A Declarative Semantics of a Subset of PDDL with Time and Concurrency

November 22, 2006

Type of Thesis

Master Thesis

Title of Thesis

A Declarative Semantics of a Subset of PDDL
with Time and Concurrency

Author

Hu, Yuxiao
Matriculation Number 262104

Program

Software Systems Engineering

Department

Department of Computer Science,
RWTH Aachen University, Germany

Thesis Supervisor

Jens Claßen
Knowledge-Based Systems Group,
Department of Computer Science

First Examiner

Professor Gerhard Lakemeyer, Ph. D.
Knowledge-Based Systems Group,
Department of Computer Science

Second Examiner

Professor Matthias Jarke, Doctor
Chair V: Information Systems and Databases
Department of Computer Science

Place and Date of Submission

Aachen, 22 November 2006

Declaration

I hereby promise that this master thesis is completed by myself alone, referring to no sources or materials other than those explicitly specified.

(Hu, Yuxiao)

A Declarative Semantics of a Subset of PDDL with Time and Concurrency

by
Hu, Yuxiao
Matriculation Number 262104

Master Thesis
in the Program of Software Systems Engineering

Department of Computer Science
RWTH Aachen University, Germany

November 2006

Typeset in Computer Modern and AMSFonts using LaTeX 2_ε

Hu, Yuxiao
Rütscher Str. 121
Room 0202
52072 Aachen
Germany
Email: Yuxiao.Hu@rwth-aachen.de

Acknowledgments

First of all, I would like to express my special gratefulness to Prof. Gerhard Lakemeyer. As head of the master program and my academic advisor, he has been offering me advice and help since my application phase three years ago, and throughout my study in the master program. Without him, I would not have been able to get the chance to do this project, nor to further my study in this area in the University of Toronto.

I would also like to thank Jens Claßen for his suggestions and advice to my work in this thesis. He helped me the most in this project. The dozen of in-depth discussions provided me with new ideas, and identified many obvious mistakes that I had made in the drafts. Besides, he also offered me the LaTeX skeleton, which I am using for the final version of this thesis.

The Germany Academic Exchange Office (DAAD) and the Siemens Corporation have helped me the most in terms of finance. It is the scholarship from them that supported the whole duration of my stay in Germany. Without their fund, I would not have got the chance to fulfill my dream in the best engineering university in Germany. Special thanks to Mrs. Heike Gabler. As the direct correspondence in the DAAD, she has been so kind to me, and offered me lots of help in the scholarship program.

Thanks to my parents, and to my girlfriend, Chen. They always understand me, and supports me with their love. Their company and encouragement has been the source of my confidence to overcome the difficulties on the way.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Planning, as one of the main branches in artificial intelligence, involves finding a sequence of actions that achieves a specific goal from an initial state [RN03]. Ever since the introduction of the STRIPS planning system [FN71], there has been a great improvement in the efficiency and powerfulness of planners. With the right heuristics, modern planners are able to generate a plan of thousands of actions within one second. In order to compare the behavior of different planners and to share plan resources among them, it is essential to have a uniform language for describing planning problems.

Based on the STRIPS and the ADL [Ped89] formalisms, McDermott *et al.* defined the Planning Domain Definition Language (PDDL) [GHK+98], which served as the standard language for the first international planning competition (IPC'98). This language has roughly equivalent expressiveness as the ADL, and is capable of concisely modeling the classical planning problems.

With the recent fast development in the planning community, the research interest goes far beyond finding a sequence of actions, but focus on modeling the reality that a planning domain resides in. This includes the temporal properties of actions and the resource constraints in solving the problem. With this trend, Fox and Long extended the PDDL with durative actions [FL03], which is further improved by Edelkamp and Hoffmann to integrate derived predicates and timed initial literals [EH04]. Instead of considering the actions as immediate and duration-free changing operators of states, the new language, PDDL 2[1], is able to model durative actions and the continuous change of the

[1] In this paper, we use PDDL 2 to stand for PDDL 2.1 by Fox and Long and PDDL 2.2 by Edelkamp and Hoffmann as a whole. Similarly, we use PDDL 1 to stand for McDermott's PDDL 1.2.

states. Meanwhile, parallel execution of actions is also allowed in it. As a result, it is a language for realistically describing planning problems, and has become a standard language in the planning community.

Since the introduction of STRIPS, the semantics of planning languages has been a problem. Lifschitz proposed a definition with the state transition model [Lif86]. Fox and Long extended his idea to define the semantics for their language PDDL 2.1. Both of the two works rely on meta-theoretic operations on logical theories, and it is interesting to define the semantics declaratively.

For this purpose, a language capable of representing and reasoning about dynamic worlds is needed. A popular candidate is the situation calculus. Originally proposed by McCarthy [McC63], it is an expressive dialect of first-order logic, especially suitable for modeling the dynamics of the changing world. Its second-order extension, formalized by Reiter [Rei01], serves as the theoretical foundation of the powerful action language GOLOG [LRL$^+$97]. However, since the essential properties of situations are asserted by axioms, the proofs are usually complicated and lengthy in the language. Recently, Lakemeyer and Levesque proposed a new logic $\mathcal{ES}$, which reasons about situations without situation terms in the language [LL04]. Unlike the situation calculus, situational properties are embedded in the semantics of the language, so it is shown that $\mathcal{ES}$ simplifies semantic definitions and proofs while preserving similar expressiveness to the situation calculus [LL05].

As a result, both the situation calculus and the logic $\mathcal{ES}$ can serve as the logic foundation for defining the declarative semantics of planning languages. As an early attempt, Lin and Reiter formalized progression in the situation calculus, and showed that state updates in STRIPS can be modeled by first order progression. Recently, Claßen *et al.* extended this result to the ADL subset of PDDL with the logic $\mathcal{ES}$, and illustrated how the semantic mapping helps embed external planners to GOLOG programs [CELN07]. These are exciting results, since they bring together the research on planning and action formalisms, such that the two may benefit from each other.

1.2 Project Goals

With the existing work at hand, it is interesting to extend their results along the development of PDDL. In particular, defining a declarative semantics of the temporal extensions to PDDL is so far an open problem, although there have been quite a few literature on extending the classical situation calculus with concurrency and durative actions [Pin94, Rei96, DGLL00, GL00, GL01].

In this thesis, we shall study the temporal extensions to the situation calculus, and investigate the possibility to define a declarative semantics for the subset of Pddl with time and concurrency. When building the mapping, instead of using the classical situation calculus, we rely on the new logic $\mathcal{ES}$. There are at least two reasons for this decision. For one, the formulation in $\mathcal{ES}$ is more succinct, especially in our case where progression is intensively used in the proofs; for the other, the existing version of $\mathcal{ES}$ does not support time or concurrency, and it is interesting to see how the results in the situation calculus can be reproduced in $\mathcal{ES}$ in a similar manner.

At the end of this thesis, we shall obtain a semantic mapping between a subset of Pddl and and the basic action theories in the logic $\mathcal{ES}$. This subset is roughly the one defined by Fox and Long in the version 2.1, along with the timed initial literals defined by Edelkamp and Hoffmann in the version 2.2. Moreover, the correctness of our semantics will be proved.

1.3 Outline

The rest of this thesis is organized as follows: Chapter 2 briefly introduces the Pddl language, including an informal account of its syntax and its state-transitional semantics. Then we have an overview of the situation calculus in Chapter 3 and the logic $\mathcal{ES}$ in Chapter 4. In Chapter 5, we discuss some extensions to $\mathcal{ES}$, which will enable us to reason about, among other things, numerical and temporal properties in the logic. The main focus of this thesis is Chapter 6, where we actually define the semantic mapping. This is accompanied with a formal proof of correctness in Chapter 7. Finally, we conclude and suggest some directions for future research in Chapter 8.

Chapter 2

The Planning Domain Definition Language

This chapter offers a brief introduction to the *Planning Domain Definition Language* (PDDL), a standard language in the planning community for representing planning problems. We start with an overview of the developmental history of PDDL in Section 2.1. This is followed by the elaboration of the its syntax and semantics in Section 2.2. Finally, Section 2.3 defines a structure for concisely representing PDDL problem descriptions, for the benefit of later discussion.

2.1 The Development of PDDL

2.1.1 STRIPS

Planning had been an active area in artificial intelligence for almost 30 years before the introduction of PDDL. An early influential work owes to Fikes and Nilsson, who invented STRIPS, a mechanism for describing planning problems, and developed a problem solver with the same name built upon this mechanism [FN71].

A STRIPS problem definition consists of an initial state description, a set of operators and a goal description. A state is represented by a set of formulas in first-order logic; an operator corresponds to an action that changes the states. Each action operator consists of, besides its name and parameter list, a precondition, an add list and a delete list. It is possible to execute an action in a state, only if its precondition is satisfied in that state. The actual execution of the action changes the state to a new one by deleting the formulas in the operator's delete list and adding the ones in its add list. The task of a STRIPS planner is then to find an executable sequence of operators that transforms the initial state to a state that satisfies the goal formula.

This definition of STRIPS is a general one, in a sense that arbitrary first-order formulas are allowed in the language. In practice, however, only a restricted form is used, since many versions of STRIPS that go beyond a certain restriction get semantical problems, and thus lead to "unreasonable" behavior, as pointed out by Lifschitz [Lif86].

One of the typical restricted forms of STRIPS requires that state descriptions, condition formulas (*e.g.* action preconditions, goal description, premises for conditional effects), the add lists and the delete lists are all represented with conjunctions of grounded positive literals. The closed-world assumption is used in this case, indicating that all the conditions that cannot be derived from a state description is assumed to be false. For example, Figure 2.1 illustrates a problem definition in STRIPS in the blocks world. As we shall later see, this highly restricted form of STRIPS is called *relational* STRIPS, and defines the most basic subset, namely, the STRIPS subset, of the PDDL.

2.1.2 ADL

In order to extend the expressiveness of STRIPS with well-defined semantics, Pednault developed the *Action Description Language* (ADL) [Ped89, Ped94], which explores the middle ground between the highly expressive situation calculus and the computationally more beneficial STRIPS.

The syntax of ADL resembles that of STRIPS, augmented with conditional add and delete lists, which make it possible to describe situation-dependent effects. However, the semantics of ADL is quite different from that of STRIPS. While STRIPS operators define transformations on formulas, ADL schema define transformations on the algebraic structures representing the states.

The details of the syntax and semantics of the original ADL are out of the scope of this thesis. Here, we are interested in the ADL subset of PDDL, which, compared with the STRIPS subset, has the following new features:

- *Objects are typed.*
  For example, one may assert that obj_1 is a $Block$, and the single parameter to the action $moveToTable$ must be of type $Block$.

- *Negation, disjunction and quantification are allowed in conditions.*
  For example, one may express

$$\neg\exists y.On(y, x)$$

Init	(	
		$On(A, Table) \wedge On(B, Table) \wedge On(C, Table) \wedge$
		$Clear(A) \wedge Clear(B) \wedge Clear(C)$
	)	
Goal	(	
		$On(A, B) \wedge On(B, C)$
	)	
Action	(	$Move(b, x, y)$
	PRE:	$On(b, x) \wedge Clear(b) \wedge Clear(y)$
	ADD:	$On(b, y) \wedge Clear(x)$
	DEL:	$On(b, x) \wedge Clear(y)$
	)	
Action	(	$MoveFromTable(b, x)$
	PRE:	$On(b, Table) \wedge Clear(b) \wedge Clear(x)$
	ADD:	$On(b, x)$
	DEL:	$On(b, Table) \wedge Clear(x)$
	)	
Action	(	$MoveToTable(b, x)$
	PRE:	$On(b, x) \wedge Clear(b)$
	ADD:	$On(b, Table) \wedge Clear(x)$
	DEL:	$On(b, x)$
	)	

Figure 2.1: The blocks world defined using Strips.

as a precondition of $moveToTable(x)$ (which has the same meaning as $Clear(x)$), and

$$\forall(x : Block).\neg Fragile(x) \vee \neg(\exists y : Block).On(y, x)$$

as a goal condition.

- *Equality is a built-in predicate.*
  For example, for the precondition of $move(x, y)$, one may explicitly require that $\neg(x = y)$.

- *Conditional effects are allowed.*
  For example,
  $$\forall z.Above(y,z) \Rightarrow Above(x,z)$$
  says that the effect $Above(x,z)$ takes place only if $Above(y,z)$ is true in the current state.

2.1.3 PDDL

The first version of the Planning Domain Definition Language, PDDL 1.2, was defined by McDermott *et al.* in 1998 for the AIPS'98 planning competition [GHK+98]. It supports the following features:

- Basic STRIPS-style actions
- Conditional effects
- Universal quantification over dynamic universes
- Domain axioms over stratified theories
- Specification of safety constraints
- Specification of hierarchical actions composed of sub-actions and subgoals
- Management of multiple problems in multiple domains using different subsets of language features

The language is intended to express the physics of planning domains without specific advice for planners. As a unification of different existing formalisms at that time, the language brought a boom in the availability of shared planning resource, and soon became a standard language in the planning community to represent and exchange planning domains and problems.

With the rapid progress in the area of planning, researchers soon found the toy domains of propositional puzzles inadequate, and start to move their focus towards solving more realistic problems. This includes reasoning about numerical properties, resource constraints, time and concurrency, among other things. This trend led to the revisions and extensions of PDDL in the following years. Specifically, Fox and Long introduced numerical expressions, metrics and durative actions in the version 2.1, and defined a formal transitional semantics for the language [FL03]; Edelkamp and Hoffmann later extended the language again to formalize (reintroduce) the derived predicates (originally called domain axioms in PDDL 1.2), and introduced timed initial literals in the version 2.2 [EH04]; finally in 2005, Gerevini and Long proposed the version 3.0, where constraints and preferences on plans are incorporated [GL05].

In the following section of this chapter, we shall present a more detailed introduction to the syntax and semantics of a subset of the Pddl language that is interesting to the topic of this thesis. This is roughly the subset of Pddl 2.1 along with the timed initial literals in Pddl 2.2. Instead of giving all the formal definitions, we only do it in an intuitive way for space and scope reasons. We advise the readers to refer to the literatures for the formal details in the original definitions.

2.2 The Language

According to the definition by Fox and Long, the features in the Pddl are grouped into five *levels* with increasing expressiveness power. Level 1 is the Strips and Adl fragment of the language, where all properties are relational and all actions are non-durative; Level 2 extends Level 1 with numerical expressions; Level 3 enables the use of discretized durative actions; Level 4 further enables continuous durative actions; and Level 5 supports spontaneous events and physical processes. In the following subsections, our discussion will mainly follow this level definition.

2.2.1 The Basics

This subsection concerns with the basic structure of Pddl definitions and the features in the first level of the language.

A planning problem is described in two parts: the *domain description* and the *problem description*. The former defines how the framework of the world is like, *e.g.* what types of objects are there, what actions are available, what are their preconditions and effects, and so on. The latter defines the specific task in this framework, *e.g.* what are the objects concerned in the problem, what is the initial and goal state, and so on. The separation of the general domain description and the specific problem description simplifies the reuse of defined domains for several problems.

Let us illustrate the syntax of Pddl with an example in [GHK$^+$98]. Figure 2.2 defines the domain of a briefcase world. Words starting with a colon (`:`) are key words, and those starting with a question mark (`?`) are variables. Notice that all the logical sentences are written in prefix format for simplicity in the parsing phase.

Each domain begins with a `:requirements` field, which contains all the required features for solving problems in this domain. A planner that does

```
( define ( domain briefcase-world )
    ( :requirements :strips :equality :typing :conditional-effects )
    ( :types location physob )
    ( :constants ( B - physob ) )
    ( :predicates ( at ?x - physob ?l - location )
                  ( in ?x - physob ) )
    ( :action mov-b
        :parameters ( ?m ?l - location )
        :precondition ( and ( at B ?m ) ( not ( = ?m ?l ) ) )
        :effect ( and ( at b ?l ) ( not ( at B ?m ) )
                      ( forall ( ?z )
                          ( when ( and ( in ?z ) ( not ( = ?z B ) ) )
                                 ( and ( at ?z ?l ) ( not ( at ?z ?m ) ) ) ) ) ) )
    ( :action put-in
        :parameters ( ?x - physob ?l - location )
        :precondition ( not ( = ?x B ) )
        :effect ( when ( and ( at ?x ?l ) ( at B ?l ) )
                       ( in ?x ) ) )
    ( :action take-out
        :parameters ( ?x -physob )
        :precondition ( not ( = ?x B ) )
        :effect ( not ( in ?x ) ) )
```

Figure 2.2: The domain description of the briefcase world with PDDL

```
( define ( problem get-paid )
    ( :domain briefcase-world )
    ( :objects home office - location P D B - physob )
    ( :init ( at B home ) ( at P home ) ( at D home ) ( in P ) )
    ( :goal ( and ( at B office ) ( at D office ) ( at P home ) ) ) )
```

Figure 2.3: A problem description in the briefcase world domain with PDDL

Requirement	*Description*
`:strips`	Basic Strips-style adds and deletes
`:typing`	Allow type names in declarations of variables
`:negative-preconditions`	Allow `not` in goal descriptions
`:disjunctive-preconditions`	Allow `or` in goal descriptions
`:equality`	Support = as built-in predicate
`:existential-preconditions`	Allow `exists` in goal descriptions
`:universal-preconditions`	Allow `forall` in goal descriptions
`:quantified-preconditions`	=`:existential-preconditions` +`:universal-preconditions`
`:conditional-effects`	Allow `when` in action effects
`:adl`	= `:strips` + `:typing` + `:negative-preconditions` + `:disjunctive-preconditions` + `:equality` + `:quantified-preconditions` + `:conditional-effects`
`:durative-actions`	Allow durative actions Note that this does not imply `:fluents`.
`:duration-inequalities`	Allow duration constraints in durative actions using inequalities.
`:continuous-effects`	Allow durative actions to affect fluents continuously over the duration of action.
`:timed-initial-literals`	Allow timed initial literals

Table 2.1: Requirements in Pddl

not satisfy all of the requirements in the list can simply stop without trying to solve the problem. Table 2.1 lists the frequently-used requirements and their meanings in the language.

There are two built-in types in the language, namely, `object` (the default and super type of all objects) and `number`. In the example of Figure 2.2, however, due to the presence of the `:typing` requirement, objects in the domain may be typed with user-defined types. This example involves two such types: `location` for locations and `physob` for physical objects. An object in a domain that is invariant in all its problem instances is a *constant*. In the example, `B` is a constant denoting the (unique) briefcase.

Finally in a domain description, all the predicate, function and action symbols that may be used in the domain are declared. As shown in Figure 2.2, there are two predicate symbols, $At(x, l)$ (whether physical object x is at location y) and $In(x)$ (whether x is in the briefcase), and three actions symbols, $mov\_b(m, l)$ (which moves the briefcase from location m to location l), $put\_in(x, l)$ (which puts physical object x into the briefcase if both are at location l) and $take\_out(x)$ (which takes the physical object x out of the briefcase).

The way in which action operators are defined in PDDL is similar to STRIPS. Each definition has three sections. The first is a list of parameters and their corresponding types. When executing an action, these parameters are instantiated with ground terms of the correct types. The second is the precondition. For an action to execute in a state, its precondition must be satisfied in the state. The third section is the effect formula. It is a combination of the add and delete lists in STRIPS, and defines how the state should be modified after the execution of the action.

Figure 2.3 shows an example of a problem instance in the briefcase-world defined in Figure 2.2. It specifies which domain the problem belongs to, what objects there are, what the initial state is, and which goal is to be achieved.

With the definition of the problem and its corresponding domain, a planner has full information about the planning problem, and can therefore proceed to generate a plan as solution.

2.2.2 Numerics and Metrics

The use of numerics and plan metrics comprises the second level of the PDDL.

Numerical expressions are formally integrated to PDDL in the version 2.1, although `number` is already a built-in type since PDDL 1.2. According to [FL03], numbers always have real values, and their possible roles are not distinguished. Following the old convention, all numerical expressions, including comparison predicates, are written in prefix format. Three operations can be used to update a numerical variable: `assign`, `increase` and `decrease`, with their obvious functionalities.

There are two important restrictions to the use of numerical expressions. First, they are not terms in the language. In particular, numerical expressions cannot appear as arguments to predicate, function or action parameters. This restriction helps keep the instantiation of predicates and actions finite. Second, all function values in the language are numbers, rather than objects, *i.e.* all functions are of the form $Object^n \to \Re$. This eliminates the identity problem that would raise if functions of the form $Object^n \to Object$ were allowed.

```
( define (domain jug-pouring )
    ( :requirement :typing :fluents )
    ( :types jug )
    ( :functions
          ( amount ?j - jug )
          ( capacity ?j -jug )

    ( :action pour
          :parameters (?jug1 ?jug2 - jug )
          :precondition ( >= ( - ( capacity ?jug2 ) ( amount ?jug2 ) )
                              ( amount ?jug1 ) )
          :effect ( and ( increase ( amount ?jug2 ) ( amount ?jug1 ) )
                        ( assign ( amount ?jug1 ) 0 ) )
)
```

Figure 2.4: Jug-pouring problem with numerical expressions

Figure 2.4 is an example in [FL03] to illustrate the use of numerical expressions in PDDL 2.1.

With a stable extension of numerics to the PDDL core, Fox and Long further introduced the *plan metric* to the language. It is a field in the problem definition, indicating on which basis the quality of the plan is evaluated.

For example, the sentence

```
(:metric minimize (+ (* 2 (fuel-used car)) (fuel-used truck)))
```

says that a good plan has a smaller weighted sum of the fuel used by the car and the truck. As we can see, the metric is simply a numerical expression, under the condition that all the numerical functions in the expression is instrumented in the domain, *i.e.* they must be initialized and each action must specify how it may change the their values.

The way a planner deals with metrics is not specified in the standard. The planner has the freedom to either use the metrics as a guideline when producing a plan, or simply ignore them during plan generation, and only evaluate the plan with them *post hoc*.

2.2.3 Durative Actions

One of the most important contributions of [FL03] is the definition of durative actions in the PDDL language. Domains with durative actions that allow for

```
( :durative-action produce
      :parameters( ?p - product )
      :duration( = ?duration 5 )
      :condition( )
      :effect( at end ( increase ( quantity ?p )
                                 ( * ?duration ?produce-rate ) ) )
)
( :durative-action consume
      :parameters( ?p - product )
      :duration( = ?duration 7 )
      :condition( >= ( quantity ?p )
                      ( * ?duration ?consume-rate ) )
      :effect( at start ( decrease ( quantity ?p )
                                   ( * ?duration ?consume-rate ) ) )
)
```

Figure 2.5: Producer-consumer example with discrete durative actions

at most discretized effects reside in Level 3 of the PDDL, whereas the ones with continuous durative actions reach Level 4.

To describe a durative action, two time points and an interval have to be taken into account, namely, at the start, at the end and during the happening of the action. As a result, temporally annotated conditions and effects are used in the representation of a durative action schema. Syntactically, (`at start p`), (`at end p`) and (`over all p`) are used to assert that `p` holds at the three time points, respectively. Note that `(over all p)` asserts that `p` holds in the *open* interval between the start and end time points, which makes it possible to accommodate different conditions within the interval and at the end points of an action, and thus allows for a higher degree of expressiveness.

From the perspective of PDDL 2.1, time is point-based rather than interval-based. Action end points are the only places where the world state is changed. Propositions hold their values over the half-open interval that is closed on the left (when the event asserting it occurs) and open on the right (when an event negates it).

As already mentioned, there are two kinds of durative actions: *discretized durative actions* and *continuous durative actions*.

Discretized durative actions provide a simplified view of numerical changes, in that all the numerical updates are modeled with step functions, as if the change happens only at the end points of the action. Figure 2.5 shows how discrete durative action can be used to model the producer-consumer problem. While the quantity of a resource increases and decreases gradually in the producing and consuming process, it is modeled as an instant increase at the end and an instant decrease at the beginning, respectively, with the discretized durative actions. Note that a conservative resource consumption model is used here, since the total amount of consumption is subtracted at the beginning and the total amount of production is added only in the end. This ensures the validity of a plan, but also eliminates the case where the resource is not enough at the beginning but is soon supplied sufficiently with a production action.

In order to model a continuous change, continuous durative actions can be used. A continuous durative action has an internal variable `#t` referring to the current time after the start of an action. As a result, it is possible to calculate the value of any variable that has a fixed changing rate with a linear function. In the producer-consumer example, for instance, we can assert in the effect section of the producer

(increase (quantity ?p) (* #t produce-rate))

and in that of the consumer

(decrease (quantity ?p) (* #t consume-rate))

Unlike the discrete case, we have the correct value for the quantity of the product at any time point during the interval of the producing and/or the consuming actions.

2.2.4 Timed Initial Literals

Timed initial literals are integrated to PDDL in the version 2.2 by Edelkamp and Hoffmann [EH04], as a means to represent simple predetermined exogenous events.

