpretation of the Atomic Activity if and only if ?a has no proper subactivities

### A.3.2 Axioms of Atomic Activity Theory

**Axiom 1** Primitive activities are atomic.

\[
\forall \ ?a \ (\text{atomic} \ ?a) \\
\text{(if} \ (\text{primitive} \ ?a) \\
\text{(atomic} \ ?a))
\]

**Axiom 2** The function conc is idempotent.
(forall (?a)
  (= ?a (conc ?a ?a)))

**Axiom 3** The function conc is commutative.

(forall (?a1 ?a2)
  (= (conc ?a1 ?a2) (conc ?a2 ?a1)))

**Axiom 4** The function conc is associative.

(forall (?a1 ?a2 ?a3)
  (= (conc ?a1 (conc ?a2 ?a3)) (conc (conc ?a1 ?a2) ?a3)))

**Axiom 5** The concurrent aggregation of atomic action is an atomic action.

(forall (?a1 ?a2)
  (iff (atomic (conc ?a1 ?a2))
       (and (atomic ?a1)
            (atomic ?a2))))

**Axiom 6** An atomic activity ?a1 is a subactivity of an atomic activity ?a2 if and only if ?a2 is an idempotent for ?a1.

(forall (?a1 ?a2)
  (if (and (atomic ?a1) (atomic ?a2))
      (iff (subactivity ?a1 ?a2)
           (= ?a2 (conc ?a1 ?a2)))))

**Axiom 7** An atomic action has a proper subactivity if and only if there exists another atomic activity which can be concurrently aggregated.

(forall (?a1 ?a2)
  (if (and (atomic ?a2)
            (subactivity ?a1 ?a2)
            (not (= ?a1 ?a2))))
Appendix A. Process Specification Language

(exists (?a3)
  (and (atomic ?a3)
  (= ?a2 (conc ?a1 ?a3))
  (not (exists (?a4)
    (and (atomic ?a4)
    (subactivity ?a4 ?a1)
    (subactivity ?a4 ?a3)))))))

Axiom 8 The semilattice of atomic activities is distributive.

(forall (?a ?b0 ?b1)
  (if (and (atomic ?a)
    (atomic ?b0)
    (atomic ?b1)
    (subactivity ?a (conc ?b0 ?b1))
    (not (primitive ?a)))
  (exists (?a0 ?a1)
    (and (subactivity ?a0 ?a)
    (subactivity ?a1 ?a)
    (= ?a (conc ?a0 ?a1)))))

Axiom 9 Only atomic activities can be generator activities. Equivalently, only occurrences of atomic activities can be elements of an occurrence tree.

(forall (?a)
  (if (generator ?a)
    (atomic ?a)))

Axiom 10 Atomic activities are activities.

(forall (?a)
  (if (atomic ?a)
    (activity ?a)))
A.4 Complex Activities

This core theory provides the foundation for representing and reasoning about complex activities and the relationship between occurrences of an activity and occurrences of its subactivities. Occurrences of complex activities correspond to sets of occurrences of subactivities; in particular, these sets are subtrees of the occurrence tree.

The basic ontological commitments of the Complex Activities Theory are based on the following intuitions:

Intuition 1 An activity tree consists of all possible sequences of atomic subactivity occurrences beginning from a root subactivity occurrence. In a sense, activity trees are a microcosm of the occurrence tree, in which we consider all of the ways in which the world unfolds in the context of an occurrence of the complex activity. Any activity tree is actually isomorphic to multiple copies of a minimal activity tree arising from the fact that other external activities may be occurring during the complex activity.

Intuition 2 Different subactivities may occur on different branches of the activity tree i.e. different occurrences of an activity may have different subactivity occurrences or different orderings on the same subactivity occurrences. In this sense, branches of the activity tree characterize the nondeterminism that arises from different ordering constraints or iteration.

Intuition 3 An activity will in general have multiple activity trees within an occurrence tree, and not all activity trees for an activity need be isomorphic. Different activity trees for the same activity can have different subactivity occurrences. Following this intuition, the Complex Activities Theory does not constrain which subactivities occur. For example, conditional activities are characterized by cases in which the state that holds prior to the activity occurrence determines which subactivities occur. In fact, an activity may have subactivities that do not occur; the only constraint is that any subactivity occurrence must correspond to a subtree of the activity tree that characterizes the occurrence of the activity.

Intuition 4 Not every occurrence of a subactivity is a subactivity occurrence. There may be other external activities that occur during an occurrence of an activity. This theory

---

Appendix A. Process Specification Language

does not force the existence of complex activities; there may be subtrees of the occurrence tree that contain occurrences of subactivities, yet not be activity trees. This allows for the existence of activity attempts, intended effects, and temporal constraints; subtrees that do not satisfy the desired constraints will simply not correspond to activity trees for the activity.