Like the initial state description, timed initial literals also specify how the world is like, not initially, but at a certain time point in the future. Figure 2.6 shows an example problem definition with timed initial literals, saying that the shop opens at 9 and closes at 20.

According to their definition, timed initial literals is only used in Level 3 in combination with discretized durative actions.

```
( :init
   ( at 9 ( shop-open ) )
   ( at 20 ( not ( shop-open ) ) )
)
```

Figure 2.6: Shop opening hours expressed with timed initial literals

2.3 A Structural Representation

As a mentioned in Section 2.2, PDDL has a strict and well-defined syntax. However, when we later discuss the semantic mapping in this thesis, it is not so convenient to always use the original syntax. For notational benefits, we extract the essential information from the problem specification, and define a structure to express it in a succinct way.

In building this structure, we only concentrate on the semantic aspect, *i.e.* how to shuffle the formulas in a way that can be easily addressed to, when we later talk about the semantical mappings. We simply omit the syntactical details in the translation from the PDDL style to the abstract representation.

For instance, a precondition formula for an action

```
(>= (- (capacity ?jug2) (amount ?jug2)) (amount ?jug1))
```

may be simply translated to

$$capacity(jug_2) - amount(jug_2) \geq amount(jug_1)$$

and an effect formula

```
increase (energy ?robot) (* (recharge-rate ?robot) ?duration)
```

may be rewritten as

$$energy'(robot) = energy(robot) + recharge\_rate(robot) \times duration$$

where $energy(robot)$ is the energy level before the action, and $energy'(robot)$ is the one after.

Definition 2.1 (The planning problem). A planning problem D in PDDL is a tuple $\langle \mathbf{T}, \mathbf{F}, \mathbf{f}, \mathbf{O}, \mathbf{A}, I, TI, G \rangle$, where

- $\mathbf{T}$ is a finite list of $\tau_1, \cdots, \tau_l$. If the domain does not have the `:typing` requirement, then $\mathbf{T}$ only contains one single type *Object*.

- $\mathbf{F}$ is a finite list of fluent predicates $F_1, \cdots, F_n$.

- **f** is a finite list of fluent functions $f_1, \cdots, f_m$. This field is non-empty only if the domain has the `:fluents` requirement.

- **O** is a finite list of objects, possibly with associated primitive types $o_1 : \tau_{o_1}, \cdots, o_k : \tau_{o_k}$. If the domain does not have the `:typing` requirement, then each τ_{o_i} is either empty or the type `object`.

- **A** is a tuple $\langle \mathfrak{A}, \widetilde{\mathfrak{A}} \rangle$ of PDDL operators, where $\mathfrak{A}$ is a finite list of simple actions $A_1, \cdots, A_p$, and $\widetilde{\mathfrak{A}}$ is a finite list of durative actions $\widetilde{A}_1, \cdots, \widetilde{A}_q$ (see below). $\widetilde{\mathfrak{A}}$ is non-empty only if the domain has the `:durative-actions` requirement.

- I is the initial state description.

- TI is a finite list of timed initial literals $\langle t_1, \varphi_1 \rangle, \cdots, \langle t_r, \varphi_r \rangle$. This field is non-empty only if the domain has the `:timed-initial-literals` requirement.

- G is the goal description

With the above definition, the ADL subset of PDDL, for example, can be represented by

$$\big\langle \mathbf{T}, \mathbf{F}, \emptyset, \mathbf{O}, \langle \mathfrak{A}, \emptyset \rangle, I, \emptyset, G \big\rangle$$

Indeed, Claßen *et al.* defined such a structure for representing problems defined with the ADL subset of PDDL [CELN07]. Our definition here is simply an extension to their work. So, exception for the new items **f**, TI and **A**, all the elements have the old meanings.

For the new items, it is not difficult to understand the elements **f** and TI: the set **f** is similar to **F** except that it defines the fluent functions instead of predicates; the timed initial literals $\langle t_i, \varphi_i \rangle$ asserts that φ_i starts to hold at time t_i.

Now, we shall elaborate the normal forms of the action operators in **A**. Compared with the ADL subset, the action operators are extended in two important ways. First, with the introduction of numerical expressions, both the precondition and the effects may involve functional fluents. Second, actions may be durative. So in the following, Section 2.3.1 is devoted to adding functional fluents to the representation of simple actions, followed by the specification of the normal form of durative actions in Section 2.3.2

2.3.1 Functional updates

In [CELN07], each action is defined by a triple $\langle \vec{z} : \vec{\tau}, \pi_A, \epsilon_A \rangle$, where $\vec{z}$ and $\vec{\tau}$ are the arguments and their corresponding types, π_A is the precondition formula specifying what conditions must hold before A may be activated, and ϵ_A is the effect formula specifying what facts are to be added or deleted in the state description after the execution of A.

Here, $\vec{z}:\vec{\tau}$ is to be understood as a list of pairs $z_i:\tau_i$. Both π_A and ϵ_A have all their free variables among $\vec{z}$, and are constructed with only the $\vec{z}$ and symbols in $\mathbf{T}$, $\mathbf{F}$, $\mathbf{f}$ and $\mathbf{O}$. Precondition formulas are defined like follows: every atomic formula and every equality atom $(t_1 = t_2)$, where t_i is either a variable, a constant or an instance from $\mathbf{f}$, is a precondition formula; if ϕ_1 and ϕ_2 are precondition formulas, then so are $\phi_1 \wedge \phi_2$, $\neg\phi_1$ and $\forall x:\tau.\phi_1$.

The normal form of ϵ_A is a conjunction of effects of the following forms, at most one of each kind for each fluent F_j:

$$\begin{array}{l} \bigwedge_{F_j} \forall \vec{x_j}:\tau_{F_j}.\big(\gamma^+_{F_j,A}(\vec{x_j},\vec{z}) \Rightarrow F_j(\vec{x_j})\big) \\ \bigwedge_{F_j} \forall \vec{x_j}:\tau_{F_j}.\big(\gamma^-_{F_j,A}(\vec{x_j},\vec{Z}) \Rightarrow \neg F_j(\vec{x_j})\big) \end{array} \tag{2.1}$$

Here, $\gamma^+_{F_j,A}(\vec{x_j},\vec{z})$ is the *add condition*, which makes $F_j(\vec{x_j})$ true; and $\gamma^-_{F_j,A}(\vec{x_j},\vec{z})$ is the *delete condition*, which makes $F_j(\vec{x_j})$ false.

Compared with the original work of Pednault, it does not contain the update conditions for function symbols, because no fluent functions exist in their subset of the language. However, our target language, PDDL 2, allows for (numerical) fluent functions in the domain, so we have to return to the more general definition of the normal form of actions.

Formally, a simple action A is defined by a triple $\langle \vec{z}:\vec{\tau}, \pi_A, \epsilon_A\rangle$ where $\vec{z}$ and $\vec{\tau}$ are the arguments and their typing constraints, respectively, as usual; π_A is the precondition that may involve numerical formulas; and ϵ_A is the effect formula with the form

$$\begin{array}{l} \bigwedge_{F_j} \forall \vec{x_j}:\tau_{F_j}.\big(\gamma^+_{F_j,A}(\vec{x_j},\vec{z}) \Rightarrow F_j(\vec{x_j})\big)\wedge \\ \bigwedge_{F_j} \forall \vec{x_j}:\tau_{F_j}.\big(\gamma^-_{F_j,A}(\vec{x_j},\vec{z}) \Rightarrow \neg F_j(\vec{x_j})\big)\wedge \\ \bigwedge_{f_j} \forall \vec{x_j}:\vec{\tau_{f_j}}, y:\mathbb{R}.\big(\gamma^v_{f_j,A}(\vec{x_j},y,\vec{z}) \Rightarrow f_j(\vec{x_j}) = y\big) \end{array} \tag{2.2}$$

Compared with Equation (2.1), we add a new condition $\gamma^v_{f_j,A}(\vec{x_j},y)$ here to denote the necessary and sufficient condition to change the value of $f_j(\vec{x_j})$ to y. We sometimes need a weaker condition that the value of $f_j(\vec{x_j})$ changes at all. We denote this condition as

$$\gamma_{f_j,A}(\vec{x_j},\vec{z}) \equiv \exists y.\gamma^v_{f_j,A}(\vec{x_j},y,\vec{z})$$

In many cases, this is equivalent to removing all sub-formulas mentioning y from $\gamma^v_{f_j,A}(\vec{x_j},y,\vec{z})$.

For example, if for some action *Heat*, $\gamma^v_{temperature,Heat}(x_1,x_2,y)$ is the formula

$$Heater(x_2) \wedge On(x_1,x_2) \wedge y = 100$$

then $\gamma_{temperature,Heat}(x_1, x_2)$ is simply the formula

$$Heater(x_2) \wedge On(x_1, x_2)$$

2.3.2 Duration and temporal annotation

Apart from the numerics, an important extension in PDDL 2.1 is the introduction of durative actions. The set of durative actions is denoted with $\widetilde{\mathfrak{A}}$, the second component of $\mathbf{A}$. Compared with simple actions, the normal form of durative actions looks more complicated, since it has the extra duration constraints, and the preconditions and effects are temporally annotated.

To take these changes into consideration, our proposal is to represent a durative action $\widetilde{A} \in \widetilde{\mathfrak{A}}$ with a four-element tuple $\langle \vec{z} : \vec{\tau}, \delta_{\widetilde{A}}, \pi_{\widetilde{A}}, \epsilon_{\widetilde{A}} \rangle$, where $\delta_{\widetilde{A}}$ is the duration constraint, and $\pi_{\widetilde{A}}$ and $\epsilon_{\widetilde{A}}$ are the precondition and effect formulas respectively.

Here, $\delta_{\widetilde{A}}$ is a tuple $\langle \delta^s_{\widetilde{A}}, \delta^e_{\widetilde{A}} \rangle$, where $\delta^s_{\widetilde{A}}$ is the duration constraint with temporal annotation `at start` and $\delta^e_{\widetilde{A}}$ is the one with annotation `at end`. For a condition without temporal annotation, we assume that all the functional values in the formula are constants, so it is not important whether we place it in $\delta^s_{\widetilde{A}}$ or $\delta^s_{\widetilde{A}}$. However, for simplicity that will soon become obvious, we choose to put it in $\delta^e_{\widetilde{A}}$.

Both $\delta^s_{\widetilde{A}}$ and $\delta^e_{\widetilde{A}}$ are a direct rewriting of the corresponding sub-formula in the `:duration` section. For example, if the duration constraint of action $\widetilde{A}$ is

(:duration (and ($\geq$?duration 5) ($<$?duration 10)))

then $\delta^s_{\widetilde{A}}$ is simply the formula TRUE and $\delta^e_{\widetilde{A}}$ is

$$5 \leq duration \wedge duration < 10$$

As a more complex case, for the the duration constraint

(:duration (and (at start (>= ?fuel-left ($*$?duration ?consume-rate)))
(at end (= ?distance ($*$?velocity ?duration)))))

we have

$$\begin{cases} \delta^s_{\widetilde{A}} \equiv (fuel\_left \geq duration \times consume\_rate) \\ \delta^e_{\widetilde{A}} \equiv (distance = velocity \times duration) \end{cases}$$

Due to the temporal annotations, the form of $\pi_{\widetilde{A}}$ and $\epsilon_{\widetilde{A}}$ have also changed a lot, compared with π_A and ϵ_A for a simple action A. $\pi_{\widetilde{A}}$ is now a triple of $\langle \pi^s_{\widetilde{A}}, \pi^o_{\widetilde{A}}, \pi^e_{\widetilde{A}} \rangle$, where $\pi^s_{\widetilde{A}}$, $\pi^o_{\widetilde{A}}$ and $\pi^e_{\widetilde{A}}$ are the conditions that must hold at the start, within the interval and at the end of the durative action, respectively.

All the conditions at the same temporal point are combined to form one single condition.[1] For example, if the precondition for a durative action $\widetilde{A}$ is

$$\begin{array}{l} \text{and (at start } \phi_1) \\ \quad\; \text{(at start } \phi_2) \\ \quad\; \text{(over all } \phi_3) \end{array}$$

then we have

$$\begin{cases} \pi^s_{\widetilde{A}} & \equiv & \phi_1 \wedge \phi_2 \\ \pi^o_{\widetilde{A}} & \equiv & \phi_3 \\ \pi^e_{\widetilde{A}} & \equiv & \text{TRUE} \end{cases}$$

The effect formula $\epsilon_{\widetilde{A}}$ consists of a triple $\langle \epsilon^s_{\widetilde{A}}, \epsilon^o_{\widetilde{A}}, \epsilon^e_{\widetilde{A}} \rangle$.

Here, $\epsilon^s_{\widetilde{A}}$ is the start effect, consisting of all the effect formulas in the PDDL description with the temporal annotation `at start`. Each single effect formula has the form

$$\varphi_i \Rightarrow \psi_i$$

where φ_i is the premise of the conditional effect ψ_i. When ψ_i is an absolute (non-conditional) effect, φ_i is simply the formula TRUE.

The end effect, $\epsilon^e_{\widetilde{A}}$, is more subtle to define. Unlike the start effect, where all the conditional effects, if any, always have the premise annotated with `at start`, the end effect may have conditional effects with premise annotated with `at start`, `over all`, `at end` or any combination of the three.

As a result, we define the normal form of an end-effect formula as

$$\langle \varphi^s_i, \varphi^o_i, \varphi^e_i \rangle \Rightarrow \psi_i$$

where φ^s_i, φ^o_i and φ^e_i are the conjunction of all premise of ψ_i annotated with `at start`, `over all` and `at end`, respectively.

In the following, we call a conditional effect whose premise and effect have the same temporal annotation an *intra-temporal conditional effect*, and one whose premise and effect have different temporal annotation an *inter-temporal conditional effect*. Later, we shall see that for intra-temporal conditional effects, the approach given by Claßen and Lakemeyer is sufficient, whereas we have to resort to auxiliary properties to handle inter-temporal ones.

The so-called "overall effect", $\epsilon^o_{\widetilde{A}}$, is used only when defining continuous effects, which have the form of the PDDL description

```
<op> P (* #t Q)
```

[1]Remember that conditions for durative actions have been restricted to *conjunctions* of temporally annotated expressions.

where `<op>` is either `increase` or `decrease`, `P` is the continuous fluent to be updated, and `Q` is the changing rate. For example, the continuous effect

```
(increase (temperature ?p) (* #t (heat-rate)))
```

means that the action will make the temperature of p increase at a rate of *heat_rate*, and

```
(decrease ?fuel-used (* #t (use-rate ?v)))
```

means that the action will decrease the quantity of $fuel\_used$ at a rate of the use rate of vehicle v.

To denote a continuous effect in our structure, we use a triple $\langle op, P, Q\rangle$, where op is either $+$ (for `increase`) or $-$ (for `decrease`), and P and Q are simple syntactical transformations from their counterparts in the PDDL sentence.

As an example, suppose that we have the following effects for a durative action $\widetilde{A}$

```
:effects (and (at start $\psi_1$)
              (when (at start $\varphi_2$)
                    (at start $\psi_2$))
              (decrease P (* #t Q))
              (at end $\psi_4$)
              (when (at start $\varphi_5$)
                    (and (at start $\psi_5$)
                         (at end $\psi_6$)))
              (when (and (at start $\varphi_6$) (at end $\varphi_7$))
                         (at end $\psi_7$))
              (when (at end $\varphi_8$)
                    (at end $\psi_8$))
              (when (over all $\varphi_9$)
                    (at end $\psi_9$)))
```

then the normal form of it has the components

$$\begin{cases} \epsilon^s_{\widetilde{A}} & \equiv & (\text{TRUE} \Rightarrow \psi_1) \wedge (\varphi_2 \Rightarrow \psi_2) \wedge (\varphi_5 \Rightarrow \psi_5) \\ \epsilon^o_{\widetilde{A}} & = & \{\langle -, P, Q\rangle\} \\ \epsilon^e_{\widetilde{A}} & \equiv & (\langle \text{TRUE}, \text{TRUE}, \text{TRUE}\rangle \Rightarrow \psi_4) \wedge \\ & & (\langle \varphi_5, \text{TRUE}, \text{TRUE}\rangle \Rightarrow \psi_6) \wedge \\ & & (\langle \varphi_6, \text{TRUE}, \varphi_7\rangle \Rightarrow \psi_7) \wedge \\ & & (\langle \text{TRUE}, \text{TRUE}, \varphi_8\rangle \Rightarrow \psi_8) \wedge \\ & & (\langle \text{TRUE}, \varphi_9, \text{TRUE}\rangle \Rightarrow \psi_9) \end{cases}$$

Chapter 3

The Situation Calculus

In this chapter, we present an intuitive introduction to the situation calculus. First proposed by McCarthy [McC63] and later refined by Reiter [Rei01], it is a language for representing and reasoning about dynamically changing worlds (Sections 3.1–3.5), and serves as the theoretical basis for the action language GOLOG (Section 3.6).

3.1 The Language

The language that we introduce here is based on Reiter's formalism, which has the foundational axioms to define the structure of situations and uses successor state axioms to solve the frame problem.

A domain in the situation calculus has three sorts of elements, namely, *action* for actions, *situation* for situations, and *object* for everything else. The world evolves with the execution of an action from one situation to another. A situation is represented by a sequence of actions, with S_0 denoting the initial situation where no action has occurred yet. The binary function $do(a, s)$ denotes the situation obtained by executing action a in situation s. For example, $do\Big(drop\big(block_1\big), do\big(pickup(block_1), S_0\big)\Big)$ is the situation obtained by first picking up $block_1$ and then dropping it in the initial situation. In the language, the only function symbols of sort *situation* are S_0 and $do(a, s)$.

Properties that may change their value from situation to situation are called *fluents*. There are in general two types of fluents. A *relational fluent* is a predicate that carries a situation term as its last argument. For example, $On(obj_3, obj_5, S_0)$ means that obj_3 is on top of obj_5 in the initial situation. Similarly, a *functional fluent* is a function that carries a situation term as its last argument. For example, $fuel\_level\Big(car, do\big(drive(rome, berlin), S_0\big)\Big)$

may stand for the fuel level of the car after driving from Rome to Berlin from the initial situation.

Apart from the domain specific fluents, there are two special fluent predicates with a fixed meaning in all domains. The fluent $Poss(a, s)$ means that it is possible to execute action a in situation s, whereas $s_1 \sqsubset s_2$ means that situation s_1 precedes situation s_2, *i.e.* the sequence of actions characterizing s_1, $\{a_i\}_{i=1,\cdots,n_1}$, is a proper sub-sequence of that of s_2, $\{a_i\}_{i=1,\cdots,n_2}$, where $n_1 < n_2$.

In contrast to the fluents, rigid predicates and functions are the ones that have a fixed value in all situations, and thus have no situation term as their parameters. For example, $Fragile(x)$, $x \leq y$ are rigid predicates, and $fatherOf(x)$, $x \times y$ are rigid fluents[1].

Finally, connectives and other symbols, such as $=$, $\neg$, $\wedge$ and $\exists$, and abbreviations, such as $\vee$, $\forall$, $\supset$ and $\equiv$ have the same meaning as in the first order logic.

3.2 The Basic Action Theory

The dynamics of a domain is characterized by a set of axioms called the *basic action theory* (BAT). In order to illustrate the form of the BAT, we need the following definition.

Definition 3.1 (Uniform formulas)**.** A formula is *uniform in* σ if and only if it does not mention the predicates $Poss$ or $\sqsubset$, does not quantify over variables of sort *situation*, does not mention equality on situations and whenever it mentions a term of sort *situation*, the term is σ and appears in the situation argument position of a fluent.

With the concept of uniform formulas, the basic action theory is defined in the form

$$\Sigma = \mathcal{FA} \cup \Sigma_{post} \cup \Sigma_{pre} \cup \Sigma_{una} \cup \Sigma_0$$

where[2]

[1] Like in most literatures, we use numbers, their operations and their relations freely in the language without explicitly axiomatizing them. Instead, the standard interpretation is used. We shall make a short discussion on how the axiomatization may be done, when we later extend the logic $\mathcal{ES}$ in Chapter 5. Following the conventions, we write numerical expressions in an infix way here, *e.g.* $x < y$ and $x \times y$ instead of $<(x, y)$ and $\times(x, y)$, respectively. While $<$ is considered a binary predicate symbol, $>$, $x \leq y$ and $x \geq y$ are abbreviations.

[2] All free variables are assumed to be universally quantified, unless otherwise specified.

- $\mathcal{FA}$ is the following foundational axioms in the situation calculus

$$S_0 \neq do(a,s) \tag{3.1}$$
$$do(a_1,s_1) = do(a_2,s_2) \supset (a_1 = a_2 \wedge s_1 = s_2) \tag{3.2}$$
$$(\forall P).P(S_0) \wedge (\forall a,s)\big[P(s) \supset P(do(a,s))\big] \supset (\forall s)P(s) \tag{3.3}$$
$$\neg(\exists s.)(s \sqsubset S_0) \tag{3.4}$$
$$s \sqsubset do(a,s') \equiv \big(Poss(a,s') \wedge s \sqsubseteq s'\big) \tag{3.5}$$

  Here, (3.1) and (3.2) are the unique names assumption for actions; (3.3) is second-order induction saying that every valid situation results from executing a number of actions from S_0; (3.4) and (3.5) form the inductive definition of the relation $\sqsubset$, where $s \sqsubseteq s'$ is the abbreviation of the formula $s \sqsubset s' \vee s = s'$.

  $\mathcal{FA}$ is domain independent, and is the only place where axiom about the structure of situations may appear.

- Σ_{post} is a set of successor state axioms. Proposed by Reiter [Rei91], the successor state axiom is a simple and effective solution to the frame problem for deterministic actions.

  For each relational fluent F, it has the form

$$F\big(\vec{x}, do(a,s)\big) \equiv \Phi(\vec{x},a,s)$$

  and for each functional fluent f, it has the form

$$f\big(\vec{x}, do(a,s)\big) = y \equiv \phi(\vec{x},y,a,s)$$

  where $\Phi(\vec{x},a,s)$ and $\phi(\vec{x},y,a,s)$ are formulas uniform in s.

  $\Phi(\vec{x},a,s)$ has all its free variables among $\vec{x}$, a and s, and according to Reiter's approach of construction, it usually has the form

$$\gamma_F^+(\vec{x},a,s) \vee F(\vec{x},s) \wedge \neg\gamma_F^-(\vec{x},a,s)$$

  where γ_F^+ and γ_F^- are the positive and negative condition for the fluent F, respectively. The former is the condition when F starts holding, whereas the latter is the one where F stops holding.

  Similarly, $\phi(\vec{x},y,a,s)$ has all its free variables among $\vec{x}$, y, a and s, and usually has the form

$$\gamma_f(\vec{x},y,a,s) \vee f(\vec{x},s) = y \wedge \neg\exists y'.\gamma_f(\vec{x},y',a,s)$$

Additionally, $\phi(\vec{x}, y, a, s)$ must satisfy the consistency condition

$$\forall x.\exists y.\phi(\vec{x}, y, a, s) \wedge \big(\forall y'.\phi(\vec{x}, y', a, s) \supset y = y'\big)$$

which ensures that there exists a unique y that satisfy the formula. Otherwise, the value of $f\big(\vec{x}, do(a, s)\big)$ becomes either undefined or inconsistent.

As an example, consider two fluent symbols On and $floor$. $On(x, y)$ holds in the situation $do(a, s)$ if and only if a is the action to move x onto y, or x was initially on y in situation s, and a is not an action that moves it away; $floor = y$ holds in the situation $do(a, s)$ if and only if a is the action to move one floor upward and the elevator was originally in floor $y - 1$, or a is the action to move one floor downward and it was originally in floor $y + 1$, or the it was originally in floor y, and a is not an action that moves it. Formally, the successor state axioms for the two fluents can be written as

$$\begin{aligned}
&On\big(x, y, do(a, s)\big) \equiv \\
&\quad a = move(x, y) \vee \\
&\quad On(x, y, s) \wedge \neg\big((\exists z).a = move(x, z) \vee a = moveToTable(x)\big) \\
&floor\big(do(a, s)\big) = y \equiv \\
&\quad a = moveUp \wedge floor(s) = y - 1 \vee \\
&\quad a = moveDown \wedge floor(s) = y + 1 \vee \\
&\quad floor(s) = y \wedge \neg\big(a = moveUp \vee a = moveDown\big)
\end{aligned}$$

- Σ_{pre} is a set of action precondition axioms, and has the form

$$Poss(A(\vec{x}), s) \equiv \Psi_A(\vec{x}, s)$$

where Ψ_A is a formula uniform in s with all free variables among $\vec{x}$ and s, indicating the condition for action A to be applicable.

For example, it is possible to move a box onto another, if and only if both boxes are clear on the top. Formally,

$$Poss(moveTo(x, y), s) \equiv \forall z.\neg On(z, x, s) \wedge \neg On(z, y, s)$$

The precondition axioms may be used to determine whether a situation s is an *executable* one. For this purpose, we introduce a predicate $Executable(s)$, which is an abbreviation defined as follows:

$$Executable(s) \triangleq (\forall a, s^*).do(a, s^*) \sqsubseteq s \supset Poss(a, s^*) \tag{3.6}$$

- Σ_{una} is a set of unique names axioms for actions. For distinct action symbols A and B,

  $$A(\vec{x}) \neq B(\vec{y})$$

  and identical actions have identical arguments

  $$A(x_1, \cdots, x_n) = A(y_1, \cdots, x_n) \supset x_1 = y_1 \wedge \cdots \wedge x_n = y_n$$

- Σ_0 is a finite set of sentences uniform in S_0, characterizing the initial situation.

  For example, the following is a description of the initial state in the blocks world, Where Blocks A, B, C form a pile and D is standing alone on the table:

  $$\begin{gathered}\forall x.\neg On(A, x, S_0) \wedge \neg On(D, x, S_0)\\ On(B, A, S_0) \wedge On(C, B, S_0)\\ \forall x.\neg On(x, C, S_0) \wedge \neg On(x, D, S_0)\end{gathered}$$

  In addition, situation-independent facts may also be declared in Σ_0, such as $On(x, y) \supset \neg On(y, x)$.

With the definition above, to see whether a basic action theory is satisfiable, Pirri and Reiter further showed the following satisfiability theorem [PR99].

Theorem 3.2 (Relative satisfiability)**.** *A basic action theory* Σ *is satisfiable if and only if* $\Sigma_{una} \cup \Sigma_0$ *is.*

So far, we have talked about the representation of a dynamic domain in the situation calculus. Now, it is interesting to reason whether a sentence, possibly involving non-initial situations, holds or not. This is called the *projection problem.* Since we only know the initial world, there are two approaches to solve this problem. One is to *regress* the sentence from the current situation through all the actions performed to the initial situation, and test whether the regressed sentence holds in it; the other is to *progress* the initial database through the actions to the current one, and test whether the current database entails the sentence. We will briefly introduce both approaches in the following two sections.

3.3 Regression

In this and following sections, we use the abbreviations

$$\begin{aligned} do([\,], s) &\triangleq s \\ do([a_1, \cdots, a_n], s) &\triangleq do\Big(a_n, do\big(a_{n-1}, \cdots do(a_1, s) \cdots\big)\Big) \end{aligned}$$

so that the situation terms look more compact and suggestive.

Now, let us first get some intuitive ideas about regression. Suppose that we have the successor state axiom for F

$$F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$$

and we want to know whether $\Sigma \models F(\vec{x_0}, do(A, S_0)$. For this purpose, we can simply test whether $\Sigma \models \Phi_F(\vec{x_0}, A, S_0)$, since the two are equivalent. Furthermore, the right hand side of the latter is uniform in the situation S_0, so this only involves the test whether $\Sigma_0 \cup \Sigma_{una} \models \Phi_F(\vec{x_0}, A, S_0)$.

In general, for an arbitrary *regressible formula*, we can perform this regression iteratively to convert it to an equivalent form that mentions the initial situation only. Here is what we mean by saying that a formula is regressible.

Definition 3.3 (Regressible formula)**.** A formula W is *regressible* if and only if

1. Every term of sort *situation* mentioned by W has the syntactic form $do([\alpha_1, \cdots, \alpha_n], S_0)$ for some $n \geq 0$, and for terms $\alpha_1, \cdots, \alpha_n$ of sort *action.*

2. For every atom of the form $Poss(\alpha, \sigma)$ mentioned by W, α has the form $A(t_1, \cdots, t_n)$ for some n-ary action function symbol A.

3. W does not quantify over situations.

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

Definition 3.3 essentially requires that all the situation terms that appear in a regressible formula are obtained by executing a certain number of actions from S_0.

Definition 3.4 (The regression operator)**.** For a regressible formula W, the regression operator $\mathcal{R}$ is defined recursively as follows:

- $\mathcal{R}[W] = W$ if W is situation independent;

- $\mathcal{R}[W] = W$ if W is a relational fluent atom of the form $F(\vec{x}, S_0)$;
- $\mathcal{R}[Poss(A(\vec{x}), s)] = \mathcal{R}[\Pi_A(\vec{x}, s)]$, where $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$;
- $\mathcal{R}[F(\vec{x}, do(a, s))] = \mathcal{R}[\Phi_F(\vec{x}, a, s)]$, where $F(\vec{x}, do(a, s))] \equiv \Phi_F(\vec{x}, a, s)$;
- $\mathcal{R}[W] = \mathcal{R}[(\exists y.)\phi_f(\vec{x}, y, a, s) \wedge W|_y^{f\left(\vec{x}, do(a,s)\right)}]$, where W is a formula mentioning $f\left(\vec{x}, do(a, s)\right)$, and $W|_y^{f\left(\vec{x}, do(a,s)\right)}$ is the formula obtained by simultaneously substituting all occurrences of $f\left(\vec{x}, do(a, s)\right)$ in W with y;
- $\mathcal{R}[\neg W] = \neg\mathcal{R}[W]$;
- $\mathcal{R}[W_1 \wedge W_2] = \mathcal{R}[W_1] \wedge \mathcal{R}[W_2]$;
- $\mathcal{R}[(\exists v).W] = (\exists v).\mathcal{R}[W]$.

The following *regression theorem* is the core of Reiter's formulation of the situation calculus [Rei01]

Theorem 3.5 (The regression theorem)**.** *Suppose W is a regressible sentence and Σ is a basic action theory of actions, then*

$$\Sigma \models W \textit{ iff } \Sigma_0 \cup \Sigma_{una} \models \mathcal{R}[W]$$

where $\mathcal{R}[W]$ is the regressed version of W that mentions only situation S_0.

With Theorem 3.5, we can reason about the validity of any regressible sentence by regressing it to an equivalent formula that is uniform in S_0, and testing whether the latter is entailed by the initial database.

For example, suppose we have the successor state axioms

$$\begin{aligned}
&At\big(x, do(a, s)\big) \equiv \\
&\quad \exists x'.a = drive(x', x) \vee \\
&\quad At(x, s) \wedge \neg\exists x''.a = drive(x, x'') \\
&fuel\big(do(a, s)\big) = y \equiv \\
&\quad \exists x, x'.a = drive(x, x') \wedge y = fuel(s) + cost(x, x') \vee \\
&\quad fuel(s) = y \wedge \neg\exists x, x''.a = drive(x, x'')
\end{aligned}$$

then for the formula

$$\begin{aligned}
W \equiv At\Big(&berlin, do\big(drive(a, b), S_0\big)\Big) \wedge \\
&fuel\Big(do\big(drive(a, b), S_0\big)\Big) > 0
\end{aligned}$$

its regression $\mathcal{R}[W]$, according to Definition 3.4, is

$$\begin{gathered}\big(\exists x'.drive(a,b) = drive(x', berlin) \vee \\ At(berlin, S_0) \wedge \neg\exists x''.drive(a,b) = drive(berlin, x'')\big) \wedge \\ \Big(\exists y.\big(\exists x, x'.drive(a,b) = drive(x,x') \wedge y = fuel(S_0) + cost(x,x') \vee \\ fuel(S_0) = y \wedge \neg\exists x, x''.a = drive(x,x'')\big) \wedge \\ y > 0\Big)\end{gathered}$$

Applying the unique names axioms, to determine whether $\Sigma \models W$, it is equivalent to test whether

$$\begin{aligned}\Sigma_{una} \cup \Sigma_0 \quad \models \quad & \big(b = berlin \vee At(berlin, S_0) \wedge \neg(a = berlin)\big) \wedge \\ & \big(\exists y.y = fuel(S_0) + cost(a,b) \wedge y > 0\big)\end{aligned}$$

3.4 Progression

Regression, as described in the previous section, is a simple and effective way to solve the projection problem. However, when the action sequence becomes long, such as in the case where the agent has performed a large number of actions, it becomes computationally expensive to regress a sentence to the initial situation. In this case, *progression* may be more efficient.

Definition 3.6. A set of sentences Σ_α is said to be a progression of Σ_0 through an action sequence α to S_α with respect to $\Sigma = \Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_{una} \cup \mathcal{FA}$ if and only if

- Σ_α is uniform in S_α;
- $\Sigma \models \Sigma_\alpha \cup \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_{una} \cup \mathcal{FA}$;
- For an observer standing in situation S_α and looking into the future, she cannot distinguish between a model for the original Σ and a model for the new theory $\Sigma_\alpha \cup \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_{una} \cup \mathcal{FA}$.

The following theorem says that the progression of a theory through an action is unique, up to logical equivalence.

Theorem 3.7. *If Σ_α and Σ'_α are two progressions of Σ_0 to the situation S_α, then Σ_α and Σ'_α are logically equivalent.*

Although intuitively, the ideas of regression and progression look similar, the latter is actually much more difficult than the former. In fact, Lin and Reiter showed that that progression, in general, is not first-order definable, although it is always second-order definable [FR97]. Fortunately, there are a few useful special cases of Σ, whose progression *is* first order definable. The following are two of them:

- *Relatively Complete Initial Databases*, where in Σ_0, each fluent is mentioned once in a sentence of the form

$$(\forall \vec{x}).F(\vec{x}, S_0) \equiv \Pi_F(\vec{x})$$

where $\Pi_F(\vec{x})$ is a situation independent formula whose free variables are among $\vec{x}$.

In this case, consider a basic action theory Σ with a relatively complete initial database Σ_0, and a successor state axiom for fluent F:

$$F\big(\vec{x}, do(a, s)\big) \equiv \Phi(\vec{x}, a, s)$$

Let S_α denote the situation $do(\alpha, S_0)$, we have

$$F(\vec{x}, S_\alpha) \equiv \Phi(\vec{x}, \alpha, S_0) \tag{3.7}$$

Remember that $\Phi(\vec{x}, \alpha, S_0)$ is a first order formula uniform in S_0, with all free variables among $\vec{x}$, α and S_0. Moreover, each occurrence of S_0 in the formula is an argument to a fluent. Suppose that $G(\vec{t}, S_0)$ appears in $\Phi(\vec{x}, \alpha, S_0)$, since the initial database is relatively complete, we may replace all the occurrences of $G(\vec{t}, S_0)$ with $\Pi_G(\vec{t})$. The repeated application of this replacement eliminates all S_0 in the right hand side of Equation (3.7), which means that $\Phi(\vec{x}, \alpha, S_0)$ finally becomes independent from S_0, and thus Equation (3.7) is reduced to the form

$$F(\vec{x}, S_\alpha) \equiv \Psi(\vec{x}, \alpha)$$

It can be shown that the initial theory Σ_α built up in this way is a progression of the Σ_0 through action α. Furthermore, Σ_α is also relatively complete, and thus this procedure of progression can be applied repeatedly.

- *Context-Free Successor State Axioms with Isolated Fluents*, where all the successor state axioms in Σ_{post} are of the form

$$F\big(\vec{x}, do(a, s)\big) \equiv \gamma_F^+(\vec{x}, a) \vee F(\vec{x}, s) \wedge \neg\gamma_F^+(\vec{x}, a)$$

where $\gamma_F^+(\vec{x}, a)$ and $\gamma_F^-(\vec{x}, a)$ are situation independent formulas whose free variables are all among those in $\vec{x}$ and a. Meanwhile, the initial database Σ_0 contains only *isolated fluents*, which means that each sentence in Σ_0 is situation independent with the form

$$E \supset (\neg)F(x_1, \cdots, x_n, S_0)$$

where F is a fluent and E is a situation independent formula.

For a basic action theory Σ of this kind and a ground primitive action α, we construct a new initial theory Σ_α, which is initially empty, and expand it with the following procedure:

1. If $\phi \in \Sigma_0$ is situation-independent, then add ϕ to Σ_α;
2. For any fluent F, add to Σ_α the sentences
$$\gamma_F^+(\vec{x}, \alpha) \supset F(\vec{x}, S_\alpha)$$
$$\gamma_F^-(\vec{x}, \alpha) \supset \neg F(\vec{x}, S_\alpha)$$
3. For any fluent F, if $\forall x.E \supset F(\vec{x}, S_0)$ is in Σ_0, then add to Σ_α the sentence
$$\big(E \wedge \neg\gamma_F^-(\vec{x}, \alpha)\big) \supset F(\vec{x}, S_\alpha)$$
4. For any fluent F, if $\forall x.E \supset \neg F(\vec{x}, S_0)$ is in Σ_0, then add to Σ_α the sentence
$$\big(E \wedge \neg\gamma_F^+(\vec{x}, \alpha)\big) \supset \neg F(\vec{x}, S_\alpha)$$

Lin and Reiter proved that under the constraint that Σ_0 is coherent[3] and consistent[4], the new theory Σ_α is a progression of Σ_0 through action α.

Like in the previous case, since the progressed database Σ_α also has the property that all fluents are isolated, this procedure of progression can be applied repeatedly, and thus an initial database can be progressed through arbitrary sequence of ground primitive actions.

As a brief introduction, we stop our dig into progression here. In Section 6.1.1 we shall return to this topic again, and show that the STRIPS subset of PDDL can be given a declarative semantics by relating its update to first-order progression in the situation calculus.

3.5 Time and Concurrency

The discussion so far is a non-temporal one, in the sense that actions are instant state changing operators, and the happening of them is abstracted into a sequence, *i.e.* only the action names and their ordering matter. In the real world, however, actions always happen at certain time points, usually have a duration, and may sometimes happen concurrently. With the hope to

[3]A database is coherent iff whenever $(\forall\vec{x}).E_1 \supset F(\vec{x}, S_0)$ and $(\forall\vec{x}).E_2 \supset \neg F(\vec{x}, S_0)$ are both in Σ_0, then $\big\{\phi | \phi \in \Sigma_0 \text{is situation independent}\big\} \models (\forall\vec{x}).\neg(E_1 \wedge E_2)$

[4]A database is consistent iff $\Sigma_0 \cup \Sigma_{una} \models \neg(\exists\vec{x}, a).\gamma_F^+(\vec{x}, a) \wedge \gamma_F^-(\vec{x}, a)$

enable the situation calculus to reason about time and concurrency, Pinto and Reiter have had a study along this direction [Pin94, Rei96]. In this section, we shall conduct a rough survey on their work and results. In fact, many new approaches to the formulation of time and concurrency in the situation calculus have emerged in the recent years, such as [DGLL00, GL00]. However, we choose to use Pinto and Reiter's formalism, due to its close similarity to the semantics of Pddl (*e.g.* both are off-line), which is our focus in this thesis.

3.5.1 Time

Pinto and Reiter's proposal for adding time to the situation calculus is to add a new temporal argument to all instantaneous actions, denoting the happening time of the action. For example, $refuel(car)$ is now extended to $refuel(car, 5.5)$, to mean that the action of refueling the car happens at time 5.5. As previously mentioned, real numbers like 5.5, their operations like $+$,$*$, and their relations like $\leq$ are used in the language without axiomatization, and the standard interpretation of them is assumed.

With this extension, the foundational axioms need to be modified to capture the meaning, and ensure correct semantics of the time.

First, a new function symbol $time : action \rightarrow number$ is needed to denote the happening time of an action. Formally,

$$time\big(A(\vec{x}, t)\big) = t$$

For example, $time\big(refuel(car, 5.5)\big) = 5.5$.

Then, it is necessary to also denote the starting time of the current situation. This is handled with a new function symbol $start : situation \rightarrow number$. $start(s)$ stands for the time when the situation s begins. The value of $start(s)$ function is captured by the axiom

$$start\big(do(a, s)\big) = time(a)$$

With these two functions, it is now possible to rule out some counter-intuitive situations. For example, it makes little sense, if any, to have a situation like

$$do\Big(turn\_on\big(light, 4\big), do\big(refuel(car, 5.5), S_0\big)\Big)$$

since after doing the refueling action at time 5.5, it is not possible to go back to the past and turn on the light at time 4. In order to identify this kind of "impossible" situations, the executability of a state is redefined:

$$\begin{aligned} Executable(s) \triangleq (\forall a, s^*).do(a, s^*) \sqsubseteq s \supset \\ Poss(a, s^*) \wedge start(s^*) \leq time(a) \end{aligned}$$

Notice that the case $start(s^*) = time(a)$ is allowed. This is the case when an action happens exactly the same as its predecessor. For instance,

$$do\Big(turn\_on\big(light, 5.5\big), do\big(refuel(car, 5.5), S_0\big)\Big)$$

may be an executable situation, since the actions of turning on the light and refueling the car conceptually happen concurrently at time 5.5. We shall introduce concurrency in more detail in Section 3.5.3.

3.5.2 Durative actions

A durative action is one that has a duration. In real life, many actions are durative. For example, it takes 9 hours for a passenger plane to fly from Frankfurt to Beijing, and $\frac{volume(tank)}{refuel\_rate}$ to fill an empty tank with oil. In this subsection, we investigate Pinto and Reiter's approach for representing such durative actions.

According to them, each durative action is represented by a relational fluent and two instantaneous (non-durative) actions to denote the start and the end event of the durative action.

For example, the durative action to walk from position x to position y, $walk(x, y)$, is represented with two instantaneous actions, $startWalk(x, y, t)$ and $endWalk(x, y, t)$. Meanwhile, a fluent predicate symbol $Walking(x, y, s)$ is introduced. $Walking(x, y, s)$ is true if and only if the agent is walking from x to y in situation s. This is equivalent to saying that $startWalk(x, y, t)$ has been executed in s, but $endWalk(x, y, t)$ has not.

In this example, the $walk(x, y)$ durative action can be axiomatized as follows:

$$\begin{aligned}
Poss\big(startWalk(x, y, t), s\big) \equiv& \neg(\exists u, v).Walking(u, v, s) \wedge location(s) = x\\
Poss\big(endWalk(x, y, t), s\big) \equiv& Walking(x, y, s)\\
Walking\big(x, y, do(a, s)\big) \equiv& (\exists t).a = startWalk(x, y, t) \vee\\
& Walking(x, y, s) \wedge \neg\exists(t').a = endWalk(x, y, t')\\
location\big(do(a, s)\big) = y \equiv& (\exists x, t).a = endWalk(x, y, t) \vee\\
& location(s) = y \wedge\\
& \quad \neg(\exists x, y', t').a = endWalk(x, y', t')
\end{aligned}$$

3.5.3 Concurrency

Concurrency means the simultaneous happening of more than one action. For instantaneous actions, this means that the actions happen at exactly the

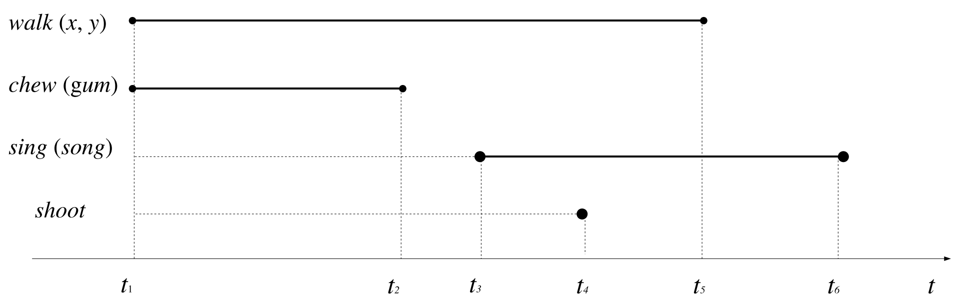

Figure 3.1: An example of concurrent actions

same time; for durative ones, this means that there is a temporal overlap in the durations of the actions.

In this subsection, we introduce two popular models for concurrency. The *interleaved concurrency* is the simpler one. It is based on the classical situation calculus, and the basic action theory need not to be changed. However, the expressiveness of this approach is limited, in that there are some cases which are not representable with it. In contrast, *true concurrency* is more general, but to accommodate it, one needs to extend the basic action theory with happenings of sets of actions.

Interleaved concurrency

In interleaved concurrency, actions that happen concurrently are serialized. So although they happen at the same time in concept, there is an ordering among them.

Let us illustrate the use of interleaved concurrency with an example in Figure 3.1.

In this example, the durative actions $walk(x, y)$ and $chew(gum)$ start at the same time at t_1. The former has a longer duration, ending at time t_5, whereas the latter only lasts till t_2. So the period between t_1 and t_2 is the interval of the concurrent happening. As we can see, the durative action $sing(song)$ happening between t_3 and t_6 also overlaps partially with $walk(x, y)$. Furthermore, the simple action *shoot* happens when the durative actions $walk(x, y)$ and $sing(song)$ are in progress.

With interleaved concurrency, these happenings may be modeled with the following action sequence

$$\begin{aligned}&[startWalk(x, y, t_1), startChew(gum, t_1),\\ &\qquad endChew(gum, t_2), startSing(song, t_3), shoot(t_4), \qquad (3.8)\end{aligned}$$

$$endWalk(x, y, t_5), endSing(song, t_6)]$$

Notice that we write $startWalk(x, y, t_1)$ before $startChew(gum, t_1)$, although nothing prevents us from writing it the other way round as

$$\begin{aligned}[&startChew(gum, t_1), startWalk(x, y, t_1),\\ &\quad endChew(gum, t_2), startSing(song, t_3), shoot(t_4),\\ &\quad endWalk(x, y, t_5), endSing(song, t_6)]\end{aligned} \tag{3.9}$$

Ideally, we would like to consider (3.8) and (3.9) as equivalent happening sequences, but in fact, the situations resulting from executing the two in a same situation are distinct, since the actions sequences are different. Fortunately, under the condition that the actions that happen at the same time are independent from each other, it can be shown that conclusions drawn on the resulting situations are the same. So, the interleaving account for modeling concurrency is usually considered appropriate, if the outcome is independent of the order in which the actions are interleaved.

A typical example where interleaved concurrency fails is the scenario of a duel, where the precondition for shooting the gun is that the duelist is alive. It is possible for both duelists to shoot each other at exactly the same time, since before the shooting, they are still alive, and immediately afterward, both dies. Unfortunately, this setting cannot be modeled with interleaved concurrency, since however we serialize the shootings, only one death can occur. In the next subsection, we shall see that this example can be dealt with correctly with true concurrency.

True concurrency

In order to accommodate true concurrency, each of the happenings is no longer represented by a single action, but instead, by a set of simple actions. For instance, in order to denote the situation after concurrently executing A_1 and A_2 in situation s, we use the situation term

$$do(\{A_1, A_2\}, s)$$

Like for numbers, we do not axiomatize sets, but simply use the standard semantics for sets, their operations like $\cup$ and $\cap$, and their relations like $\subseteq$ and $\in$.

Now, let us see how the foundational axioms should be modified in order to capture the notion of true concurrency with sets of actions. Neglecting the time dimension for actions for the moment, this is done by defining the new *do* function and *Poss* predicate with their first parameter being a set of actions. Specifically, we have

$$S_0 \neq do(c,s) \tag{3.10}$$
$$do(c,s) = do(c',s') \supset c = c' \wedge s = s' \tag{3.11}$$
$$(\forall P).P(S_0) \wedge (\forall c,s).[P(s) \supset P(do(c,s))] \supset (\forall s).P(s) \tag{3.12}$$
$$\neg s \sqsubset S_0 \tag{3.13}$$
$$s \sqsubset do(c,s') \equiv s \sqsubseteq s' \tag{3.14}$$
$$Poss(c,s) \supset (\exists a).a \in c \wedge (\forall a).[a \in c \supset Poss(a,s)] \tag{3.15}$$

where (3.10)–(3.14) are the counterparts of (3.1)–(3.5) with concurrency extension, and (3.15) says that if a concurrent action set is possible, then it contains at least one action, and all its simple actions must themselves be possible. Note that the reverse direction of (3.15) need not hold, due to the precondition interaction problem [Pin94].

With these foundational axioms, if we have the precondition for $shoot(x,y)$ as

$$Poss\big(shoot(x,y),s\big) \equiv \neg Dead(x,s)$$

and the successor state axiom for *Dead* as

$$Dead\big(x, do(c,s)\big) \equiv \exists y.shoot(y,x) \in c \vee Dead(x,s)$$

then it is easy to prove that both duelists may die due to simultaneous shootings, *e.g.*

$$\begin{gathered} Poss\big(\{shoot(d_1,d_2), shoot(d_2,d_1)\}, S_0\big) \wedge \\ Dead\Big(d_1, do\big(\{shoot(d_1,d_2), shoot(d_2,d_1)\}, S_0\big)\Big) \wedge \\ Dead\Big(d_2, do\big(\{shoot(d_1,d_2), shoot(d_2,d_1)\}, S_0\big)\Big) \end{gathered}$$

Now, let us consider the more complicated case where actions are temporal, *i.e.* the happening times of actions are taken into account.