A.4.1 Informal Semantics for Complex Activities

- \((\text{min\_precedes } ?s1 \ ?s2 \ ?a)\) is TRUE in an interpretation of the Complex Activity Theory if and only if \(?s1\) and \(?s2\) are subactivity occurrences in the activity tree for \(?a\), and \(?s1\) precedes \(?s2\) in the subtree. Any occurrence of an activity \(?a\) corresponds to an activity tree (which is a subtree of the occurrence tree). The activity occurrences within this subtree are the subactivity occurrences of the occurrence of \(?a\).

- \((\text{root } ?s \ ?a)\) is TRUE in an interpretation of the Complex Activity Theory if and only if the activity occurrence \(?s\) is the root of an activity tree for \(?a\).

- \((\text{subtree } ?a1 \ ?a2 \ ?occ2)\) is TRUE in an interpretation of the Complex Activity Theory if and only if every atomic subactivity occurrence in the activity tree for \(?a1\) is an element of the activity tree for \(?a2\).

- \((\text{do } ?a \ ?s1 \ ?s2)\) is TRUE in an interpretation of the Complex Activity Theory if and only if \(?s1\) is the root of an activity tree and \(?s2\) is a leaf of the same activity tree such that both activity occurrences are elements of the same branch of the activity tree.

- \((\text{leaf } ?s \ ?a)\) is TRUE in an interpretation of the Complex Activity Theory if and only if the activity occurrence \(?s\) is the leaf of an activity tree for \(?a\).

- \((\text{next\_subocc } ?s1 \ ?s2 \ ?a)\) is TRUE in an interpretation of the Complex Activity Theory if and only if \(?s1\) precedes \(?s2\) in the tree and there does not exist a subactivity occurrence that is between them in the tree.

A.4.2 Axioms of Complex Activities

Axiom 1 Occurrences in the activity tree for an activity correspond to atomic subactivity occurrences of the activity.
Appendix A. Process Specification Language

Axiom 2 Occurrences in the activity tree for an activity correspond to atomic subactivity occurrences of the activity.

\[
\text{(forall(?a ?s1 ?s2)} \\
\text{(if (min_precedes ?s1 ?s2 ?a)} \\
\text{(exists (?a1 ?a2)} \\
\text{(and (subactivity ?a1 ?a)} \\
\text{(atomic ?a2)} \\
\text{(subactivity ?a1 ?a2)} \\
\text{(occurrence_of ?s1 ?a2)})}))
\]

Axiom 3 Root occurrences in the activity tree correspond to atomic subactivity occurrences of the activity.

\[
\text{(forall(?a ?s)} \\
\text{(if (root ?s ?a)} \\
\text{(exists (?a2 ?a3)} \\
\text{(and (subactivity ?a2 ?a)} \\
\text{(atomic ?a3)} \\
\text{(subactivity ?a2 ?a3)} \\
\text{(occurrence_of ?s2 ?a3)})}))
\]

Axiom 4 All activity trees have a root subactivity occurrence.

\[
\text{(forall(?s1 ?s2 ?a)} \\
\text{(if (min_precedes ?s1 ?s2 ?a)} \\
\text{(exists (?s3)}
\]
(and (root ?s3 ?a)
(or (min_precedes ?s3 ?s1 ?a)
(= ?s3 ?s1)))))

**Axiom 5** No subactivity occurrences in an activity tree occur earlier than the root subactivity occurrence.

(forall(?s ?a)
(if (root ?s ?a)
(not (exists (?s2)
(and (activity_occurrence ?s2)
(min_precedes ?s2 ?s ?a))))))

**Axiom 6** An activity tree is a subtree of the occurrence tree.

(forall(?s1 ?s2 ?a)
(if (min_precedes ?s1 ?s2 ?a)
(precedes ?s1 ?s2)))

**Axiom 7** Root occurrences are elements of the occurrence tree.

(forall(?s ?a)
(if (root ?s ?a)
(legal ?s)))

**Axiom 8** Every legal atomic activity occurrence is an activity tree containing only one occurrence.

(forall(?a1 ?a2 ?s)
(if (and (atomic ?a1)
(occurrence_of ?s ?a1)
(legal ?s)
(subactivity ?a2 ?a1))
(root ?s ?a2)))

**Axiom 9** Activity trees are discrete.
(forall(?s1 ?s2 ?a)
  (if (min_precedes ?s1 ?s2 ?a)
    (exists (?s3)
      (and (next_subocc ?s1 ?s3 ?a)
        (or (min_precedes ?s3 ?s2 ?a)
          (= ?s3 ?s2))))))

**Axiom 10** Subactivity occurrences on the same branch of the occurrence tree are on the same branch of the activity tree.