First, recall that in Section 3.5.1, we introduced the function *time* to denote the happening time of an action. Now, however, since a happening is a set of actions, we need to modify the definition of *time* accordingly. Notice that the time of a concurrent happening is meaningful, only if there is at least one action in the happening, and all actions in it happen at the same time. We call this property *coherence*. Formally,

$$coherent(c) \triangleq (\exists a).a \in c \wedge (\exists t)(\forall a').[a' \in c \supset time(a') = t] \tag{3.16}$$

With this condition, the time of a concurrent happening can be defined as

$$coherent(c) \supset [time(c) = t \equiv (\exists a).(a \in c \wedge time(a) = t)]$$

Second, it is natural to extend the definition of the *start* function as

$$start\big(do(c, s)\big) = time(c) \tag{3.17}$$

and the definition for *Executable*(s) as

$$\begin{aligned} &Executable(s) \triangleq \\ &\quad (\forall c, s^*).do(c, s^*) \sqsubseteq s \supset Poss(c, s^*) \wedge start(s^*) \leq time(c) \end{aligned} \tag{3.18}$$

Finally, for the precondition of executing a concurrent set of actions, we need to further require that the actions are coherent:

$$Poss(c, s) \supset coherent(c) \wedge (\forall a).[a \in c \supset Poss(a, s)] \tag{3.19}$$

In summary, (3.10)–(3.19) form the foundational axioms for the *concurrent, temporal situation calculus.*

3.5.4 Continuous effects

So far, the effects of actions are discrete. Specifically, all numerical updates are modeled either as an assignment or as an instant increase/decrease of the fluent function. For example, in order to model the temperature of a pot of water that is being heated by a durative *boil* action, with the current model, we assign the value 100 to the temperature function at the end of the interval. In reality, however, the heating is a gradual process, and the temperature of the water increases (approximately) linearly to time.

In order to model continuous changes like this, Pinto introduced *parameters* and *names of real functions of time* to the situation calculus [Pin94]. Grosskreutz and Lakemeyer later used a similar approach under the name "continuous fluent", when defining their action language cc-Golog [GL00].

The idea is roughly like this: although the change of a numerical property is continuous, and the value of it varies within a situation, the pattern of its change is already determined at the start of the the situation. For example, suppose $temperature(water, S_0) = 20$, and in the situation $S_1 = do\big(boil(water, 5), S_0\big)$, the temperature increases with a rate of 0.5. Although the temperature of water is 20.5 at time 6 and 30 at time 25, the way it changes precisely follows the formula

$$temperature(water, S_1)[t] = 20 + 0.5 \cdot (t - 5)$$

where $temperature(water, S_1)[t]$ denotes the temperature of $water$ in situation S_1 and at time t. As a result, the numerical function $20+0.5\cdot(t-5)$ is defined as an entity, and is denoted with $linear(20, 0.5, 5)$.

Generally speaking, $linear(x_0, v_0, t_0)$ stands for a linear function $x_0 + v_0 \cdot (t - t_0)$, where t is the time at which the function value is desired. In order to obtain the function value at a specific time point, we further define an auxiliary function $eval$, which takes a continuous fluent as its first argument and the time as its second. So, we have

$$\begin{cases} temperature(water, S_1) & = \quad linear(20, 0.5, 5) \\ eval\big(temperature(water, S_1), t\big) & = \quad 20 + 0.5 \cdot (t-5) \end{cases}$$

Notice that the first equation above asserts the equality between two continuous fluents, whereas the second is between two numerical expressions.

3.5.5 Natural actions

The actions that we consider so far are deliberate, in the sense that the basic action theory only says which actions are possible in a situation, and the agent has the freedom to choose actions among all the candidates. In reality, however, many actions happen naturally regardless of the will of the agent. For example, if the agent releases a ball in the middle of the air, the ball naturally falls onto the ground. Similarly, at a certain time point, it hits the ground, and bounces back. Unlike the deliberate actions that we have considered thus far, the cause of the happening of actions like $fall$ and *bounce* is due to the laws of the nature. Now, we shall briefly investigate Pinto and Reiter's formulation of the *natural actions* in the situation calculus.

Recall that the precondition axioms in the basic action theory only serves as the necessary condition for an action to happen. That means, it only says that an action *may* happen, but not that it *must* happen. In contrast, in order to model natural actions, we need to be able to express that the action *must* happen in a certain situation.

For this purpose, we first define the predicate $Natural(a)$, which is true if and only if a is a natural action. Then, we modify the definition of the executability of a situation by extending 3.18 to incorporate the additional requirement for natural actions:

$$\begin{aligned} &Executable(s) \triangleq \\ &\qquad (\forall c, s^*).[do(c, s^*) \sqsubseteq s \supset Poss(c, s^*) \wedge start(s^*) \leq time(c)] \wedge \\ &\qquad (\forall a, c, s').[natural(a) \wedge Poss(a, s') \wedge do(c, s') \sqsubseteq s \supset \\ &\qquad\qquad\qquad\qquad\qquad\qquad a \in c \vee time(c) < time(a)] \end{aligned}$$

which, in addition to the old definition, requires that for any predecessor situation s' of s, if a is a possible natural action in s', then either a has indeed happened or there exists some other actions that has happened before the expected time of a.

As a simple example, consider the natural action $fall$, when we release a ball. We may have the following axioms for this action:

$$\left\{\begin{array}{c} natural\big(fall(t)\big) \\ Poss\big(fall(t), s\big) \equiv \neg Holding(s) \\ Poss\big(release(t), s\big) \equiv \text{TRUE} \\ Holding\big(do(a, s)\big) \equiv Holding(s) \wedge a \neq release(t) \\ Holding(S_0) \end{array}\right.$$

Then, we have

$$do\Big(\big\{fall(5)\big\}, do\big(\{release(5)\}, S_0\big)\Big)$$

is an executable situation, but

$$do\Big(\big\{fall(7)\big\}, do\big(\{release(5)\}, S_0\big)\Big)$$

is not, since in the situation $do\big(\{release(5)\}, S_0\big)$, the natural action $fall(5)$ is possible, but neither is it actually executed, nor another action happen before its expected happening time 5.

3.6 Complex Actions and Golog

The Golog language is a logic programming language for the high-level control of intelligent agents [LRL+97]. It is based on the formal basic action theory in the situation calculus, as introduced in the previous sections of this chapter. An explicit representation of the dynamics of the world is maintained in the language with axioms on preconditions, effects and the initial state, so it is possible to project the possible futures with different candidate primitive actions before committing to one of them. As a result, Golog programs enjoys a high level of abstraction, and thus much flexibility and powerfulness in reasoning about dynamic domains.

The language offers the possibility to express complex actions with basic control structures, such as **if** $\cdots$ **then** $\cdots$ and **while** $\cdots$ **do** $\cdots$, which are similar to the constructs in other programming languages. Meanwhile, it allows for the specification of nondeterministic behavior. All of these features are treated as macros which finally expand to formulas in the situation calculus.

For simplicity, we only introduce the most basic version of Golog here. The central definition is the abbreviation $Do(\delta, s, s')$, where δ is a complex action, and s and s' are situations. Intuitively, $Do(\delta, s, s')$ holds if it is possible to execute δ in situation s, resulting in the situation s'. The formal definition is an inductive one on the structure of δ:

1. *Primitive actions*

$$Do(a, s, s') \triangleq Poss(a[s], s) \land s' = do(a[s], s)$$

   where $a[s]$ is the formula obtained by restoring the situation arguments s for all fluents in a.

2. *Test action*

$$Do(\phi?, s, s') \triangleq \phi[s] \land s = s'$$

3. *Sequence*

$$Do\big([\delta_1; \delta_2], s, s'\big) \triangleq (\exists s^*).\big(Do(\delta_1, s, s^*) \land Do(\delta_2, s^*, s')\big)$$

4. *Non-deterministic choice between two actions*

$$Do\big((\delta_1|\delta_2), s, s'\big) \triangleq Do(\delta_1, s, s') \lor Do(\delta_2, s, s')$$

5. *Non-deterministic choice of action arguments*

$$Do\big((\pi x)\delta(x), s, s'\big) \triangleq (\exists x).Do\big(\delta(x), s, s'\big)$$

6. *Non-deterministic iteration*

$$\begin{aligned}&Do(\delta^*, s, s') \triangleq\\ &\quad (\forall P).\big\{(\forall s_1).P(s_1, s_1)\land\\ &\quad (\forall s_1, s_2, s_3).[P(s_1, s_2) \land Do(\delta, s_2, s_3) \supset P(s_1, s_3)]\big\}\\ &\qquad\qquad\qquad \supset P(s, s')\end{aligned}$$

7. *Procedure call*

$$Do\big(P(t_1, \cdots, t_n), s, s'\big) \triangleq P(t_1[s], \cdots, t_n[s], s, s')$$

Based on these constructs, we can further define the abbreviation of the following common control structures

$$\begin{aligned}
&Do(\mathbf{if}\,\phi\,\mathbf{then}\,\delta_1\,\mathbf{else}\,\delta_2\,\mathbf{endIf}, s, s') \triangleq\\
&\qquad Do([\phi?;\delta_1]|[\neg\phi?;\delta_2], s, s')\\
&Do(\mathbf{while}\,\phi\,\mathbf{do}\,\delta\,\mathbf{endWhile}, s, s') \triangleq\\
&\qquad Do([[\phi?;\delta]^*;\neg\phi?], s, s')\\
&Do([\mathbf{proc}\,P_1(\vec{v_1})\delta_1\,\mathbf{endProc}, \cdots, \mathbf{proc}\,P_n(\vec{v_n})\delta_n\,\mathbf{endProc},], s, s') \triangleq\\
&\qquad (\forall P_1, \cdots, P_n).[\bigwedge_{i=1}^{n}(\forall s_1, s_2, \vec{v_i}).Do(\delta_i, s_1, s_2) \supset Do(P_i(\vec{v_i}), s_1, s_2)]\\
&\qquad\qquad Do(\delta_0, s, s')
\end{aligned}$$

As a small example, suppose we want to write a program to empty a briefcase, then we may define the following procedure:

```
proc $emptyBriefcase$
  while $\exists x.In(x)$ do
    $\pi x.In(x)?; takeOut(x)$
  endWhile
endProc
```

Chapter 4

The Logic $\mathcal{ES}$

The logic $\mathcal{ES}$, introduced by Lakemeyer and Levesque [LL04], is a dialect of the situation calculus. While it reasons about situations, it does not explicitly mention a situation term in the language. Instead, the future situations are referred to with the help of modal operators in the language, and their meanings are defined in the semantics. One consequence is that $\mathcal{ES}$ is more succinct, and simplifies the proofs of theorems in the situation calculus. Meanwhile, as shown in [LL05], the simplification does not lead to a loss of expressiveness in that the second order extension of $\mathcal{ES}$ captures the non-epistemic fragment of the situation calculus.

In Section 4.1, we first introduce the syntax and semantics of $\mathcal{ES}$. Then, the development of the following sections resembles what we did in the previous chapter. Section 4.2 presents the basic action theories, and Sections 4.3 and 4.4 discusses the regression and progression, respectively, in the logic $\mathcal{ES}$.

The standard version of $\mathcal{ES}$ does not consider time or concurrency, so we postpone the discussion on how to rebuild these features to Chapter 5.

4.1 The Language

The language that we introduce here is a subset of the general one defined in [LL05]. Compared with their definition, we do not consider second-order variables or the modal operators for denoting the epistemic state.

4.1.1 The Alphabet

The language of $\mathcal{ES}$ consists of formulas over symbols from the following vocabulary:

- First order variables: $x_1, x_2, \cdots, y_1, y_2, \cdots, a_1, a_2, \cdots$;

- Fluent function symbols of arity k: $f_1^k, f_2^k, \cdots$, for example $temperature$, $bestAction$;
- Rigid function symbols of arity k: $g_1^k, g_2^k, \cdots$, for example $fatherOf$, $moveTo$;
- Fluent predicate symbols of arity k: $F_1^k, F_2^k, \cdots$, for example $AtHome$, $Happy$;
- Rigid predicate symbols of arity k: $G_1^k, G_2^k, \cdots$, for example $Human$, $Fragile$;
- Connectives and other symbols: $=$, $\wedge$, $\neg$, $\forall$, $\Box$, round and square parentheses, period, comma.

Here, variables and function symbols have two sorts, *action* for actions (like $bestAction$ and $moveTo$) and *object* for everything else (like $temperature$ and $fatherOf$). Instead of distinguishing them in their possible roles in the semantics, we lump them together, and allow to use any term as an action or an object.

4.1.2 Terms and Formulas

The *terms* of the language are of sort *action* or *object*, and form the least set of expressions such that

1. Every first-order variable is a term of the corresponding sort;
2. If $t_1, \cdots, t_k$ are terms and h is a k-ary (rigid or fluent) function symbol, then $h(t_1, \cdots, t_k)$ is a term of the same sort as h.

A *primitive* term is one with the form $h(n_1, \cdots, n_k)$, where h is a function symbol of arity k, and all of the n_i are ground terms. We let $\mathcal{N}$ denote the set of all ground terms, and $\mathcal{Z}$ denote the set of all sequences of ground action terms, including $\langle\rangle$, the empty sequence.

The *well-formed formulas* of the language form the least set such that

1. If $t_1, \cdots, t_k$ are terms, and H is a k-ary (rigid or fluent)predicate symbol, then $H(t_1, \cdots, t_k)$ is an (atomic) formula;
2. If t_1 and t_2 are terms, then $(t_1 = t_2)$ is a formula;
3. If t is an (action) term and α is a formula, then $[t]\alpha$ is a formula;
4. If α and β are formulas, and v is a first-order variable, then $(\alpha \wedge \beta)$, $\neg\alpha$, $\forall x.\alpha$, $\Box\alpha$ are also formulas.

We read $[t]\alpha$ as "α holds after action t", and $\Box\alpha$ as "α holds after any sequence of actions". As usual, we treat $(\alpha \vee \beta)$, $(\alpha \supset \beta)$, $(\alpha \equiv \beta)$ and $\exists x.\alpha$ as abbreviations. A *sentence* is a formula without free variables, and a *primitive formula* is one with the form $H(n_1, \cdots, n_k)$, where H is a predicate symbol of arity k, and all of the n_i are ground terms. Besides, a formula without $\Box$ is called *bounded*, one without $\Box$ or $[t]$ is called *static*, and one without $\Box$, $[t]$ or $Poss$ is called a *fluent formula*.

As an example of formulas in the logic $\mathcal{ES}$, consider the following sentence:[1]

$$\forall x.Fragile(x) \supset \big(\Box[drop(x)]Broken(x)\big) \tag{4.1}$$

This is a well-formed formula in $\mathcal{ES}$, which may intuitively read as "for anything that is fragile, if we drop it in any situation, it gets broken". Notice that here, we do not have any situation term, and fluents do not have an extra situation argument. Nevertheless, it is possible to reason about the dynamical changes in the world with the help of the $\Box$ and $[\,]$ operators. If we translate Equation (4.1) to the situation calculus, we get the sentence

$$(\forall x).\Bigg(Fragile(x) \supset (\forall s).Broken\Big(x, do\big(drop(x), s\big)\Big)\Bigg)$$

4.1.3 The Semantics

In order to determine whether a sentence α is true or not after a sequence of actions z has been performed, we need to specify the world w in which the evaluation is performed. Formally, we write it as

$$w, z \models \alpha$$

The world w here determines both the values of primitive functions and the truths for primitive sentences, not only initially, but also after any sequence of actions. More precisely, a world $w \in W$ is any function from primitive terms and $\mathcal{Z}$ to $\mathcal{N}$, and from primitive sentences and $\mathcal{Z}$ to $\{0, 1\}$, satisfying the rigidity constraint: if ϕ is a rigid function or predicate symbol, then $w[\phi(n_1, \cdots, n_k), z] = w[\phi(n_1, \cdots, n_k), z']$, for all z and z' in $\mathcal{Z}$.

The identity of t given w and z, denoted with $|t|_w^z$, is defined inductively by

1. If $t \in \mathcal{N}$, then $|t|_w^z = t$;
2. $|h(t_1, \cdots, t_k)|_w^z = w[h(n_1, \cdots, n_k), z]$, where $n_i = |t_i|_w^z$.

[1] We assume that logical connectives have higher priority than $\Box$, but lower priority than $[\,]$

For the truth value of well-formed formulas, given a world $w \in W$ and an action sequence $z \in \mathcal{Z}$, we define $w, z \models \alpha$ (read: α is true after z in w) as

1. $w, z \models H(t_1, \cdots, t_k)$ iff $w[H(n_1, \cdots, n_k), z] = 1$, where H is a k-ary predicate symbol and $n_i = |t_i|_w^z$;
2. $w, z \models (t_1 = t_2)$ iff n_1 and n_2 are identical, where $n_i = |t_i|_w^z$;
3. $w, z \models [t]\alpha$ iff $w, z \cdot n \models \alpha$, where $n = |t|_w^z$;
4. $w, z \models (\alpha \wedge \beta)$ iff $w, z \models \alpha$ and $w, z \models \beta$;
5. $w, z \models \neg\alpha$ iff $w, z \not\models \alpha$;
6. $w, z \models \forall x.\alpha$ iff $w, z \models \alpha_n^x$ for every ground term n (of the same sort as x);
7. $w, z \models \Box\alpha$ iff $w, z \cdot z' \models \alpha$ for every $z' \in \mathcal{Z}$.

We write $w \models \alpha$ to mean $w, \langle\rangle \models \alpha$, and $\Sigma \models \alpha$, where Σ is a set of sentences and α is a sentence, to mean that Σ logically entails α, *i.e.* for all world w, if $w \models \alpha'$ for every $\alpha' \in \Sigma$, then $w \models \alpha$. Finally, $\models \alpha$ means α is valid, *i.e.* $\{\} \models \alpha$.

4.2 Basic Action Theories

The basic action theory in $\mathcal{ES}$ resembles that in the situation calculus, except for the following major differences. First, since the language does not have situation terms, and the structure of situations is defined in the semantics, there is no need to have the foundational axioms $\mathcal{FA}$. Second, we assume a fixed domain of discourse, and the unique names assumption on ground terms is a built-in property.

As a result, the basic action theory in the logic $\mathcal{ES}$ has the following form:

$$\Sigma = \Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \tag{4.2}$$

where

1. Σ_0, the initial theory, is any set of fluent sentences, expressing the facts in the initial situation, *e.g.* what relations are true and what are the initial values of functions;
2. Σ_{pre}, the precondition axiom, is a singleton sentence of the form

$$\Box Poss(a) \equiv \pi$$

where π is a fluent formula, which has the form of a big disjunction specifying the preconditions for each action;

3. Σ_{post}, the successor state axiom, is a set of sentences for each of the fluent symbols in the domain, either of the form $\Box[a]F(\vec{x}) \equiv \gamma_F$ for predicates, or of the form $\Box[a]f(\vec{x}) = y \equiv \gamma_f$ for functions, where γ_F and γ_f are both fluent formulas.

As an example, consider the vehicle domain, where a car drives from one place to another. In this domain, we may have the following basic action theory:

$$\begin{cases} \Sigma_0 & = \left\{At(berlin)\right\} \\ \Sigma_{pre} & = \left\{Poss(a) \equiv a = drive(x_1, x_2) \wedge At(x_1) \wedge x_1 \neq x_2\right\} \\ \Sigma_{post} & = \left\{ \begin{array}{ll} \Box[a]At(x) \equiv & \exists x'.a = drive(x', x) \vee \\ & At(x) \wedge \neg\exists x''.a = drive(x, x'') \end{array} \right\} \end{cases} \tag{4.3}$$

Then, we may have, for instance

$$[drive(berlin, aachen)][drive(aachen, paris)]At(paris)$$

4.3 Regression

Like in the situation calculus, in order to solve the *projection problem*, either regression or progression has to be utilized. Now, we start from the former case.

In the logic $\mathcal{ES}$ as we define here, any bounded sentence is regressible. For the actual regression, we define $\mathcal{R}[\alpha]$, the regression of α with respect to Σ, to be the fluent formula $\mathcal{R}[\langle\rangle, \alpha]$, where for any sequence of actions $z \in \mathcal{Z}$, $\mathcal{R}[z, \alpha]$ is defined inductively on the structure of α by[2]

1. $\mathcal{R}[z, (t_1 = t_2)] = (t_1 = t_2)$;
2. $\mathcal{R}[z, \forall x.\alpha] = \forall x.\mathcal{R}[z, \alpha]$;
3. $\mathcal{R}[z, (\alpha \wedge \beta)] = (\mathcal{R}[z, \alpha] \wedge \mathcal{R}[z, \beta])$;
4. $\mathcal{R}[z, \neg\alpha] = \neg\mathcal{R}[z, \alpha]$;
5. $\mathcal{R}\big[z, [t]\alpha\big] = \mathcal{R}[z \cdot t, \alpha]$;
6. $\mathcal{R}[z, Poss(t)] = \mathcal{R}[z, \pi^a_t]$;

[2] For simplicity, we only define the regression without functional fluents. The case where functional fluents do exist is similar to that in the situation calculus, and thus omitted here.

7. $\mathcal{R}[z, F(t_1, \cdots, t_k)]$ is defined inductively on z by

   a) $\mathcal{R}[\langle\rangle, F(t_1, \cdots, t_k)] = F(t_1, \cdots, t_k)$;

   b) $\mathcal{R}[z \cdot t, F(t_1, \cdots, t_k)] = \mathcal{R}[z, (\gamma_F)^{a\, x_1 \cdots x_k}_{t\, t_1 \cdots t_k}]$

Note that this definition uses the right-hand sides of both the precondition and successor state axioms from Σ.

With the definition of regression above, Lakemeyer and Levesque further proved the following *regression theorem*.

Theorem 4.1 (The regression theorem in $\mathcal{ES}$)**.** *Let $\Sigma = \Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post}$ be a basic action theory and let α be a bounded sentence. Then $\mathcal{R}[\alpha]$ is a fluent sentence and satisfies*

$$\Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \models \alpha \quad \textit{iff} \quad \Sigma_0 \models \mathcal{R}[\alpha]$$

4.4 Progression

Progression in $\mathcal{ES}$ is very similar to that in the situation calculus, but due to the fact that no situation term exists in the syntax of $\mathcal{ES}$, the formulation is simpler than Reiter's original definition. The following definition of *progression* is given by Claßen and Lakemeyer [CELN07].

Definition 4.2 (Progression)**.** A set of sentences Σ_t is a *progression* of Σ_0 through a ground (action) term t (wrt. Σ_{pre} and Σ_{post}) iff

1. all sentences in Σ_t is in $\langle t\rangle$ (*i.e.* equivalent to $[t]\varphi$ for some fluent formula φ);

2. $\Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \models \Sigma_t$;

3. for every world w_t with $w_t \models \Sigma_t \cup \Sigma_{pre} \cup \Sigma_{post}$, there is a world w with $w \models \Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post}$ such that

$$w_t, t \cdot z \models \phi(\vec{o}) \text{ iff } w, t \cdot z \models \phi(\vec{o})$$

   for all $z \in \mathcal{Z}$ and all primitive formulas $\phi(\vec{o})$.

Due to the similarity between the progression in the situation calculus and the one in the logic $\mathcal{ES}$, we do not repeat the discussion of the properties of progression in $\mathcal{ES}$. We advise the reader to refer to Section 3.4 for details.

Chapter 5

The Extensions to $\mathcal{ES}$

So far, the logic $\mathcal{ES}$ only concerns with discrete objects and sequential actions. No numerical or temporal properties have been taken into account. However, they are necessary for defining the declarative semantics for the latest versions of PDDL. Therefore, we explore in this section a possible way to extend $\mathcal{ES}$ with numbers, time, concurrency, *etc.* As we shall see, this is done mainly by adding new axioms to the basic action theory previously defined in Equation (4.2) of Chapter 4.

First of all, we introduce a minor modification in Section 5.1, which enables us to reason about the executability of action sequences. This is followed by a discussion on numbers and numerical expressions, in Section 5.2, which is, in turn, a foundation for the temporal extension described in Section 5.3. In Section 5.4, we introduce durative actions, and see how concurrent processes can be modeled with our extensions. We then investigate continuous changes in Section 5.5. Finally, Section 5.6 discusses how to model coercive actions in $\mathcal{ES}$.

5.1 Executability

On many occasions, it is useful to reason about the executability of action sequences. Recall that in the situation calculus, we use the relation $\mathit{Executable}(s)$, as defined in (3.6), to check whether s is an executable situation or not. However, in the logic $\mathcal{ES}$, there is no situation term in the language, so it is not possible to define the predicate *Executable* in the old fashion. To solve this problem, we introduce a new axiom Σ_{exec} to the basic action theory, when we need to reason whether action sequences are executable in a domain.

Σ_{exec} uses a 0-ary predicate symbol *Executable*, which is initially true. In the succeeding situations, we modify the truth value of *Executable* according to its old value and to whether the latest action is a possible one. Formally, Σ_{exec} is the following sentence:

$$Executable \wedge \big(\Box[a]Executable \equiv Executable \wedge Poss(a)\big) \tag{5.1}$$

(5.1) essentially asserts two things. First, the static formula *Executable*, which resembles sentences in the initial theory, says that the initial situation is executable. Second, $\Box[a]Executable \equiv Executable \wedge Poss(a)$ has the form of a successor state axiom, which says that in any situation, *Executable* will be true after action a if and only if it is true and the execution of a is possible.

As an example, let us consider the example in (4.3) again. With the definition of Σ_{exec} above, we have

$$\begin{aligned}&\Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_{exec} \models\\ &\quad [drive(berlin, aachen)][drive(aachen, paris)]Executable\end{aligned}$$

but

$$\begin{aligned}&\Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_{exec} \models\\ &\quad [drive(berlin, aachen)][drive(li\grave{e}ge, paris)]\neg Executable\end{aligned}$$

since after $drive(berlin, aachen)$, $At(li\grave{e}ge)$ is false, so the precondition of $drive(li\grave{e}ge, paris)$ is not satisfied, and therefore the action sequence is not executable.

5.2 Numerics

In Section 3.5, we have seen how numbers, their operations and their relations are used in the situation calculus to represent properties like the happening time of actions. An interesting question is whether we can do the similar thing in the logic $\mathcal{ES}$. Although $\mathcal{ES}$, in its current form, is similar to the situation calculus in many respects, it has so far been unclear how we may use numbers in it.

In this section, we shall look into this problem. In particular, we would like to allow for elementary numerical expressions in the logic, so that we may express basic numerical properties, such as the sentence

$$\exists v. fuel\_left(v) = 75 \wedge \exists dest.[drive(v, paris, dest)]fuel\_left(v) = 15$$

For this purpose, we first have to allow for numbers in our logical domain. Unlike Reiter's and other formalisms, where the real numbers with their "standard interpretation" are used directly, we try to be more precise, and start with the axiomatization of numbers.

There are several ways to axiomatize real numbers, among which the most frequently used is based on the Zermelo-Fraenkel set theory. According to this approach, the real number system is a Dedekind-complete ordered field. That is to say, a model for the real number system consists of a set $\mathbb{R}$, two elements of $\mathbb{R}$ (0 and 1), two binary functions on $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$ ($+$ and $\cdot$), and a binary relation on $\mathbb{R} \times \mathbb{R}$ ($\leq$), satisfying the following properties

$$(\forall x, y, z \in \mathbb{R}).\ (x + y) + z = x + (y + z) \tag{5.2}$$

$$(\exists 0 \in \mathbb{R}. \forall x \in \mathbb{R}).\ x + 0 = x \tag{5.3}$$

$$(\forall x \in \mathbb{R}. \exists y \in \mathbb{R}).\ x + y = 0 \tag{5.4}$$

$$(\forall x, y \in \mathbb{R}).\ x + y = y + x \tag{5.5}$$

$$(\forall x, y, z \in \mathbb{R}).\ (x \cdot y) \cdot z = x \cdot (y \cdot z) \tag{5.6}$$

$$(\exists 1 \in \mathbb{R}. \forall x \in \mathbb{R}).\ x \cdot 1 = x \tag{5.7}$$

$$\big(\forall x \in (\mathbb{R} \setminus \{0\}). \exists y \in \mathbb{R}\big).\ x \cdot y = 1 \tag{5.8}$$

$$(\forall x, y \in \mathbb{R}).\ x \cdot y = y \cdot x \tag{5.9}$$

$$(\forall x, y, z \in \mathbb{R}).\ (x + y) \cdot z = x \cdot z + y \cdot z \tag{5.10}$$

$$0 \neq 1 \tag{5.11}$$

$$(\forall x \in \mathbb{R}).\ x \leq x \tag{5.12}$$

$$(\forall x, y \in \mathbb{R}).\ x \leq y \wedge y \leq x \Rightarrow x = y \tag{5.13}$$

$$(\forall x, y, z \in \mathbb{R}).\ x < y \wedge y < z \Rightarrow x < z \tag{5.14}$$

$$(\forall x, y \in \mathbb{R}).\ x \leq y \vee y \leq x \tag{5.15}$$

$$(\forall x, y, z \in \mathbb{R}).\ x < y \Rightarrow x + z < y + z \tag{5.16}$$

$$(\forall x, y, z \in \mathbb{R}).\ x < y \wedge 0 < z \Rightarrow x \cdot z < y \cdot z \tag{5.17}$$

$$\begin{aligned}&(\forall S \subset \mathbb{R}).(S \neq \emptyset \wedge \exists y \in \mathbb{R}, \forall x \in S. x \leq y) \Longrightarrow \\ &\qquad (\exists z \in \mathbb{R}. \forall x \in S. x \leq z \wedge \forall y \in \mathbb{R}.\ (\forall x \in S. x \leq y) \Rightarrow z \leq y)\end{aligned} \tag{5.18}$$

Here, (5.2) – (5.11) say that $(\mathbb{R}, +, \cdot)$ forms a field; (5.12) – (5.15) say that $(\mathbb{R}, \leq)$ is a total order; (5.16) and (5.17) say that the field operations $+$ and $\cdot$ are compatible with the order $\leq$; (5.18) says that $\mathbb{R}$ is Dedekind complete, *i.e.* every subset of $\mathbb{R}$ that has an upper bound has a least upper bound in $\mathbb{R}$.

Axioms (5.2) – (5.18) characterize all the intrinsic properties of real numbers. Other properties can be derived from these axioms. Moreover, it can be shown that there exists exactly one model for the above axioms up to isomorphism, which means, given any two Dedekind-complete ordered fields $\mathbb{R}_1$ and $\mathbb{R}_2$, we can consider them as the same mathematical object, with just renaming and relabeling.

However, two problems would arise if we were to use this axiomatization in $\mathcal{ES}$. First, the set of real numbers, $\mathbb{R}$, is uncountable [Can91], whereas $\mathcal{ES}$ assumes a countably infinite domain. Including an uncountable set in the domain will lose the nice properties of $\mathcal{ES}$, such as the substitutional interpretation of quantifiers and the representation theorem. Second, Axiom (5.18) quantifies over subsets of $\mathbb{R}$, which means that this sentence cannot be expressed in first order logic. On the other hand, there is no complete proof theory for second order logic, so allowing for general second-order quantification would make the logic even not recursively enumerable.

To solve these difficulties, let us first have a closer look at the second problem. Here, what Axiom (5.18) asserts, in fact, is the continuity of real numbers. This is an important property for calculus and real analysis, but somewhat irrelevant to our goal of doing arithmetics in the logic. As a result, we try dropping this last axiom, and consider the theory of an ordered field, described by Axioms (5.2)-(5.17) only.

Unlike the original theory, which has exactly one model, the theory of an ordered field has several models. To see why, note that both the rational numbers ($\mathbb{Q}$) and the real numbers ($\mathbb{R}$) are ordered fields, but they are obviously not isomorphic, since they have distinct cardinality. It can also be shown that the algebraic numbers ($\mathbb{A}$), the computable numbers ($\mathbb{C}$) and the definable numbers ($\mathbb{D}$) are all ordered fields.

Here, an *algebraic number* is a real number that is a root of a non-zero polynomial with integer coefficients, such as $\frac{3}{7}$ and $\sqrt{2}$, but not π; a *computable number* is a real number that can be computed to any desired precision by a finite, terminating algorithm, such as $3\sqrt[3]{2}$, π and e; a real number, a, is first-order *definable* in the language of set theory, without parameters, if there is a formula ψ in the language of set theory, with one free variable, such that a is the unique real number such that $\psi(a)$ holds.[1] Indeed, $\mathbb{Q}$, $\mathbb{A}$, $\mathbb{C}$, $\mathbb{D}$ and $\mathbb{R}$, are increasingly general concepts, *i.e.* $\mathbb{Q} \subset \mathbb{A} \subset \mathbb{C} \subset \mathbb{D} \subset \mathbb{R}$. Moreover, $\mathbb{Q}$, $\mathbb{A}$, $\mathbb{C}$ and $\mathbb{D}$ are all countable, and thus contain far "fewer" elements than $\mathbb{R}$.

Among the countable ones, the set of particular interest for us is that of the computable numbers, since it contains all the specific numbers that can ever be represented by algorithms. As a result, it is a broad enough subset of the real numbers for our purpose with the desired property of countability.

[1]All three definitions are from www.wikipedia.org.

Compared with the reals, the weaker axioms on the computable numbers cannot capture the property of continuity, so it is unknown whether analysis (including calculus) can be reconstructed with the weaker axioms. However, as mentioned above, we are only concerned with the elementary operations, and this axiomatization is powerful enough for this purpose. Furthermore, Tarski proved in 1951 that elementary algebra[2] is decidable, by showing that every formula can be reduced to an equivalent one without quantifiers [Tar51]. This result offers us the freedom to write arbitrarily complex arithmetic formulas in elementary algebra, and it is guaranteed that the truth value of the formula is computable.

As a result, our solution to the two problems above is to include the countably infinite set of computable numbers in the domain of $\mathcal{ES}$[3], allowing for elementary operations ($+$, $\times$ and $<$) on them, and axiomatize them with Axioms (5.2)-(5.17). In the rest part of this paper, we use Σ_{num} to stand for the numerical axioms for an ordered field.

Finally, in order to identify the numbers in the domain, we define a predicate $Number : \mathcal{N} \rightarrow \{\textsc{true}, \textsc{false}\}$, whose only positive instances are all the numbers in the domain, *i.e.* $Number(x)$ is true if and only if x is a number.

5.2.1 An example

In this subsection, we analyze a simple example taken from Figure 4 of [FL03].

Suppose that we have a few jugs that can serve as water containers. One can pour water from one jug to another as long as the destination does not overflow. Two properties are associated with each jug, namely, the *capacity* and the current *amount* of water in the jug.

In order to model this domain, we use two functions symbols $amount(j)$ and $capacity(j)$ and an action $pour(j_1, j_2)$, with their obvious meanings. Now we can write the basic action theory as follows

The precondition axiom Σ_{pre}

$$\begin{aligned}&\Box Poss(a) \equiv \\ &\quad a = pour(j_1, j_2) \land amount(j_1) \leq capacity(j_2) - amount(j_2)\end{aligned}$$

[2]Elementary algebra concerns with the properties of real numbers restricted to operations $+$ and $\times$, using only the constants 0 and 1, and the predicate $\leq$.

[3]Strictly speaking, we should use the symbol $\mathbb{C}$ for the set of all the numbers in the domain. However, following the convention, we sometimes also use the symbol $\mathbb{R}$, since all the real numbers that can be represented by algorithms are computable, anyway.

The successor state axioms Σ_{post}

$$\begin{aligned}
&\Box[a]capacity(j) = x \equiv capacity(j) = x\\
&\Box[a]amount(j) = x \equiv\\
&\quad \exists j'.a = pour(j, j') \wedge x = 0 \vee\\
&\quad \exists j'.a = pour(j', j) \wedge x = capacity(j) + capacity(j') \vee\\
&\quad \forall j'.a \neq pour(j, j') \wedge a \neq pour(j', j) \wedge amount(j) = x
\end{aligned}$$

The initial description Σ_0

$$\begin{aligned}
capacity(jug_1) &= 10\\
capacity(jug_2) &= 5\\
amount(jug_1) &= 7\\
amount(jug_2) &= 2
\end{aligned}$$

With the basic action theory above, we may conclude, for example,

$$\Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_{num} \models \neg Poss(pour(jug_1, jug_2))$$

and

$$\begin{aligned}
&\Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_{num} \models\\
&\qquad [pour(jug_2, jug_1)](amount(jug_1) = 9 \wedge amount(jug_2) = 0)
\end{aligned}$$

5.3 Temporal Extension

So far, the logic is atemporal. That is, we only consider the sequence of actions, and disregard the temporal properties, like when an action happens or how long the duration of an action is. With the numerical extension in the previous section, we are now able to rescue this by adding the happening time of actions in the basic action theory.

Basically, we follow Reiter's approach in the situation calculus as described in Section 3.5, and add a time argument to each action. For example, $A(\vec{x})$ is now extended to $A(\vec{x}, t)$, to mean that action $A(\vec{x})$ happens at time t. The advantage is that we can stick to the standard syntax and semantics of $\mathcal{ES}$.

Strictly speaking, $A(\vec{x}, t)$ is not an action, but instead a *happenings*, since it consist of an action $A(\vec{x})$ and the corresponding time t at which it happens. However, we use the term "action" and "happening" interchangeably for convenience, whenever there is no confusion.

The happening time of an action is thus always the last argument in its parameter list. Unfortunately, this fact cannot be expressed in the language. As a result, we define an auxiliary function $time : \mathcal{N} \rightarrow \mathbb{R}$ as

$$\begin{aligned} \forall a, t. \Box time(a) = t \equiv \\ \exists \vec{x}_1 . a = A_1(\vec{x}_1, t) \vee \\ \cdots\cdots \vee \\ \exists \vec{x}_m . a = A_m(\vec{x}_m, t) \end{aligned} \tag{5.19}$$

where $A_1, \cdots, A_m$ are all the action symbols in the domain. With this definition, $time(a)$ always has the value of the happening time of action a.

In order to have the "current" time, we further define a 0-ary fluent function *now*, whose value is always the happening time of the most recent action. Formally, we have

$$\Box[a] now = time(a) \tag{5.20}$$

This has the effect that *now* always refers to the time at which the current situation starts. The functionality of *now* is similar to $start(s)$ in Reiter's formalism, where s is the current situation. Notice that our view of time is point based, and we only update the value of "*now*" at the points where some action happens.

One important application of *now* is to ensure the correct temporal ordering of actions. When we write $[a_1][a_2]\alpha$, we assume that a_2 happens after (or, at earliest, simultaneously with) a_1. It makes little sense, if any, to write $[a_1]$ before $[a_2]$, but the happening time of a_1 is actually later than a_2. In the temporal situation calculus, this paradox is avoided by the clause

$$do(a, s^*) \sqsubseteq s \supset start(s^*) \leq time(a)$$

in the $Executable(s)$ predicate, where $start(s^*)$ is the time when situation s^* starts. Here, however, it is difficult to define such an *Executable* predicate in $\mathcal{ES}$, since it would then have to reason about the start times of more than one situation without resorting explicitly to any situation term. Fortunately, with the help of *now*, we may add the following axiom, saying that the happening time of an action must not be earlier than the start time of the current situation. Formally,

$$\Box Poss(a) \supset now \leq time(a) \tag{5.21}$$

This formula, together with the ones in (5.19) and (5.20), forms the three temporal axioms in $\mathcal{ES}$, which we denote with Σ_{time}[4].

[4] Strictly speaking, (5.21) itself is not an independent axiom, but is instead compiled into

Notice that in (5.21), $time(a) = now$ is true only when the action a happens "simultaneously" with its predecessor. In this way, concurrent happening of more than one event becomes possible. From the introduction in Section 3.5.3, we know that this is the model of interleaved concurrency, In this thesis, we shall mainly stick to the interleaved account to model concurrency, due to its simplicity. In Chapter 8, we shall discuss the limitations of interleaved concurrency and the prospect of integrating true concurrency in $\mathcal{ES}$.

5.4 Durative Actions and Concurrency

Up to now, all the actions we express in $\mathcal{ES}$ are duration-free, *i.e.* an action is an instantaneous event that changes a situation and finishes immediately after activation. However, in practice, most actions have a duration. In Section 3.5.2, we have already talked about how durative actions may be modeled in the situation calculus. Now, we focus on the same topic in the logic $\mathcal{ES}$.

Following Pinto and Reiter's approach, we represent a durative action by splitting it into two instantaneous actions denoting the start and end events of it, along with a predicate denoting whether the durative action is in progress [Pin94, Rei96]. However, instead of doing so literally, where a durative action $walk(x, y)$, for example, is split into $startWalk(x, y, t)$, $endWalk(x, y, t)$ and $Walking(x, y, s)$, we treat durative actions as domain elements in $\mathcal{ES}$, and have two function symbols $start, end : \mathcal{N} \times \mathbb{R} \to \mathcal{N}$, and a predicate symbol $Performing : \mathcal{N} \to \{\text{FALSE}, \text{TRUE}\}$ for similar purposes. The "walking" example above is thus represented by $start\big(walk(x, y), t\big)$, $end\big(walk(x, y), t\big)$ and $Performing\big(walk(x, y)\big)$ in our formalism. The advantage is that we can more easily explore the correspondence among the three when necessary. For example, the following properties may be conveniently expressed in $\mathcal{ES}$

$$\begin{aligned}\Box[a]Performing(\widetilde{a}) \equiv \exists t.a = start(\widetilde{a}, t)\vee \\ Performing(\widetilde{a}) \wedge \neg\exists t'.a = end(\widetilde{a}, t')\end{aligned} \tag{5.22}$$

$$\Box Poss(start(\widetilde{a}, t)) \supset \neg Performing(\widetilde{a}) \tag{5.23}$$

$$\Box Poss(end(\widetilde{a}, t)) \supset Performing(\widetilde{a}) \tag{5.24}$$

the precondition axiom when we concatenate Σ_{time} with the original basic action theory. For example, $\Sigma_{pre} \cup \Sigma_{time}$ entails the formula

$$\Box Poss(a) \equiv now \leq time(a) \wedge \pi$$

where π is the right hand side of the original Σ_{pre}. This is also the case when we later define Σ_{dura}, *etc.*

whereas we have to write three sentences for each action if Pinto and Reiter's approach is used. In fact, (5.22)–(5.24) are the first three of the durative-action axioms Σ_{dura}.

With this representation of durative actions, arbitrary overlapping actions (both durative and non-durative) can be expressed. For example, the concurrent happening of actions in Figure 3.1 can be captured by the following action sequence:

$$\begin{aligned}&[start(walk(x,y),t_1)][start(chew(gum),t_1)]\\&\quad[end(chew(gum),t_2)][start(sing(song),t_3)]\\&\quad[shoot(t_4)][end(walk(x,y),t_5)][end(sing(song),t_6)]\end{aligned}$$

In some cases, it is necessary to reason about the duration of an action. For this purpose, we record the start time of the durative action instance, and obtain the duration when its end event is activated. We do so by introducing a new function symbol $since : \mathcal{N} \rightarrow \mathbb{R}$, where $since(\widetilde{a}) = t$ means that the durative action $\widetilde{a}$ has been executing since time t. Formally, the correct value of $since(\widetilde{a})$ is correctly guaranteed by

$$\begin{aligned}\Box[a]&since(\widetilde{a}) = t \equiv\\&a = start(\widetilde{a},t)\vee\\&since(\widetilde{a}) = t \wedge \neg\exists t'.a = end(\widetilde{a},t')\end{aligned} \tag{5.25}$$

(5.25) serves as the fourth and last axiom in Σ_{dura}. With this axiom, when $end(a,t)$ is executed, the duration of action a can be simply obtained by $\big(t - since(a)\big)$. Notice that the value of the $since(a)$ function is meaningful only if $\widetilde{a}$ is in being executed (*i.e.* $Performing(a)$ is true).

Now, let us look at a producer-consumer example which involves the features mentioned in this section so far. We assume that there are two durative actions, $produce(v)$ and $consume(x,v)$. The former increases the amount of the product at rate v and the latter decreases it by x at rate v. Figure 5.1 illustrates the basic action theory of this domain.[5]

Notice that a conservative resource model is used here, *i.e.* we check whether there is enough product to consume at the start point of the *consume* action, and increment the quantity of the product only at the end of the *produce* action. This ensures the value of *quantity* never becomes negative.

Notice also that in the precondition axiom (5.26), we assert

$$t - since\big(consume(x,v)\big) = x/v$$

[5] As mentioned before, axioms like Σ_{exec}, Σ_{time} and Σ_{dura} only appear conceptually in the definitions. In practice, they are compiled into Σ_{pre} or Σ_{post}.

Precondition axiom Σ_{pre}

$$\begin{aligned}
&\Box Poss(a) \equiv now \leq time(a) \land \Big(\\
&\quad \exists x, v, t.\ a = start(consume(x, v), t) \land \\
&\qquad \neg Performing(consume(x, v)) \land x \leq quantity \lor \\
&\quad \exists x, v, t.\ a = end(consume(x, v), t) \land Performing(consume(x, v)) \land \\
&\qquad \big(t - since(consume(x, v))\big) = x/v \lor \\
&\quad \exists v, t.\ a = start(produce(v), t) \land \neg Performing(produce(v)) \lor \\
&\quad \exists v, t.\ a = end(produce(v), t) \land Performing(produce(v))\Big)
\end{aligned} \tag{5.26}$$

Successor state axioms Σ_{post}

$$\begin{aligned}
&\Box[r]quantity = y \equiv \\
&\quad \exists x, v, t. r = start(consume(x, v), t) \land y = quantity - x \lor \\
&\quad \exists v, t. r = end(produce(v), t) \land \\
&\qquad y = quantity + v \cdot \big(t - since(produce(v))\big) \lor \\
&\quad \forall x, v, t. r \neq start(consume(x, v), t) \land r \neq end(produce(v), t) \land \\
&\qquad quantity = y
\end{aligned} \tag{5.27}$$

$$\begin{aligned}
&\Box[r]Performing(a) \equiv \\
&\quad \exists t. r = start(a, t) \lor \\
&\quad Performing(a) \land \forall t. r \neq end(a, t)
\end{aligned} \tag{5.28}$$

$$\begin{aligned}
&\Box[r]since(a) = t \equiv \\
&\quad r = start(a, t) \lor \\
&\quad since(a) = t \land \forall t'. r \neq start(a, t')
\end{aligned} \tag{5.29}$$

$$\Box[r]now = time(r) \tag{5.30}$$

Initial state axioms Σ_0

$$now = 0 \land quantity = 9$$

Figure 5.1: Basic action theory of the producer-consumer domain

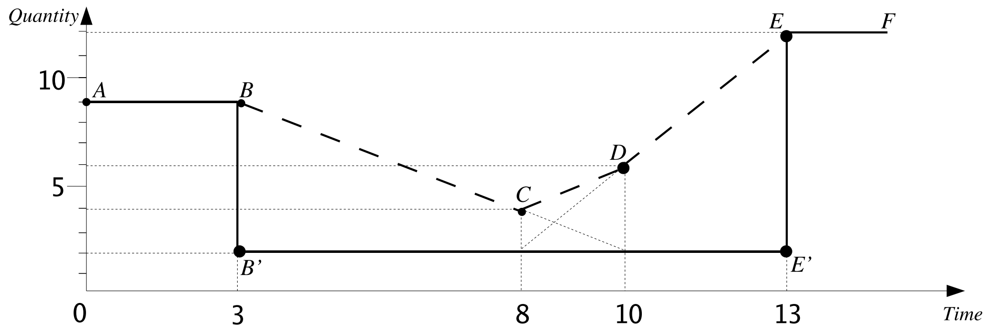

Figure 5.2: Linear change modeled by discretized durative actions

This condition is used as a duration constraint, which says, if the *consume* action reduces the quantity of product by x at a constant rate of v, then the duration of it must be $\frac{x}{v}$. If the end event of $consume(x, v)$ does not happen exactly $\frac{x}{v}$ time after the start event of it, then the action sequence is considered impossible. For example,

$$[start\big(consume(7,1),3\big)]\cdots[end\big(consume(7,1),10\big)]$$

may be a executable trace, whereas

$$[start\big(consume(7,1),3\big)]\cdots[end\big(consume(7,1),13\big)]$$

is not, since $13 - 3 \neq 7/1$.

From (5.27), we see that we only take into account the overall result of numerical changes, *i.e.* how much the quantity increases or decreases due to the execution of a durative actions. We do not differentiate how the numerical value *quantity* actually changes within the interval of the durative actions. In the producer-consumer example, we may have a linear producer and a linear consumer, where, for instance, the consumer runs between time 3 and 10, linearly consuming 7 units of product, and the producer runs between time 8 and 13, linearly producing 10 units of product, as shown in Figure 5.2 with dashed line $\overline{ABCDEF}$. However, the basic action theory in Figure 5.1 models the numerical change in a way illustrated by the continuous line $\overline{ABB'E'EF}$.

As a result, we may get incorrect value of a numerical fluent within the interval of an action that modifies it. For example, in the example above, we have

$$\Sigma \models [start(consume(7,1),3)][start(produce(2),8)]quantity = 2$$

but in reality, the value is 4 at that time point.

However, due to its simplicity, discretized models like this are useful in many domains. In fact, it is powerful enough as long as we do not reason about numerical values when the durative actions that change them are in progress. For example, the basic action theory in Figure 5.1 draws the correct conclusion

$$\begin{aligned}\Sigma \models\ & [start(consume(7,1),3)][start(produce(2),8)]\\ & [end(consume(7,1),10)][end(produce(2),13)]quantity = 12\end{aligned}$$

For those domains where we do need to model how a numerical fluent changes, we have to further extend the language to integrate continuous fluents. This will be the focus of the next section.