(forall(?a ?s1 ?s2 ?s3)
  (if (and (min_precedes ?s1 ?s2 ?a)
            (min_precedes ?s1 ?s3 ?a)
            (precedes ?s2 ?s3))
    (min_precedes ?s2 ?s3 ?a)))

**Axiom 11** The activity tree for a complex subactivity occurrence is a subtree of the activity tree for the activity occurrence.

(forall(?a1 ?a2)
  (if (subactivity ?a1 ?a2)
    (not (exists (?s)
                 (and (activity_occurrence ?s)
                      (subtree ?s ?a2 ?a1)))))

**Axiom 12** Only complex activities can be arguments to the min.precedes relation.

(forall(?s1 ?s2 ?a)
  (if (min_precedes ?s1 ?s2 ?a)
    (not (atomic ?a)))

**Axiom 13** Subactivity occurrences on the same branch of the activity tree are linearly ordered by the min.precedes relation.

(forall(?a ?s1 ?s2 ?s3)
  (if (and (min_precedes ?s2 ?s1 ?a)
               (min_precedes ?s1 ?s2 ?a))
    (= ?s3 ?s2)))

(forall(?a ?s1 ?s2 ?s3)
  (if (and (min_precedes ?s2 ?s1 ?a)
               (min_precedes ?s3 ?s2 ?a))
    (= ?s3 ?s2)))
A.4.3 Supporting Definitions of Complex Activities

**Definition 1** An occurrence is the leaf of an activity tree if and only if there exists an earlier atomic subactivity occurrence but there does not exist a later atomic subactivity occurrence.

\[
\forall (s, a) \ (\text{iff} \ (\text{leaf} s a) \\
\exists (s1) \\
(\text{arboreal} s1) \\
(\text{or} \ (\text{root} s a) \\
(\text{min_precedes} s1 s a)) \\
(\text{not} \ (\exists (s2) \\
(\text{min_precedes} s s2 a))))))
\]

**Definition 2** The do relation specifies the initial and final atomic subactivity occurrences of an occurrence of an activity.

\[
\forall (a, s1, s2) \ (\text{iff} \ (\text{do} a s1 s2) \\
(\text{root} s1 a) \\
(\text{leaf} s2 a) \\
(\text{or} \ (\text{min_precedes} s1 s2 a) \\
(= s1 s2))))
\]

**Definition 3** An activity occurrence \( s2 \) is the next subactivity occurrence after \( s1 \) in an activity tree for \( a \) if and only of \( s1 \) precedes \( s2 \) in the tree and there does not exist a subactivity occurrence that is between them in the tree.

\[
\forall (s1, s2, a) \ (\text{iff} \ (\text{next_subocc} s1 s2 a) \\
(\text{min_precedes} s1 s2 a) \\
(\text{not} \ (\exists (s3) \\
(\text{activity_occurrence} s3) \\
(\text{min_precedes} s1 s3 a)))))
\]
Definition 4 The activity tree for \(?a_1\) with root occurrence \(?s_1\) contains an activity tree for \(?a_2\) as a subtree if and only if every atomic subactivity occurrence in the activity tree for \(?a_2\) is an element of the activity tree for \(?a_1\), and there is an atomic subactivity occurrence in the activity tree for \(?a_1\) that is not in the activity tree for \(?a_2\).

\[
\text{(forall } (?s_1 \ ?a_1 \ ?a_2) \ (\text{iff } (\text{subtree } ?s_1 \ ?a_1 \ ?a_2) \\
\quad \text{ (and } (\text{root } ?s_1 \ ?a_1) \\
\quad \quad \text{ (exists } (?s_2 \ ?s_3) \\
\quad \quad \quad \text{ (and } (\text{root } ?s_2 \ ?a_2) \\
\quad \quad \quad \quad \text{ (min_precedes } ?s_1 \ ?s_2 \ ?a_1) \\
\quad \quad \quad \quad \text{ (min_precedes } ?s_1 \ ?s_3 \ ?a_1) \\
\quad \quad \quad \quad \text{ (not (min_precedes } ?s_2 \ ?s_3 \ ?a_2)))))))
\]

Definition 5 The atomic subactivity occurrences \(?s_1\) and \(?s_2\) are siblings in an activity tree for \(?a\) iff they either have a common predecessor in the activity tree or they are both roots of activity trees that have a common predecessor in the occurrence tree.