5.5 Modeling Continuous Changes

In this section, we go beyond the discretized view of numerical changes, and consider how we may model continuous effects in $\mathcal{ES}$. Here, we stick to linear changes only, although in principle, arbitrary elementary functions can be modeled with the approach introduced here. The reason is two-fold. First, the linear function is the simplest non-trivial function, so its property is easy to study. Second, our target language, PDDL 2, only allows for at most linear changes in continuous actions.

Following our discussion in Section 3.5.4, we use continuous fluents of the form $linear(x, v, t)$ to represent linearly changing numerical fluent functions. However, we go one step further here: We assume that *linear*'s are domain elements. In particular, we consider them as extra instances of *Number*, and define operations and relations on them. In particular, we assert:

$$\begin{aligned}& linear(x_0, v_0, t_0) + y = z \equiv \\ & \qquad \exists x_1, v_1, t_1 . y = linear(x_1, v_1, t_1) \land \\ & \qquad\quad z = linear\big((x_0 + x_1) - (v_0 t_0 + v_1 t_1), v_0 + v_1, 0\big) \lor \\ & \qquad \forall x_1, v_1, t_1 . y \neq linear(x_1, v_1, t_1) \land \\ & \qquad\quad z = linear\big(x_0 + y, v_0, t_0\big)\end{aligned} \tag{5.31}$$

$$linear(x_0, v_0, t_0) = linear\big(x_0 + v_0 \cdot (t - t_0), v_0, t\big) \tag{5.32}$$

$$linear(x_0, 0, t_0) = x_0 \tag{5.33}$$

Furthermore, we define the following two operations on *linear* objects

$$\begin{aligned}
&eval(x,t) = y \equiv \\
&\qquad \exists x_0, v_0, t_0 . x = linear(x_0, v_0, t_0) \wedge y = x_0 + v_0 \cdot (t - t_0) \vee \qquad (5.34)\\
&\qquad \forall x_0, v_0, t_0 . x \neq linear(x_0, v_0, t_0) \wedge y = x \\
&rate(x) = y \equiv \\
&\qquad \exists x_0, v_0, t_0 . x = linear(x_0, v_0, t_0) \wedge y = v_0 \vee \qquad (5.35)\\
&\qquad \forall x_0, v_0, t_0 . x \neq linear(x_0, v_0, t_0) \wedge y = 0
\end{aligned}$$

(5.31) says that if we add two *linear* entities together, then we combine the two and get a new *linear* object; otherwise, we are adding a real number to a *linear* entity, and we simply add it to the first component of the *linear* function. Notice that the addition operation, as axiomatized in Σ_{num}, is commutative, so the case of $y + linear(x_0, v_0, t_0)$ can be reduced to the one in (5.31). (5.32) builds the equivalence between linear objects with different parameters. (5.33) says that if a linear object has a changing rate of 0, then it degrades to a number. (5.34) defines the evaluation function for a linear object at a certain time point. $eval(x,t)$ returns the value of x at t. Finally, (5.35) defines the function $rate(x)$, which extracts the changing rate out of x.

In some cases, we need to evaluate the truth value of a fluent formula W that mentions continuous fluents, at a certain time point t. For this purpose, we substitute in W all the terms whose values are linear objects with with their instant numerical values at time t. This process is called the *numerization* of a formula. For notational convenience, we define a macro $Eval[W,t]$, which denotes the numerized formula of W at time t. Formally, $Eval[W,t]$ is defined inductively on the structure of W as follows:[6]

$$\begin{array}{lcl}
Eval[l_1 \otimes l_2, t] & = & \big(eval(l_1,t) \otimes eval(l_2,t)\big) \\
Eval[\neg W, t] & = & \neg Eval[W,t] \\
Eval[W_1 \wedge W_2, t] & = & (Eval[W_1,t] \wedge Eval[W_2,t]) \\
Eval[\forall x . W, t] & = & \forall x . Eval[W,t] \\
Eval[Poss(A), t] & = & Eval[\pi^a_A, t] \\
Eval[[r]W, t] & = & [r]Eval[W,t] \\
Eval[\Box W, t] & = & \Box Eval[W,t]
\end{array}$$

One of the applications of *Eval* is the numerization of precondition formulas. For example, when we write

$$\Box Poss(a) \equiv \pi$$

[6] Here $\otimes$ stands for the relations $=$ and $<$.

$$\begin{aligned}
&\Box[r]quantity = y \equiv\\
&\quad \exists x, v, t.\ r = start(consume(x, v), t) \wedge\\
&\qquad y = quantity - linear(0, v, t) \vee\\
&\quad \exists x, v, t.\ r = end(consume(x, v), t) \wedge\\
&\qquad y = quantity + linear(0, v, t) \vee\\
&\quad \exists v, t.\ r = start(produce(v), t) \wedge\\
&\qquad y = quantity + linear(0, v, t) \vee\\
&\quad \exists v, t.\ r = end(produce(v), t) \wedge\\
&\qquad y = quantity - linear(0, v, t) \vee\\
&\quad \forall x, v, t.\ r \neq start(consume(x, v), t) \wedge r \neq end(consume(x, v), t) \wedge\\
&\qquad r \neq start(produce(v), t) \wedge r \neq end(produce(v), t) \wedge\\
&\qquad quantity = y
\end{aligned}$$

Figure 5.3: Successor state axioms of a continuous fluent in the producer-consumer domain

we expect that all linear terms in π are substituted with their numerical values at the happening time of a. As a result, the precondition axiom should be rewritten as

$$\Box Poss(a) \equiv Eval[\pi, time(a)]$$

when we allow for continuous effects in the domain.

With this model of linear change, the successor state axiom (5.26) of the producer-consumer example in the previous section can be rewritten as shown in Figure 5.3.

To understand this axiom, let us take the $consume(x, v)$ action for example. At its start event, $quantity$ begins to decrease at a rate of v, so we subtract a $linear(0, v, t)$ component from it; at its end event, this decrease stops, so we add the $linear(0, v, t)$ back. At the first glance, it may seem strange that the first argument to the $linear$ function is 0 instead of x, but actually, this is correct, in that we do not subtract anything immediately when $consume(x, v)$ starts, and only alter the changing rate of $product$. A natural question is then: How do we ensure that an amount of x is consumed finally? Remember that the duration of $consume(x, v)$ is always $\frac{x}{v}$ according to the precondition axiom, so a correct subtraction from $quantity$ is realized due to the fact that the product of v and the duration of $consume(x, v)$ is exactly the value of x. This makes sense especially when another action modifies the value of *product* when $consume(x, v)$ is in progress, which is the case in Figure 5.2, where the quantity of product at time 10 is 6 instead of 2, for example.

Now, let us have a look at how our new basic action theory correctly models all the happenings in Figure 5.2.

1. Initially, we have

   $$\Sigma \models quantity = 9$$

   So

   $$\Sigma \models eval(quantity, t) = 9$$

2. After $start(consume(7, 1), 3)$, we have

   $$\Sigma \models [start(consume(7, 1), 3)]quantity = 9 - linear(0, 1, 3)$$

   or equivalently

   $$\Sigma \models [start(consume(7, 1), 3)]quantity = linear(9, -1, 3)$$

   So

   $$\begin{aligned}\Sigma \models\ & [start(consume(7, 1), 3)] \\ & eval(quantity, t) = eval(linear(9, -1, 3), t)\end{aligned}$$

   which may be simplified to

   $$\Sigma \models [start(consume(7, 1), 3)]eval(quantity, t) = 6 - t$$

3. After $start(produce(2), 8)$, we have

   $$\begin{aligned}\Sigma \models\ & [start(consume(7, 1), 3)][start(produce(2), 8] \\ & quantity = linear(9, -1, 3) + linear(0, 2, 8)\end{aligned}$$

   or simply

   $$\begin{aligned}\Sigma \models\ & [start(consume(7, 1), 3)][start(produce(2), 8] \\ & quantity = linear(-4, 1, 0)\end{aligned}$$

   So we have

   $$\begin{aligned}\Sigma \models\ & [start(consume(7, 1), 3)][start(produce(2), 8] \\ & eval(quantity, t) = -4 + t\end{aligned}$$

   For example, at time 9, the quantity of product is $-4+9 = 5$ unit, which coincides with Figure 5.2.

4. After $end(consume(7,1),10)$, we have

$$\begin{aligned}\Sigma \models [start(consume(7,1),3)][start(produce(2),8)]\\ [end(consume(7,1),10)]\\ quantity = linear(-4,1,0) + linear(0,1,10)\end{aligned}$$

or equivalently,

$$\begin{aligned}\Sigma \models [start(consume(7,1),3)][start(produce(2),8)]\\ [end(consume(7,1),10)]quantity = linear(-14,2,0)\end{aligned}$$

So we have

$$\begin{aligned}\Sigma \models [start(consume(7,1),3)][start(produce(2),8)]\\ [end(consume(7,1),10)]eval(quantity,t) = -14+2t\end{aligned}$$

5. Finally, after $end(produce(2),13)$, we have

$$\begin{aligned}\Sigma \models [start(consume(7,1),3)][start(produce(2),8)]\\ [end(consume(7,1),10)][end(produce(2),13]\\ quantity = linear(-14,2,0) - linear(0,2,13)\end{aligned}$$

which can be simplified to

$$\begin{aligned}\Sigma \models [start\big(consume(7,1),3\big)][start\big(produce(2),8\big)]\\ [end\big(consume(7,1),10\big)][end\big(produce(2),13\big)]quantity = 12\end{aligned}$$

As we can see, the result corresponds exactly to the reality illustrated in Figure 5.2.

5.6 Coercive Actions

In the current form of the basic action theory, all the actions happen deliberately, in the sense that the agent has the freedom to choose from a candidate pool of possible actions, and determine which action happens next. In reality, however, the agent does not always have that freedom. For example, the snack bar closes daily at 20:00, regardless whether the robot is hungry at midnight and wants a fat dinner; a pot of water starts boiling 5 minutes after the robot puts it on the heater, although the robot is now enjoying a movie in another room. The former "closing shop" action is a sort of predetermined exogenous event, whereas the latter "boiling" action follows the laws of the nature. Whichever reason the actions are caused to happen, they share a common property that they are coercive, in that they *must* occur at a certain time point.

Remember that in Section 3.5.5, we discussed the concept of natural actions in the situation calculus, which was originally proposed by Pinto [Pin94] and later further elaborated by Reiter [Rei96].

However, both work focuses on natural actions that is determined by the laws of nature. What we would like to have here, in contrast, is to model the coerciveness caused by both natural laws and predetermined exogenous actions. Furthermore, we are more interested in the influence of such coercive actions on the deliberate actions, rather than how a domain evolves under the laws of nature and without external interference. So in this section, we will adapt their concepts and ideas, and define our own notion of coerciveness in the logic $\mathcal{ES}$.

To identify a coercive action, we do not use the predicate symbol *natural* as Pinto and Reiter did, since it seems a bit strange to assert that a shop closing action is natural. Instead, we name it *Obli*, which is a predicate on $\mathcal{N}$. *Obli*(a) means that action a must happen in the current situation. Note that the happening time of a is simply $time(a)$. The definition of *Obli* resembles that of the *Poss* predicate, and has the form

$$\Box Obli(a) \equiv \Pi$$

where Π is a big disjunction of the necessary and sufficient conditions for each action to occur. For example,

$$\begin{aligned}\Box Obli(a) \equiv&\\ &a = closeShop(20) \vee\\ &\exists p, t. a = boil(p,t) \wedge Performing\big(heat(p)\big)\\ &\qquad \wedge t - since\big(heat(p)\big) > 5\end{aligned}$$

which means, at time 20, the shop must be closed, and if we are heating a pot of water for more than 5 minutes, then the water boils.

The next question is how to guard the obligatory condition, *i.e.* how to ensure that in all the situations and at all times in an action sequence, as long as $Obli(a)$ is true, a is indeed executed.

Unfortunately, due to the fact that we are using interleaved concurrency, which is a simplified but limited model of concurrency, we can only consider a special case here, where the following Assumptions 5.1 and 5.2 are satisfied. In Chapter 8, we shall illustrate the necessity of having Assumption 5.1 in the interleaved account for concurrency with a concrete example, and discuss how this assumption may be dropped when true concurrency is used.

Assumption 5.1. *Coercive actions never happen concurrently with other actions.*

Assumption 5.2. *Coercive actions are always followed by other actions.*

With the assumptions above, our proposal is to add a new axiom Σ_{Obli} to the basic action theory, insisting that for an action to be possible, no other obligatory actions are scheduled before it. Formally,

$$\Box Poss(a) \supset \neg\big(\exists a'.Obli(a') \wedge now < time(a') < time(a)\big) \tag{5.36}$$

In this way, we consider an action sequence that violates the obligatory condition $Obli(a)$ to be impossible and thus invalid. In the next section, we will return to coercive actions when we model continuous invariant constraints and timed initial literals.

Chapter 6

The Semantic Mapping between PDDL and $\mathcal{ES}$

In Chapter 2, we introduced the language of PDDL along with an informal account of its state-transitional semantics; in Chapter 5, we investigated how the logic $\mathcal{ES}$ may be extended to incorporate time and concurrency. With these results at hand, we are now ready to establish the semantic mapping between PDDL and the basic action theories in $\mathcal{ES}$. This mapping will serve as a declarative semantics of PDDL, since for each construct in the PDDL problem definition as well as the plan, we know exactly what it means in terms of logic.

The development of this chapter is like the following. First, we briefly survey, in Section 6.1, the existing work on two proper subsets of PDDL, namely, the STRIPS and the ADL fragments of the language. Then, we extend these results by considering each of the new features in our target subset. Section 6.2 deals with numerics and plan metrics, followed by the mapping of durative actions in Section 6.3. Finally in Section 6.4, we turn to timed initial literals.

6.1 Existing Work

6.1.1 STRIPS as Progression in the Situation Calculus

The first attempt to define a declarative semantics for planning languages dates back to Lin and Reiter, who related the state updates in STRIPS to first-order progression in the situation calculus [LR95]. They proved, among other things, that there exists a direct correspondence between relational STRIPS and basic action theories with complete initial database and strongly context free successor state axioms.

Definition 6.1 (Relational STRIPS)**.** A STRIPS system is called *relational*, if its state representation

- consists of a set of ground atomic facts about predicates other than equality;
- assumes the closed-world assumption on predicates including equality.

Definition 6.2 (Strongly context free successor state axioms)**.** A successor state axiom is *strongly context free* if and only if it has the form

$$\begin{aligned} & F\big(\vec{x}, do(a,s)\big) \equiv \\ & \quad (\exists \vec{v}^{(1)})a = A_1(\vec{\xi}^{(1)}) \vee \cdots \vee (\exists \vec{v}^{(m)})a = A_m(\vec{\xi}^{(m)}) \vee \\ & \quad F(\vec{x}, s) \wedge \neg\big((\exists \vec{w}^{(1)})a = B_1(\vec{\eta}^{(1)}) \vee \cdots \vee (\exists \vec{w}^{(n)})a = B_n(\vec{\eta}^{(n)})\big) \end{aligned} \tag{6.1}$$

Here, the A_i and B_j are function symbols of sort *action*, not necessarily distinct from one another. $\vec{\xi}^{(i)}$ and $\vec{\eta}^{(j)}$ are sequences of distinct variables that include *all* the variables in $\vec{x}$; the remaining variables of $\vec{\xi}^{(i)}$ and $\vec{\eta}^{(j)}$ are those being existentially quantified by $\vec{v}^{(i)}$ and $\vec{w}^{(j)}$, respectively.

Strongly context free successor state axioms are special cases of context free successor state axioms that are defined in Section 3.4. They satisfy the additional condition that for any action term other than the variable a, its parameter list contains all the variables that appear in $\vec{x}$.

For example, according to this definition, the successor state axiom

$$\begin{aligned} At\big(x, y, do(a,s)\big) \equiv & \exists y'.a = move(x, y', y) \vee \\ & At(x, y, s) \wedge \neg\big(\exists y''.a = move(x, y, y'')\big) \end{aligned}$$

is strongly context free, whereas

$$\begin{aligned} In\big(x, do(a,s)\big) \equiv & a = putIn(x) \vee \\ & In(x, s) \wedge a \neq emptyBriefcase \end{aligned}$$

is not, since the action term *emptyBriefcase* does not have the parameter x.

In order to represent the relational databases in purely logic notion, Lin and Reiter defined the *official axiomatization* as follows:

Definition 6.3 (Official axiomatization of relational database)**.** Let $\mathbf{D}$ be a relational database. $\mathbf{D}$'s *official axiomatization* Ω is constructed by:

1. Suppose $P(\vec{C}^{(1)}), \cdots, P(\vec{C}^{(n)})$ are *all* of the ground atoms of predicate P occurring in $\mathbf{D}$, where $\vec{C}^{(i)}$ are tuples of constant symbols. Then include in Ω the sentence

$$P(\vec{x}) \equiv \vec{x} = \vec{C}^{(1)} \vee \cdots \vee \vec{x} = \vec{C}^{(n)}$$

If the predicate P has no ground instances in $\mathbf{D}$, then include in Ω the sentence

$$P(\vec{x}) \equiv \text{FALSE}$$

2. Include in Ω unique names axioms for all constant symbols.

With this definition, a Strips database, $\mathbf{D}_0$, where $F(\vec{C}^{(1)}), \cdots, F(\vec{C}^{(n)})$ are all the positive instances of the predicate symbol F, has the corresponding initial theory containing the sentence

$$F(\vec{x}, S_0) \equiv \vec{x} = \vec{C}^{(1)} \vee \cdots \vee \vec{x} = \vec{C}^{(n)} \tag{6.2}$$

where $\vec{C}^{(i)}$ are tuples of constant symbols of sort *object*.

In order to show how this is related with progression in the situation calculus, let us assume that we have a basic action theory of the form $\Sigma = \mathcal{FA} \cup \Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_{una}$, where

1. The sort *object* ranges over all objects other than situations and actions in the domain; the only function symbols of sort *object* are constants.

2. Σ_0 contains one sentence of the form in (6.2) for each fluent symbol F.

3. All successor state axioms in Σ_{post} are strongly context free.

Now, we investigate the progression of such a theory through a ground action term α.

Due to the unique names axioms on actions and the form of Σ_0, we may reduce (6.1) to

$$\begin{aligned} F\big(\vec{x}, do(\alpha, S_0)\big) \equiv \vec{x} = \vec{X}^{(1)} \vee \cdots \vee \vec{x} = \vec{X}^{(l)} \vee \\ [\vec{x} = \vec{C}^{(1)} \vee \cdots \vee \vec{x} = \vec{C}^{(n)}] \wedge \vec{x} \neq \vec{Y}^{(1)} \wedge \cdots \wedge \vec{x} \neq \vec{Y}^{(m)} \end{aligned}$$

Let $\vec{x} = \vec{C}^{(1)} \vee \cdots \vee \vec{x} = \vec{C}^{(r)}$ be all the $\vec{C}^{(k)}$ that are different tuples from all of the $\vec{Y}^{(i)}$, then

$$F\big(\vec{x}, do(\alpha, S_0)\big) \equiv \vec{x} = \vec{X}^{(1)} \vee \cdots \vee \vec{x} = \vec{X}^{(l)} \vee \vec{x} = \vec{C}^{(1)} \vee \cdots \vee \vec{x} = \vec{C}^{(r)}$$

This construction can be performed for all the predicate symbols in the domain, and let us denote the resulting theory with Σ_α, then it can be shown that Σ_α is the progression of Σ_0 through the action term α.

To see how this is related to the state update in Strips, notice that the situation suppressed version of Σ_α is an official axiomatization of a new database, which we denote with $\mathbf{D}_\alpha$. It contains, for the predicate symbol F for example, the instances $F(\vec{X}^{(1)}), \cdots, F(\vec{X}^{(l)}), F(\vec{C}^{(1)}), \cdots, F(\vec{C}^{(r)})$.

Comparing this set of instances of F with $\mathbf{D}_0$, it is easy to see that the new database is obtained by performing the following addition and deletion to the initial database:

1. Delete from $\mathbf{D}_0$ the instances $F(\vec{Y}^{(1)}), \cdots, F(\vec{Y}^{(m)})$;
2. Add to to $\mathbf{D}_0$ the instances $F(\vec{X}^{(1)}), \cdots, F(\vec{X}^{(l)})$.

These addition and deletion are exactly what the add list and delete list express in the operator definition of α. Therefore, the progression of the initial theory correctly models the state update with STRIPS action operators.

6.1.2 ADL as Progression in the Logic $\mathcal{ES}$

Generalizing the result in the STRIPS subset, Claßen *et al.* showed that ADL can be given a declarative semantics with the logic $\mathcal{ES}$ [CELN07]. Although in principle, it is also possible in the situation calculus, they argue that the formulation in $\mathcal{ES}$ is more succinct. In this subsection, we shall look at how they have defined the semantic mapping. Notice that in their original work, both closed-world and open-world cases are considered. Here, for simplicity, we only review the closed-world fragment, since the open-world case is excluded from PDDL 2, and not interesting to the topic of this thesis.

We begin with constructing the basic action theory given an ADL problem description.

Recall that in Section 2.3, we defined a structural representation of PDDL descriptions, where an action A is represented by the tuple $\langle \vec{z} : \vec{\tau}, \pi_A, \epsilon_A \rangle$, and the normal form of the effects ϵ_A has the form

$$\begin{aligned}&\textstyle\bigwedge_{F_j} \forall \vec{x_j} : \tau_{F_j}.\big(\gamma^+_{F_j,A}(\vec{x_j}, \vec{z}) \Rightarrow F_j(\vec{x_j})\big)\wedge\\ &\textstyle\bigwedge_{F_j} \forall \vec{x_j} : \tau_{F_j}.\big(\gamma^-_{F_j,A}(\vec{x_j}, \vec{z}) \Rightarrow \neg F_j(\vec{x_j})\big)\end{aligned}$$

The following construction of the basic action theory makes use of this notational convention.

The successor state axioms Σ_{post}:

Given the normal form of operators in (2.1), we construct, for each fluent predicate F_j, the formulas

$$\gamma^+_{F_j} \triangleq \bigvee_{\gamma^+_{F_j,A_i} \in NF(A_i)} \exists \vec{z_i}.a = A_i(\vec{z_i}) \wedge \gamma^+_{F_j,A_i} \tag{6.3}$$

$$\gamma^-_{F_j} \triangleq \bigvee_{\gamma^-_{F_j,A_i} \in NF(A_i)} \exists \vec{z_i}.a = A_i(\vec{z_i}) \wedge \gamma^-_{F_j,A_i} \tag{6.4}$$

where $\gamma^*_{F_j,A_i} \in NF(A_i)$ means that a clause with A_i exists in $\gamma^*_{F_j}$ only if some $\gamma^*_{F_j,A_i}$ exists in the effect of A_i $(* \in \{+,-\})$. Then, the successor state axiom for F_j is simply

$$\Box[a]F_j(\vec{x_j}) \equiv \gamma^+_{F_j} \wedge \tau_{F_j}(\vec{x_j}) \vee F_j(\vec{x_j}) \wedge \neg\gamma^-_{F_j} \tag{6.5}$$

where $\tau_{F_j}(\vec{x_j})$ is the typing constraint, which ensures that $F_j(\vec{x_j})$ can become true only if $\vec{x_j}$ are consistent with the type definitions for F_j's arguments.

In addition, we need to construct the successor state axioms for each type τ_i. Since typing is situation-independent, we simply have

$$\Box[a]\tau_i(x) \equiv \tau_i(x) \tag{6.6}$$

The precondition axiom Σ_{pre}:

The precondition axiom is obtained by a disjunction of all the precondition formulas of operators with case distinction, which has the form

$$\Box Poss(a) \equiv \bigvee_{1\le i\le m} \exists \vec{z_i} : \vec{\tau_i}.a = A_i(\vec{z_i}) \wedge \pi_{A_i}$$

The initial description Σ_0:

The initial theory actually consists of two parts, sentences characterizing the initial world state and those for the typing of objects.

The construction from the initial world state is similar to how we obtain the official axiomatization of relational databases in Section 6.1.1. Specifically, suppose that $F_j(\vec{o_1}), \cdots, F_j(\vec{o_{k_j}})$ are *all* the positive instances of F_j in the initial state I, then we include the sentence

$$F_j(\vec{x_j}) \equiv (\vec{x_1} = \vec{o_1} \vee \cdots \vee \vec{x_{k_j}} = \vec{o_{k_j}}) \tag{6.7}$$

in the initial theory Σ_0, which we sometimes write as $\Sigma_0(I)$ to indicate that it is an initial theory obtained from the state description I.

As for typing, we add to Σ_0 the following sentences

$$\tau_i(x) \equiv \big(\tau_{i_1}(x) \vee \cdots \vee \tau_{i_{k_i}}(x)\big) \tag{6.8}$$

$$F(x_{j_1}, \cdots, x_{j_{k_j}}) \supset \big(\tau_{j_1}(x_{j_1}) \wedge \cdots \wedge \tau_{j_{k_j}}(x_{j_{k_j}})\big) \tag{6.9}$$

$$\tau_i(x) \equiv (x = o_{j_1} \vee \cdots \vee x = o_{j_{k_i}}) \tag{6.10}$$

$$Object(x) \equiv \big(\tau_1(x) \vee \cdots \vee \tau_l(x)\big) \tag{6.11}$$

Here, (6.8) deals with "either" statements; (6.9) declares the type of each argument to a predicate symbol; (6.10) says that $x = o_{j_1}, \cdots, x = o_{j_{k_i}}$ are all the instances of primitive type τ_i; finally, (6.11) declares *Object* to be the union of all other types.

With the basic action theory constructed in this way, now we consider the state I_α obtained by executing an ADL operator, α, in I. First, Lemma 6.4 formalizes some simple consequences from the construction above.

Lemma 6.4. *Let $\alpha = A(\vec{p})$ be an action and $\vec{o}$ be object parameters for the fluent F_j, then*

1. *$\Sigma_0 \wedge \tau_{\vec{F_j}}(\vec{o})$ is satisfiable iff $\vec{o}$ are of the correct types according to the* ADL *problem description.*
2. *$\Sigma_0 \models \gamma^{*}_{F_j}{}^{\vec{x_j}}_{\vec{o}}{}^{a}_{\alpha}$ iff $\gamma^{*}_{F_j,A}(\vec{o},\vec{p})$ is satisfiable in the original* ADL *state I, where $* \in \{+,-\}$.*
3. *$\Sigma_0 \cup \Sigma_{pre} \models Poss(\alpha)$ iff $\pi_A(\vec{p})$ is satisfied in the original* ADL *state I and $\vec{p}$ are of the correct types.*

Proof. We shall prove a more general version of this lemma, as well as the coming Theorem 6.5, in Chapter 7, after we make the extensions with functional fluents. $\square$

Under the condition that $\Sigma_0(I) \cup \Sigma_{pre} \models Poss(\alpha)$, Lemma 6.4 tells us that the execution of α is a possible one in the ADL state I. So, we do the following for all F_j and all $\vec{o}$ such that $F_j(\vec{o})$ is type consistent

1. If $\Sigma_0 \models \gamma^{+}_{F_j}{}^{\vec{x_j}}_{\vec{o}}{}^{a}_{\alpha}$, then add $F_j(\vec{o})$;
2. If $\Sigma_0 \models \gamma^{-}_{F_j}{}^{\vec{x_j}}_{\vec{o}}{}^{a}_{\alpha}$, then delete $F_j(\vec{o})$;

Let us denote the set of literals to be added by *Adds* and those to be deleted by *Dels*, then the new state description is

$$I_\alpha = (I \setminus Dels) \cup Adds$$

Claßen *et al.* related the state update above to first-order progression in $\mathcal{ES}$ (c.f. Section 4.4), and proved the following theorem:

Theorem 6.5. *Let I_α be the state description obtained as shown above, given an* ADL *problem and a ground action α. Further, let $\Sigma_\alpha = \{[\alpha]\psi | \psi \in \Sigma_0(I_\alpha)\}$, where $\Sigma_0(I_\alpha)$ is the initial theory constructed from I_α instead of I. For all F_j, let the consistency condition $\models \neg(\gamma^{+}_{F_j} \wedge \gamma^{-}_{F_j})^{a}_{\alpha}$ hold. Then Σ_α is a progression of Σ_0 through α.*

Theorem 6.5 implies that the application of an ADL operator under the state transitional semantics is the same as progressing the initial theory of the corresponding basic action theory.

6.2 Numerical Expressions and Plan Metrics

With the existing results described above as a starting point, we are now ready to investigate the additional features in the more general version of PDDL 2. This section is devoted to the numerical extensions, which corresponds to PDDL domains with the `:fluents` requirement.

In the coming subsections, 6.2.1 investigates how numerical properties in the PDDL problem definition can be mapped to the basic action theory in $\mathcal{ES}$, 6.2.2 discusses the plan metrics, and finally 6.2.3 illustrates all of our ideas in this section with an example.

6.2.1 Numerical Expressions

As mentioned in Section 2.2.2, two restrictions are proposed for the usage of numerical expressions in PDDL according to Fox and Long's definition. First, only numerical valued functions are allowed, *i.e.* all functions are of the form $Object^n \to \mathbb{R}$, and functions like $Object^n \to Object$ are disallowed. Second, numbers are not terms in the language, and thus cannot appear as parameters to actions, predicates or functions.

Notice that our extension to $\mathcal{ES}$ is more general, in that all syntactically valid numerical expressions in PDDL can be represented in $\mathcal{ES}$, but not vice versa.

In order to build a direct correspondance, we define additional constraints in the basic action theory, such that the restrictions in Fox and Long's semantics can be modeled. Indeed, we only need to ensure that all function symbols in $\mathbf{f}$ have parameters of type $Object$ (or its subtype) and function value of type $Number$. For this purpose, we add the following typing constraint for each function symbol f_j in the initial theory:

$$f_j(x_{j_1}, \cdots, x_{j_{k_j}}) = y \supset \big(\tau_{j_1}(x_{j_1}) \wedge \cdots \wedge \tau_{j_{k_j}}(x_{j_{k_j}}) \wedge Number(y)\big) \tag{6.12}$$

Now, let us investigate how the numerical extensions, compared to the ADL subset, influences the basic action theory. We shall start from the successor state axioms Σ_{post} for functional fluents, and then discuss the precondition axioms Σ_{pre} and the initial description Σ_0.

The successor state axioms

The mapping from PDDL operators to the successor state axioms in $\mathcal{ES}$ is similar to the ADL subset, except that we add additional axioms for numerical fluent functions. So for predicate symbols and typing constraints, the construction still follows (6.3)–(6.6).

For the numerical function symbols, remember that in Section 2.2, we used the formula $\gamma^v_{f_j,A_i}(\vec{x_j}, y, \vec{z_i})$ to denote the condition to assign y to $f_j(\vec{x_j})$ in the effect of action $A_i(\vec{z_i})$. So, like in the case of fluent predicates, we define the effect formula to assign y to $f_j(\vec{x_j})$ as

$$\gamma^v_{f_j} \equiv \bigvee_{\gamma^v_{f_j,A_i} \in NF(A_i)} \exists \vec{z_i}.a = A_i(\vec{z_i}) \wedge \gamma^v_{f_j,A_i} \tag{6.13}$$

and the one to change the value of $f_j(\vec{x_j})$ at all as

$$\gamma_{f_j} \equiv \bigvee_{\gamma_{f_j,A_i} \in NF(A_i)} \exists \vec{z_i}.a = A_i(\vec{z_i}) \wedge \gamma_{f_j,A_i} \tag{6.14}$$

Then the successor state axiom for numerical function f_j is

$$\Box[a]f_j(\vec{x_j}) = y \equiv \gamma^v_{f_j} \wedge \tau_{f_j}(\vec{x_j}) \wedge Number(y) \vee f_j(\vec{x_j}) = y \wedge \neg\gamma_{f_j} \tag{6.15}$$

where τ_{f_j} is the typing constraints for the function symbol f_j.

The precondition axiom

The precondition axiom is obtained in the same fashion as in the Adl subset. Remember that $<$ is a predicate symbol, $=$ is the identity relation in the logic, and $>$, $\leq$ and $\geq$ are abbreviations, so the precondition axiom does not contain any new feature, although we may now freely write comparison between numerical expressions in it.

The initial state description

The initial description Σ_0 is constructed in a similar way to what we did in Section 6.1.2. The mapping is exactly the same for the initialization of predicates. Then, for typing of objects, apart from (6.8)–(6.11), we add the new constraint for function symbols in (6.12). Finally, for the (numerical) functional atoms, we include for each functional symbol f_j the sentence

$$\begin{aligned} f_j(\vec{x}) = y \equiv \vec{x} = \vec{o_1} \wedge y = r_1 \vee \\ \cdots \vee \\ \vec{x} = \vec{o_{k_j}} \wedge r = y_{k_j} \end{aligned} \tag{6.16}$$

where $f_j(\vec{o_1}) = r_1, \cdots, f_j(\vec{o_{k_j}}) = r_{k_j}$ characterize the initial values of all the instances of f_j.

6.2.2 Metrics

Before presenting an example to illustrate our solution to numerical expressions above, let us first have a closer look at the plan metrics in PDDL.

As defined by Fox and Long, a plan metric is a numerical value, on which basis the quality of a plan is evaluated. Any quantity that is used for defining a metric has to be instrumented in the domain description, *i.e.* an action has to specify how it may change the quantity in the `:effect` section. So pragmatically, a metric is nothing but a real value obtained from arithmetics on some of the numerical fluents in the domain.

The use of metrics for a planner is to optimize the plan that it generates. In contrast, for our purpose here, metrics do not serve as a guideline in the basic action theory in $\mathcal{ES}$. Nevertheless, we always have access to the metric values, since they are formulas on normal numerical fluents in the domain. This is somewhat similar to the goal description in PDDL, in that we do not map the goal sentence to anything in the basic action theory, yet we can always check whether the goal is satisfied. In the following example, we will also see how the metrics of a plan may be evaluated according to the basic action theory in $\mathcal{ES}$.

6.2.3 An Example

We now illustrate, with a concrete example, how to map a PDDL problem description with numerical expressions and metrics to the basic action theory in the logic $\mathcal{ES}$. We do so by deriving a basic action theory from the sample PDDL specification in Figure 6.1.

The Precondition Axiom Σ_{pre}

As usual, the right hand side of the precondition axioms is obtained by

$$\pi = \bigvee_{1\leq i\leq m} \exists \vec{z_i} : \vec{\tau_i}.a = A_i(\vec{z_i}) \wedge \pi_{A_i} \tag{6.17}$$

In our example here, there is only one action $drive(v, l_1, l_2)$, so the precondition axiom is

$$\begin{aligned}&\Box Poss(a) \equiv\\ &\qquad \exists v, l_1, l_2.\ Vehicle(v) \wedge Location(l_1) \wedge Location(l_2) \wedge\\ &\qquad\quad a = drive(v, l_1, l_2) \wedge At(v, l_1) \wedge Accessible(v, l_1, l_2) \wedge\\ &\qquad\quad fuel\_level(v) \geq fuel\_required(l_1, l_2)\end{aligned}$$

```
(define (domain metricVehicle)
   (:requirements :strips :typing :fluents)
   (:types vehicle location)
   (:predicates (at ?v - vehicle ?p - location)
                (accessible ?v - vehicle ?p1 ?p2 - location ))
   (:functions  (fuel-level ?v - vehicle)
                (fuel-used ?v - vehicle)
                (fuel-required ?p1 ?p2 - location)
                (total-fuel-used))
   (:action drive
     :parameters (?v - vehicle ?from ?to - location)
     :precondition (and (at ?v ?from)
                        (accessible ?v ?from ?to)
                        (>= (fuel-level ?v) (fuel-required ?from ?to)))
     :effect (and (not (at ?v ?from))
                  (at ?v ?to)
                  (decrease (fuel-level ?v) (fuel-required ?from ?to))
                  (increase (total-fuel-used) (fuel-required ?from ?to))
                  (increase (fuel-used ?v) (fuel-required ?from ?to))))
)
(define (problem metricVehicleExample)
  (:domain metricVehicle)
  (:objects truck car - vehicle Paris Berlin Rome Madrid - location)
  (:init (at truck Rome)
         (at car Paris)
         (= (fuel-level truck) 100)
         (= (fuel-level car) 100)
         (accessible car Paris Berlin)
         (accessible car Berlin Rome)
         (accessible car Rome Madrid)
         (accessible truck Rome Paris)
         (accessible truck Rome Berlin)
         (accessible truck Berlin Paris)
         (= (fuel-required Paris Berlin) 40)
         (= (fuel-required Berlin Rome) 30)
         (= (fuel-required Rome Madrid) 50)
         (= (fuel-required Rome Paris) 35)
         (= (fuel-required Rome Berlin) 40)
         (= (fuel-required Berlin Paris) 40)
         (= (total-fuel-used) 0)
         (= (fuel-used car) 0)
         (= (fuel-used truck) 0))
  (:goal (and (at truck Paris) (at car Rome)))
  (:metric minimize (total-fuel-used))
)
```

Figure 6.1: Domain and problem description with numerics and metrics

The Successor State Axioms Σ_{post}

There should be one successor state axiom for each predicate and each function symbol. In the example domain, we have two predicate symbols $At(v,l)$ and $Accessible(v,l_1,l_2)$, and four function symbols $fuel\_level(v)$, $fuel\_used(v)$, $fuel\_required(l_1,l_2)$ and $total\_fuel\_used$.

The effect formulas for predicates (including those for typing) are derived in the same way as illustrated in Section 6.1.2 for the Adl subset of Pddl, *i.e.* from the normal form of the effects of the only action $drive(v,l_1,l_2)$, we obtain

$$\begin{aligned}
\gamma^+_{At} &= \exists v,l_1,l_2.a = drive(v,l_1,l_2) \wedge x_1 = v \wedge x_2 = l_2\\
\gamma^-_{At} &= \exists v,l_1,l_2.a = drive(v,l_1,l_2) \wedge x_1 = v \wedge x_2 = l_1\\
\gamma^+_{Accessible} &= false\\
\gamma^-_{Accessible} &= false
\end{aligned}$$

then, the successor state axioms for the predicates are simply

$$\begin{aligned}
&\Box[a]At(x_1,x_2) \equiv\\
&\quad \gamma^+_{At} \wedge Vehicle(x_1) \wedge Location(x_2) \vee\\
&\quad At(x_1,x_2) \wedge \neg\gamma^-_{At}\\
&\Box[a]Accessible(x_1,x_2,x_3) \equiv\\
&\quad \gamma^+_{Accessible} \wedge Vehicle(x_1) \wedge Location(x_2) \wedge Location(x_3) \vee\\
&\quad Accessible(x_1,x_2,x_3) \wedge \neg\gamma^-_{Accessible}
\end{aligned}$$

For the function symbols $f(\vec{x})$, we define their update conditions in the way as specified in Section 6.2.1:

$$\begin{array}{lcl}
\gamma^v_{fuel\_required}(x_1,x_2,y) & \equiv & \text{FALSE}\\
\gamma_{fuel\_required}(x_1,x_2) & \equiv & \text{FALSE}\\
\gamma^v_{fuel\_level}(x,y) & \equiv & \exists v,l_1,l_2.a = drive(v,l_1,l_2) \wedge x = v \wedge\\
 & & y = fuel\_level(v) - fuel\_required(l_1,l_2)\\
\gamma_{fuel\_level}(x) & \equiv & \exists v,l_1,l_2.a = drive(v,l_1,l_2) \wedge x = v\\
\gamma^v_{fuel\_used}(x,y) & \equiv & \exists v,l_1,l_2.a = drive(v,l_1,l_2) \wedge x = v \wedge\\
 & & y = fuel\_used(v) + fuel\_required(l_1,l_2)\\
\gamma_{fuel\_used}(x) & \equiv & \exists v,l_1,l_2.a = drive(v,l_1,l_2) \wedge x = v\\
\gamma^v_{total\_fuel\_used}(y) & \equiv & \exists v,l_1,l_2.a = drive(v,l_1,l_2) \wedge\\
 & & y = total\_fuel\_used + fuel\_required(l_1,l_2)\\
\gamma_{total\_fuel\_used} & \equiv & \exists v,l_1,l_2.a = drive(v,l_1,l_2)
\end{array}$$

Then the successor state axioms for functional fluents are

$$\begin{aligned}
&\Box[a]fuel\_required(x_1, x_2) = y \equiv\\
&\quad \gamma^v_{fuel\_required}(x_1, x_2, y) \land Location(x_1) \land Location(x_2) \land Number(y) \lor\\
&\quad fuel\_required(x_1, x_2) = y \land \neg\gamma_{fuel\_required}(x_1, x_2)\\
&\Box[a]fuel\_level(x) = y \equiv\\
&\quad \gamma^v_{fuel\_level}(x, y) \land Vehicle(x) \land Number(y) \lor\\
&\quad fuel\_level(x) = y \land \neg\gamma_{fuel\_level}(x)\\
&\Box[a]fuel\_used(x) = y \equiv\\
&\quad \gamma^v_{fuel\_used}(x, y) \land Vehicle(x) \land Number(y) \lor\\
&\quad fuel\_used(x) = y \land \neg\gamma_{fuel\_used}(x)\\
&\Box[a]total\_fuel\_used = y \equiv\\
&\quad \gamma_{total\_fuel\_used}(y) \land Number(y) \lor\\
&\quad total\_fuel\_used = y \land \neg\gamma_{total\_fuel\_used}
\end{aligned}$$

In summary, if we simplify all the sentences above, we get the successor state axioms for all typing constraints, relational and functional fluents in the domain as follows:

$$\begin{aligned}
&\Box[a]Vehicle(x) \equiv Vehicle(x)\\
&\Box[a]Location(x) \equiv Location(x)\\
&\Box[a]At(v, l) \equiv\\
&\quad \exists l'.Location(l') \land a = drive(v, l', l) \land Vehicle(v) \land Location(l) \lor\\
&\quad At(v, l) \land \neg\exists l'.Location(l') \land a = drive(v, l, l')\\
&\Box[a]Accessible(v, l_1, l_2) \equiv Accessible(v, l_1, l_2)\\
&\Box[a]fuel\_required(l_1, l_2) = y \equiv fuel\_required(l_1, l_2) = y\\
&\Box[a]fuel\_level(v) = y \equiv\\
&\quad \exists l_1, l_2.Location(l_1) \land Location(l_2) \land a = drive(v, l_1, l_2) \land\\
&\qquad y = fuel\_level(v) - fuel\_required(l_1, l_2) \land\\
&\qquad Vehicle(v) \land Number(y) \lor\\
&\quad fuel\_level(v) = y \land \neg\exists l_1, l_2.a = drive(v, l_1, l_2)\\
&\qquad Location(l_1) \land Location(l_2)\\
&\Box[a]fuel\_used(v) = y \equiv\\
&\quad \exists l_1, l_2.Location(l_1) \land Location(l_2) \land a = drive(v, l_1, l_2) \land\\
&\qquad y = fuel\_used(v) + fuel\_required(l_1, l_2) \land\\
&\qquad Vehicle(v) \land Number(y) \lor
\end{aligned}$$

$$
\begin{aligned}
&\quad fuel\_used = y \wedge \neg\exists l_1, l_2.a = drive(v, l_1, l_2)\wedge \\
&\qquad Location(l_1) \wedge Location(l_2) \\
&\Box[a]total\_fuel\_used = y \equiv \\
&\quad \exists v, l_1, l_2.Vehicle(v) \wedge Location(l_1) \wedge Location(l_2) \\
&\qquad a = drive(v, l_1, l_2) \wedge Number(y)\wedge \\
&\qquad y = total\_fuel\_used + fuel\_required(l_1, l_2)\vee \\
&\quad total\_fuel\_used = y \wedge \neg\exists v, l_1, l_2.a = drive(v, l_1, l_2) \\
&\qquad Vehicle(v) \wedge Location(l_1) \wedge Location(l_2)
\end{aligned}
$$

The Initial Description Σ_0

Finally, following the construction procedure described above, it is easy to obtain the following initial theory for this domain:

$$
\begin{aligned}
Vehicle(x) &\equiv x = truck \vee x = car \\
Location(x) &\equiv x = paris \vee x = berlin \vee x = rome \vee x = madrid \\
At(x, y) &\supset Vehicle(x) \wedge Location(y) \\
Accessible(x, y, z) &\supset Vehicle(x) \wedge Location(y) \wedge Location(z) \\
fuel\_level(x) = y &\supset Vehicle(x) \\
fuel\_used(x) = y &\supset Vehicle(x) \\
fuel\_required(x, y) &\supset Location(x) \wedge Location(y) \\
At(x, y) &\equiv (x = truck \wedge y = rome) \vee (x = car \wedge y = paris) \\
fuel\_level(x) = y &\equiv (x = truck \wedge y = 100) \vee (x = car \wedge y = 100) \\
Accessible(x, y, z) &\equiv (x = car \wedge y = paris \wedge z = berlin)\vee \\
&\quad (x = car \wedge y = berlin \wedge z = rome)\vee \\
&\quad (x = car \wedge y = rome \wedge z = madrid)\vee \\
&\quad (x = truck \wedge y = rome \wedge z = paris)\vee \\
&\quad (x = truck \wedge y = rome \wedge z = berlin)\vee \\
&\quad (x = truck \wedge y = berlin \wedge z = paris) \\
fuel\_required(x, y) = z &\equiv (x = paris \wedge y = berlin \wedge z = 40)\vee \\
&\quad (x = berlin \wedge y = rome \wedge z = 30)\vee \\
&\quad (x = rome \wedge y = madrid \wedge z = 50)\vee \\
&\quad (x = rome \wedge y = paris \wedge z = 35)\vee \\
&\quad (x = rome \wedge y = berlin \wedge z = 40)\vee \\
&\quad (x = berlin \wedge y = paris \wedge z = 40) \\
total\_fuel\_used = y &\equiv (y = 0)
\end{aligned}
$$

$$fuel\_used(x) = y \equiv (x = car \wedge y = 0) \vee (x = truck \wedge y = 0)$$

With the basic action theory above, we may correctly conclude, for example,

$$\begin{aligned}\Sigma_0 \cup \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_{num} \models & [drive(car, paris, berlin)] \\ & [drive(truck, rome, paris)][drive(car, berlin, rome)] \\ & \big(At(truck, paris) \wedge At(car, rome) \wedge total\_fuel\_used = 105\big)\end{aligned}$$

and we can see that the metric $total\_fuel\_used$ has got the desired correct value 105.

6.3 Durative Actions

In this section, we investigate how we may give the durative actions in PDDL a declarative semantics with the logic $\mathcal{ES}$. First, we shall look at the simpler case where at most discretized numerical effects exist. Then, we turn to the more general but complicated case with continuous numerical effects. In both cases, instead of building everything anew, we base our discussion upon the results from the previous sections, and only remark on the necessary changes.

6.3.1 Discretized Durative Actions

This subsection deals with discretized durative actions with at most end duration constraints and temporally local conditional effects. Following the notational convention in Section 2.3, suppose that we have simple actions[1] $A_1, \cdots, A_p \in \mathfrak{A}$ and durative actions $\widetilde{A}_1, \cdots, \widetilde{A}_q \in \widetilde{\mathfrak{A}}$, where $\epsilon^o_{\widetilde{A}_j} = \emptyset$, $\delta^s_{\widetilde{A}_j} \equiv$ TRUE, and each effect in $\epsilon^e_{\widetilde{A}_j}$ has the form $\langle \text{TRUE}, \text{TRUE}, \varphi^e_{j,i} \rangle \Rightarrow \psi_{j,i}$, for all $j = 1, \cdots, q$. Our goal is to translate these action descriptions to the basic action theory in $\mathcal{ES}$.

Remember that as mentioned in Section 5.4, we represent durative actions in $\mathcal{ES}$ with two simple actions denoting the start and the end events of it. This coincides with the definition of durative actions in PDDL, where conditions and effects are temporally annotated, as denoted by $\langle \pi^s_{\widetilde{A}_j}, \pi^e_{\widetilde{A}_j} \rangle$ and $\langle \epsilon^s_{\widetilde{A}_j}, \epsilon^e_{\widetilde{A}_j} \rangle$, respectively. So the first step in deriving the basic action theory from the PDDL description is to map the start and the end conditions and effects.

For the durative action $\widetilde{A}_j$, to make its start event possible to happen, at least $\pi^s_{\widetilde{A}_j}$ must be satisfied. And if it really happens, then the effects $\epsilon^s_{\widetilde{A}_j}$

[1] In this section and the following, the terms "simple action" and "spontaneous action" both stand for non-durative actions.

will take place. Similarly, $\pi^e_{\widetilde{A}_j}$ and $\epsilon^e_{\widetilde{A}_j}$ can be considered as the (partial) precondition and effects of the end event. Since we assume that no end effect $\epsilon^e_{\widetilde{A}_j}$ have a premise that is temporally prior to its happening, all the conditional effects, if any, are local as well. So, to some extent, the start and the end events are just like the simple actions in $\mathfrak{A}$. Therefore, for notational benefits, we define them as two *virtual simple actions* with the normal form $\widetilde{A}^s_j\langle\vec{z} : \vec{\tau}, \pi^s_{A_j}, \epsilon^s_{A_j}\rangle$ and $\widetilde{A}^e_j\langle\vec{z} : \vec{\tau}, \pi^e_{A_j}, \epsilon^e_{A_j}\rangle$. Further, we denote $\mathscr{S}(\widetilde{\mathfrak{A}}) = \{\widetilde{A}^s_j\}$ and $\mathscr{E}(\widetilde{\mathfrak{A}}) = \{\widetilde{A}^e_j\}$. Then, to translate the problem description, we need to consider not only the simple actions in $\mathfrak{A}$, as we described in the previous section, but also the virtual simple actions in $\mathscr{S}(\widetilde{\mathfrak{A}})$ and $\mathscr{E}(\widetilde{\mathfrak{A}})$, derived from $\widetilde{\mathfrak{A}}$.