\[
\text{(forall } (?s_1 \ ?s_2 \ ?a) \ (\text{iff } (\text{sibling } ?s_1 \ ?s_2 \ ?a) \\
\quad (\text{or } (\text{exists } (?s_3) \\
\quad \text{ (next_subocc } ?s_3 \ ?s_1 ?a) \\
\quad \text{ (next_subocc } ?s_3 \ ?s_2 ?a)))) \\
\quad (\text{and } (\text{root } ?s_1 ?a) \\
\quad \text{ (root } ?s_2 ?a) \\
\quad (\text{or } (\text{and } (\text{initial } ?s_1) \\
\quad \text{ (initial } ?s_2)) \\
\quad \text{ (exists } (?s_4 \ ?a_1 ?a_2) \\
\quad \text{ (and } (= ?s_1 (\text{successor } ?a_1 ?s_4)) \\
\quad (= ?s_2 (\text{successor } ?a_2 ?s_4))))) )))
\]
Appendix B

On Representing Conflict-free Petri nets in $\mathcal{D}_{scope}$

In this section, we provide arguments to support the conjecture that $\mathcal{D}_{scope}$ is a complete characterization to Petri nets, including the conflict-free ones. We leave the formulation of a rigorous proof for future work, however.

B.1 Conflict-free Petri nets

In a conflict-free Petri net, each place $p$ is either an input for at most one transition, or all such transitions enter backwards to the same place $p$ to construct a so-called self-loop. As such, a token in any place of such systems will not encounter potential conflicts in choosing which transition to go.

Definition 68 A Petri net $PN = (N, M_0)$, where $N$ is $(P, T, F)$, is a conflict-free system if $p^* \subseteq \cdot p$ for every place $p$ with more than one output transition.

An example of conflict-free Petri nets is given in Figure B.1, where the place $p_1$ enters exactly one transition node $t_2$. Meanwhile, the place $p_2$ enters both $t_1$ and $t_2$, and both $t_1$ and $t_2$ enter $p_2$, to construct two self-loops. If, for example, $t_1$ does not enter $p_2$, then firing $t_1$ will disable $t_2$ from firing, leading to “conflict”. The self-loop component in conflict-free Petri nets is given in Figure B.2. The basic property of a self-loop is that, as shown in B.2, firing of $t_1$ does not change the number of tokens in $p_1$. 

192
B.2 Self-loops in $D_{scope}$

Given a Petri net, a place $p$, which contains $k$ tokens in the current marking, the firing of an enabled transition node $t$ will not change the number of $k$ iff

- the place $p$ is the input and the output of $t$ (in other words, $t$ and $p$ construct a self-loop);

- $p$ is not directly connected to $t$ (for example, as shown in Figure B.1, firing of the transition $t_1$ will not affect the number of tokens $k$ at $p_1$, since $p_1$ and $t_1$ are not directly connected).

For the remaining two cases, however, $k$ of $p$ will be affected by firing of $t$: 
• the place $p$ is the input and is not the output of $t$, or

• the place $p$ is the output and is not the input of $t$.

Note that the above two cases cover all the conditions under which an action fire can cause a change to the number of tokens of a place in the successor situation. As such, $D_{\text{scope,ss}}$ is justified to be the Successor State Axiom for the fluent $\text{NumTkns}(p, s)$.\footnote{Consequently, we conjecture that $D_{\text{scope}}$ is a complete characterization of Petri nets. However, we leave the formulation of a rigorous proof for future work.} In particular, the definition $\gamma_f(p, n, a, s)$ on Page 92 of the thesis deals with the two cases that make marking changes. That is,

$$\gamma_f(p, n, a, s) \overset{\text{def}}{=} \gamma_{fe}(p, n, a, s) \lor \gamma_{fl}(p, n, a, s),$$

referring to the two sets of firing actions that cause the number of tokens at place $p$ on situation $do(a, s)$ to be equal to $n$:

- $\gamma_{fe}(p, n, a, s) \overset{\text{def}}{=} ((\exists t).\text{pre}(t, p) \land \neg\text{post}(t, p) \land n = \text{NumTkns}(p, s) + 1 \land a = \text{fire}(t))$

  (at situation $s$, the number of tokens at place $p$ is $(n - 1)$, and transition $t$, which enters $p$ and does not leave $p$, fires at $s$);

- $\gamma_{fl}(p, n, a, s) \overset{\text{def}}{=} ((\exists t).\text{pre}(p, t) \land \neg\text{post}(p, t) \land n = \text{NumTkns}(p, s) - 1 \land a = \text{fire}(t))$

  (at situation $s$, the number of tokens at place $p$ is $(n + 1)$, and transition $t$, which leaves $p$ and does not enter $p$, fires at $s$);

As shown in Section 6.5.1, conflict-free subclasses can be characterized by the following axiom:

$$(\forall p, t_3)(\exists(t_1, t_2) \text{post}(t_1, p) \land \text{post}(t_2, p) \land t_1 \neq t_2 \supset \text{post}(t_3, p) \supset \text{pre}(t_3, p)),$$

that is, for every place with more than one output transition (say $t_1$ and $t_2$), any of its output transitions is also its input transition.
Bibliography