As a result, the precondition axiom will have the form

$$
\begin{aligned}
\Box Poss(a) \equiv time(a) \geq now \wedge \Big(\\
&\bigvee_{1\leq i\leq p} \exists\vec{z_i}, t.a = A_i(\vec{z_i}, t) \wedge \pi_{A_i} \vee \\
&\bigvee_{1\leq j\leq q} \exists\vec{z_j}, t.a = start(\widetilde{A}_j(\vec{z_j}), t) \wedge \pi^s_{\widetilde{A}_j} \vee \\
&\bigvee_{1\leq j\leq q} \exists\vec{z_j}, t.a = end(\widetilde{A}_j(\vec{z_j}), t) \wedge \pi^e_{\widetilde{A}_j} \Big)
\end{aligned}
$$

and the structure of successor state axioms remains unchanged, except that the effect formulas, $\gamma^+_{F_j}$, $\gamma^-_{F_j}$, $\gamma^v_{f_j}$ and γ_{f_j}, consider actions in $\mathfrak{A} \cup \mathscr{S}(\widetilde{\mathfrak{A}}) \cup \mathscr{E}(\widetilde{\mathfrak{A}})$

$$
\begin{aligned}
\gamma^+_{F_j} &\equiv \bigvee\nolimits_{\substack{A_i\in\mathfrak{A}\cup\mathscr{S}(\widetilde{\mathfrak{A}})\cup\mathscr{E}(\widetilde{\mathfrak{A}}) \\ \gamma_{F^+_j,A_i}\in NF(A_i)}} \exists\vec{z_i}.a = A_i(\vec{z_i}) \wedge \gamma^+_{F_j,A_i} \\
\gamma^-_{F_j} &\equiv \bigvee\nolimits_{\substack{A_i\in\mathfrak{A}\cup\mathscr{S}(\widetilde{\mathfrak{A}})\cup\mathscr{E}(\widetilde{\mathfrak{A}}) \\ \gamma^-_{F_j,A_i}\in NF(A_i)}} \exists\vec{z_i}.a = A_i(\vec{z_i}) \wedge \gamma^-_{F_j,A_i} \\
\gamma^v_{f_j} &\equiv \bigvee\nolimits_{\substack{A_i\in\mathfrak{A}\cup\mathscr{S}(\widetilde{\mathfrak{A}})\cup\mathscr{E}(\widetilde{\mathfrak{A}}) \\ \gamma^v_{f_j,A_i}\in NF(A_i)}} \exists\vec{z_i}.a = A_i(\vec{z_i}) \wedge \gamma^v_{f_j,A_i} \\
\gamma_{f_j} &\equiv \bigvee\nolimits_{\substack{A_i\in\mathfrak{A}\cup\mathscr{S}(\widetilde{\mathfrak{A}})\cup\mathscr{E}(\widetilde{\mathfrak{A}}) \\ \gamma_{f_j,A_i}\in NF(A_i)}} \exists\vec{z_i}.a = A_i(\vec{z_i}) \wedge \gamma_{f_j,A_i}
\end{aligned}
$$

However, two things are neglected in this direct transformation. First, the duration constraint $\delta_{\widetilde{A}_j}$ is not yet taken into account. Second, the invariant conditions $\pi^o_{\widetilde{A}_j}$ for durative actions are not protected. Now, we shall look at how they may be implemented in the basic action theory.

Remember that in Section 5.4, we defined the following axioms in Σ_{dura}:

$$\begin{aligned}
&\Box[a]Performing(\widetilde{a}) \equiv\\
&\quad \exists t.a = start(\widetilde{a}, t)\vee\\
&\quad Performing(\widetilde{a}) \wedge \neg\exists t.a = end(\widetilde{a}, t)\\
&\Box[a]since(\widetilde{a}) = t \equiv\\
&\quad a = start(\widetilde{a}, t)\vee\\
&\quad since(\widetilde{a}) = t \wedge \neg\exists t'.a = start(\widetilde{a}, t')\\
&\Box Poss(start(\widetilde{a}), t) \supset \neg Performing(\widetilde{a})\\
&\Box Poss(end(\widetilde{a}), t) \supset Performing(\widetilde{a})
\end{aligned}$$

and concluded that we may obtain the duration of durative action $\widetilde{a}$ with $\big(t - since(\widetilde{a})\big)$ at $end(\widetilde{a}, t)$, the end event of $\widetilde{a}$. As a result, we may check whether the end duration constraint $\delta^{e}_{\widetilde{a}}$ is fulfilled at this time point, more precisely, in the precondition of the end event.

Formally, we assert

$$\Box Poss\big(end(\widetilde{a}, t)\big) \supset (\delta^{e}_{\widetilde{a}})^{duration}_{t-since(\widetilde{a})} \tag{6.18}$$

where $(\delta^{e}_{\widetilde{a}})^{duration}_{t-since(\widetilde{a})}$ is the formula obtained by simultaneously substituting all the free occurrences of *duration* in $\delta^{e}_{\widetilde{a}}$ with $\big(t - since(\widetilde{a})\big)$.

Here, we do not consider start durative constraints, and assume $\delta^{s}_{\widetilde{a}} =$ TRUE. We shall return to the general case, where start duration constraints do exist, at the end of Section 6.3.2, after we discuss about the inter-temporal conditional effects.

(6.18) implies that if the (end) duration constraint of any durative action is violated, then the end event of it is not possible, and thus the whole plan becomes invalid. In this way, we ensure that the duration constraint is satisfied in any valid plan.

Now, let us turn to the invariant conditions for durative actions. Here, the difficulty in protecting the invariants is that they must hold in a *continuous* interval, whereas $\mathcal{ES}$ can only reason about discrete points in the situations. One naïve solution would be to discretize the interval, and say that the condition holds at each point in the discretization. However, the happenings within the duration of the action may be arbitrary, so no matter how we discretize it, even if we can guarantee that the condition holds at all the points, we do not know whether some action happens in between, violating the condition.

To overcome this problem, we propose to protect the invariant conditions by not allowing actions that violates them to happen. The idea is as follows:

we check, before executing any action a, whether after its execution the invariant condition $\pi^o_{\widetilde{A}_j}$ of some active durative action $\widetilde{A}_j$ is violated. If this is the case, the execution of a should be forbidden.

As a result, we consider action a possible to happen, only if in the resulting situation, the invariant condition of all durative actions that are in progress are satisfied. Formally, the idea can be captured by the following $\mathcal{ES}$ sentence:

$$\Box Poss(a) \supset \bigwedge_{\widetilde{A}_j} \mathcal{R}[a, Performing(\widetilde{A}_j) \supset \pi^o_{\widetilde{A}_j}] \tag{6.19}$$

Here, the right-hand side of (6.19) makes use of the regressed formula of $Performing(\widetilde{A}_j) \supset \pi^o_{\widetilde{A}_j}$ through action a. The validity of this approach is established from the following fact: We want to ensure $Performing(\widetilde{A}_j) \supset \pi^o_{\widetilde{A}_j}$ holds after executing a, which can be expressed with

$$[a]\big(Performing(\widetilde{A}_j) \supset \pi^o_{\widetilde{A}_j}\big)$$

However, this formula is in $[a]$. In order to remove the $[a]$ operator and get a formula that only talks about the current situation, we may resort to regression, and finally get (6.19).

With the considerations on duration constraints and invariant conditions above, we should extend the precondition axiom as shown in the following (6.20):

$$\begin{aligned}
&\Box Poss(a) \equiv time(a) \geq now \wedge \big(\bigwedge_{\widetilde{A}_j} \mathcal{R}[a, Performing(\widetilde{A}_j) \supset \pi^o_{\widetilde{A}_j}]\big) \wedge \Big(\\
&\bigvee_{A_i} a = A_i \wedge \pi_{A_i} \vee\\
&\bigvee_{\widetilde{A}_j} \exists t.a = start(\widetilde{A}_j, t) \wedge \neg Performing(\widetilde{A}_j) \wedge \pi^s_{\widetilde{A}_j} \vee\\
&\bigvee_{\widetilde{A}_j} \exists t.a = end(\widetilde{A}_j, t) \wedge Performing(\widetilde{A}_j) \wedge \pi^e_{\widetilde{A}_j} \wedge (\delta^e_{\widetilde{A}_j})^{duration}_{t-since(\widetilde{A}_j)}\Big)
\end{aligned} \tag{6.20}$$

In order to see how our approach above works, let us look at a "light tunnel" example in Figure 6.2, and derive the basic action theory from this PDDL action definition.

In this example, we consider a domain where there are a few tunnels of different lengths. Each tunnel is equipped with lighting facilities. In order to go through a tunnel, there must be light in it. This is guaranteed by the infrared sensor: if someone is entering a dark tunnel, the sensor automatically

```
(:durative-action go-thru
    :parameters (?l - tunnel)
    :duration (= ?duration (/ (length ?l) velocity))
    :condition (over all (light ?l))
    :effects (and (at start (in ?l))
                  (when (at start (not (light? l)))
                        (at start (light ?l)))
                  (at end (not (in ?l))))
)

(:action switch-on
    :parameters (?l - tunnel)
    :condition (not (light ?l))
    :effect (light ?l))
)

(:action switch-off
    :parameters (?l - tunnel)
    :condition (light ?l)
    :effect (not (light ?l)))
)
```

Figure 6.2: Action definitions for a simple light-tunnel problem.

turns on the lights in the tunnel. Apart from that, it is possible to turn the lights on and off manually, by the two simple actions `switch-on` and `switch-off`, respectively.

Now, let us build the basic action theory for this domain, step by step from the action specifications.

According to (6.18), the duration constraint

```
:duration (= ?duration (/ (length ?l) velocity))
```

can be guaranteed by inserting the formula

$$t - since(goThru(l)) = \frac{length(l)}{velocity}$$

to the precondition of the end event $end(goThru(l), t)$.

Then, to ensure that the invariant condition

```
(over all (light ?l))
```

for the durative action $goThru(l)$ is satisfied, we need to ensure the satisfaction of the condition

$$\mathcal{R}[a, Performing\big(goThru(l)\big) \supset Light(l)]$$

which is extended to the following formula

$$\begin{gathered}\Big(\exists t.a = start\big(goThru(l), t\big) \vee \\ Performing\big(goThru(l)\big) \wedge \neg\exists t.a = end\big(goThru(l)\big)\Big) \supset \\ \Big(\exists t.a = switchOn(l, t) \vee \\ \exists t.a = start\big(goThru(l), t\big) \vee \\ Light(l) \wedge \neg\exists t.a = switchOff(l, t)\Big)\end{gathered} \tag{6.21}$$

For example, suppose a situation where $Performing(goThru(l)) \wedge Light(l)$ is true, and let a be the action $switchOff(l)$, then the evaluation result of (6.21) is false, which intuitively means, when someone is in the middle of a light tunnel, it is not possible to switch off the light in it. This is exactly what we desire.

With the considerations above, the precondition axiom for the light tunnel domain is the formula:

$$\begin{aligned}&\Box Poss(a) \equiv \\ &\quad time(a) \geq now \wedge \\ &\quad \Big(\exists t.a = start\big(goThru(l), t\big) \vee \\ &\quad Performing\big(goThru(l)\big) \wedge \\ &\quad \neg\exists t.a = end\big(goThru(l)\big)\Big) \supset \\ &\quad \Big(\exists t.a = switchOn(l, t) \vee \\ &\quad\quad \exists t.a = start\big(goThru(l), t\big) \vee \\ &\quad\quad Light(l) \wedge \neg\exists t.a = switchOff(l, t)\Big) \wedge \\ &\quad \Big(\ \exists t.a = switchOn(x, t) \wedge \neg Light(x) \vee \\ &\quad\quad \exists t.a = switchOff(x, t) \wedge Light(x) \vee \\ &\quad\quad \exists t.a = start\big(goThru(x), t\big) \wedge \neg Performing\big(goThru(x)\big) \vee \\ &\quad\quad \exists t.a = end\big(goThru(x), t\big) \wedge Performing\big(goThru(x), t\big) \wedge \\ &\quad\quad\quad t - since\big(goThru(x)\big) = length(x)/velocity\Big)\end{aligned} \tag{6.22}$$

Finally, it is not difficult to obtain the following successor state axioms for the target fluent predicates. For space reasons, we omit the ones for the rigid properties here.

$$\begin{aligned}&\Box[a]In(x) \equiv \\ &\quad \exists t.a = start\big(goThru(x), t\big) \vee \\ &\quad In(x) \wedge \neg\exists t.a = end\big(goThru(x), t\big)\end{aligned} \tag{6.23}$$

$$\begin{aligned}&\Box[a]Light(x) \equiv \\ &\quad \exists t.a = switchOn(x, t) \vee \\ &\quad \exists t.a = start\big(goThru(x), t\big) \vee \\ &\quad Light(x) \wedge \neg\exists t.a = switchOff(x, t)\end{aligned} \tag{6.24}$$

6.3.2 Inter-Temporal Property Reference

In Section 6.3.1, we only consider temporally local properties. This includes the following two restrictions. First, we only allow intra-temporal conditional effects, and disallow for the end effects with premise annotated with `at start` or `over all`. Second, we do not consider start duration constraints. In this subsection, we loosen these restrictions, and investigate the more general cases.

Inter-temporal conditional effects

Let us start with inter-temporal conditional effects. So far, we assumed that for a durative action $\widetilde{A}_j(\vec{x_j})$, each effect in $\epsilon^e_{\widetilde{A}_j}$ has the form

$$\langle true, true, \varphi^e_{j,i}\rangle \Rightarrow \psi_{j,i}$$

Now, let us consider the general case

$$\langle \varphi^s_{j,i}, \varphi^o_{j,i}, \varphi^e_{j,i}\rangle \Rightarrow \psi_{j,i}$$

where $\varphi^s_{j,i}$ and $\varphi^o_{j,i}$ may be non-trivial formulas. In the following discussion, we use the $\vec{q_i}$ to denote all the free variables in $\psi_{j,i}$ that are distinct from those in $\vec{x_j}$.

Like Fox and Long, to accommodate inter-temporal conditional effects, we introduce an auxiliary fluent predicate to memorize the "old state", when we meet a premise that has a temporal annotation prior to its corresponding effect.

In the case of a start-end conditional effect of a durative action $\widetilde{A}_j(\vec{p_j})$, for each non-trivial premise formula $\varphi^s_{j,i}$, we define a predicate $\xi^s_{j,i}(\vec{p_j}, \vec{q_i})$. The value of $\xi^s_{j,i}(\vec{p_j}, \vec{q_i})$ is updated to the truth value of $\varphi^s_{j,i}$ whenever the

$start(\widetilde{A}_j(\vec{p_j}), t)$ event is activated, and to FALSE after the duration of the action. Formally, the successor state axiom for $\xi^s_{j,i}(\vec{p_j}, \vec{q_i})$ is

$$\begin{aligned}
&\Box[a]\xi^s_{j,i}(\vec{p_j}, \vec{q_i}) \equiv \\
&\quad \exists t.a = start\big(\widetilde{A}_j(\vec{p_j}), t\big) \land \varphi^s_{j,i} \lor \\
&\quad \xi^s_{j,i}(\vec{p_j}, \vec{q_i}) \land \neg \exists t'.a = end(\widetilde{A}_j(\vec{p_j}), t')
\end{aligned}$$

This has the effect that $\xi^s_{j,i}(\vec{p_j}, \vec{q_i})$ "memorizes" the truth value of $\varphi^s_{j,i}$ at the time point when the start event of $\widetilde{A}_j(\vec{p_j})$ is executed.

The overall-end conditional effects are a little more subtle to deal with, since the satisfaction of the premise is not determined by a single state, but instead according to an interval. Like for the start premise, we introduce an auxiliary fluent predicate $\xi^o_{j,i}(\vec{p_j}, \vec{q_i})$ for each non-trivial overall premise formula $\varphi^o_{j,i}$. The value of $\xi^o_{j,i}(\vec{p_j}, \vec{q_i})$ is determined in a way similar to how we protected the invariant condition for durative actions described in Section 6.3.1. Assigned to FALSE initially, the truth value of $\xi^o_{j,i}(\vec{p_j}, \vec{q_i})$ may be later changed in two ways. First, when the start event of $\widetilde{A}_j(\vec{p_j})$ is executed, if $\varphi^o_{j,i}$ becomes TRUE immediately afterward, then the value of $\xi^o_{j,i}(\vec{p_j}, \vec{q_i})$ is assigned to TRUE. Second, if the durative action finishes or an action falsifies the overall premise, then the value of $\xi^o_{j,i}(\vec{p_j}, \vec{q_i})$ is updated to FALSE. This idea is captured by the following successor state axiom

$$\begin{aligned}
&\Box[a]\xi^o_{j,i}(\vec{p_j}, \vec{q_i}) \equiv \\
&\quad \exists t.a = start(\widetilde{A}_j(\vec{p_j}), t) \land \mathcal{R}[a, \varphi^o_{j,i}] \lor \\
&\quad \xi^o_{j,i}(\vec{p_j}, \vec{q_i}) \land \neg \exists t'.a = end(\widetilde{A}_j(\vec{p_j}), t') \land \mathcal{R}[a, \varphi^o_{j,i}]
\end{aligned}$$

Then, to decide whether the effect $\psi_{j,i}$ should take place at the end event of $\widetilde{A}_j(\vec{p_j})$, we simply test whether $\xi^s_{j,i}(\vec{p_j}, \vec{q_i})$ and $\xi^o_{j,i}(\vec{p_j}, \vec{q_i})$ both have the value TRUE, apart from considering the end-premise $\varphi^e_{j,i}$.

In order to illustrate our solution to the problem of inter-temporal conditional effects, we consider, as an example, a variant of the light tunnel problem, whose action definitions are shown in Figure 6.3.

In the new version, the infrared sensor "remembers" the state of the lights before someone enters the tunnel: if the lights are originally on, it remains on after she exits the tunnel; otherwise, it turns on the light in the beginning, and turns it off again when she leaves. Besides, there is only one simple action `switch` that toggles the light between on and off.

The main focus here is the conditional end effect

$$\langle Infrared(l) \land \neg Light(l), true, true \rangle \Rightarrow \neg Light(l)$$

```
(:durative-action go-thru
    :parameters (?l - tunnel)
    :duration (= ?duration (/ (length ?l) velocity))
    :condition (over all (light ?l))
    :effects (and (at start (in ?l))
                  (at end (not (in ?l)))
                  (when (and (at start (infrared ?l))
                             (at start (not (light? l))))
                        (and (at start (light ?l))
                             (at end (not (light ?l))))))
)

(:action switch
    :parameters (?l - tunnel)
    :effect (and (when (light ?l) (not (light ?l)))
                 (when (not (light ?l) (light ?l))))
)
```

Figure 6.3: The full version of light tunnel problem.

Since the premise is a start condition, we introduce a predicate $\xi^s_{goThru,Light}(l)$, whose successor state axiom is

$$\begin{aligned}
&\Box[a]\xi^s_{goThru,Light}(l) \equiv \\
&\quad \exists t.a = start\big(goThru(l),t\big) \wedge Infrared(l) \wedge Light(l) \vee \\
&\quad \xi^s_{goThru,Light}(l) \wedge \neg\exists t'.a = end\big(goThru(l),t'\big)
\end{aligned}$$

The successor state axiom for $Light(l)$, then, becomes

$$\begin{aligned}
&\Box[a]Light(l) \equiv \\
&\quad \exists t.a = switch(l,t) \wedge \neg Light(l) \vee \\
&\quad \exists t.a = start\big(goThru(l),t\big) \wedge Infrared(l) \wedge \neg Light(l) \vee \\
&\quad Light(l) \wedge \neg\Big(\exists t.a = switch(l,t) \wedge Light(l) \vee \\
&\qquad \exists t.a = end\big(goThru(l),t\big) \wedge \xi^s_{goThru,Light}(l)\Big)
\end{aligned}$$

Start duration constraints

With the solution to inter-temporal conditional effects presented above, let us now turn to the duration constraints with `at start` annotations.

When Fox and Long defined their semantics for the Pddl, they split the duration constraint into a start constraint DC^{DA}_{start} and an end constraint DC^{DA}_{end}, and handle them respectively as a precondition of the start and end events. This approach is possible, since the duration of an action is specified explicitly in a plan, and their semantics is offline in nature.

In contrast, in $\mathcal{ES}$, all durative actions are represented by their derived simple actions, and no duration is represented explicitly in the plan. As a result, there is no way to access the duration of an action before the end of it. So it is impossible to directly copy Fox and Long's approach to check the satisfaction of the start duration constraint.

To solve this problem, our proposal is to memorize the fluent functional properties at the beginning of the durative action, and postpone the evaluation of the formula to the end, when the duration value is also accessible.

Formally, for each functional fluent $f_i(\vec{q_i})$ that is mentioned in some start duration constraint $\delta^s_{\widetilde{A}_j}(\vec{p_j})$, we introduce a new fluent $f^s_{j,i}(\vec{p_j}, \vec{q_i})$, which has the following successor state axiom:

$$\begin{aligned}
&\Box[a] f^s_{j,i}(\vec{p_j}, \vec{q_i}) = y \equiv \\
&\quad \exists t. a = start(\widetilde{A}_j(\vec{p_j}), t) \wedge y = f_i(\vec{q_i}) \vee \\
&\quad f^s_{j,i}(\vec{p_j}, \vec{q_i}) = y \wedge \neg\exists t. a = start(\widetilde{A}_j(\vec{p_j}), t)
\end{aligned}$$

Then, the satisfaction of the whole duration constraint may be checked when the end event is executed by

$$\Box Poss\big(end(\widetilde{A}_j, t)\big) \supset (\delta^s_{\widetilde{A}_j})^{f_i(\vec{q_i})}_{f^s_{j,i}(\vec{p_j}, \vec{q_i})}{}^{duration}_{t-since(\widetilde{A}_j)} \wedge (\delta^e_{\widetilde{A}_j})^{duration}_{t-since(\widetilde{A}_j)} \qquad (6.25)$$

For notational simplicity, we sometimes write $(\delta_{\widetilde{A}_j})^{duration}_{t-since(\widetilde{A}_j)}$ to denote the right hand side of (6.25).

6.3.3 Continuous Effects

In the previous two subsections, we demonstrated how to map a Pddl description with durative actions to the basic action theories in the logic $\mathcal{ES}$, under the assumption $\epsilon^o_{\widetilde{A}_j} = \emptyset$ for all durative actions $\widetilde{A}_j$. This is the subset with only discretized durative actions. Now, let us consider the mapping with continuous durative actions, where $\epsilon^o_{\widetilde{A}_j} \neq \emptyset$, for some durative actions $\widetilde{A}_j$.

According to Section 2.3.2, the normal form of $\epsilon^o_{\widetilde{A}_j}$ is a set of structures in $\langle op, P, Q\rangle$. So, we shall discuss how to map the effects of this form to the continuous extensions of the logic $\mathcal{ES}$ as described in Section 5.5.

Remember that in the logic $\mathcal{ES}$ (as well as the situation calculus), all the changes of the states occur at the happening points of actions. Although for continuous changing quantities, the value varies from time to time within a situation, the pattern of its change is already determined at the start of the situation. This fact suggests that continuous effects may be modeled at the points where the start and end events occur as well.

As a result, our proposal is to add an $\big((op) \cdot linear(0, Q, t)\big)$ component to $f_k(\vec{x_k})$ at the start and subtract the same component at the end of the durative action $\widetilde{A}_j$ whose "overall effect" $\epsilon^o_{\widetilde{A}_j}$ contains the effect $\langle op, f_k(\vec{x_k}), Q\rangle$. Here, (op) is positive $(+)$ when the effect is an `increase` and negative $(-)$ when it is a `decrease`; t is the time when the start (respectively, end) event happens.

For example, suppose that the overall effect of a durative action $fly(p)$ is

```
(decrease (fuel-level ?p) (* #t (consume-rate ?p)))
```

which has the normal form

$$\langle -, fuel\_level(p), consume\_rate(p)\rangle$$

then we have the following update conditions:

$$\begin{aligned}
&\gamma^v_{fuel\_level(p), fly^s(p)} \equiv \\
&\qquad y = fuel\_level(p) + (-1) \cdot linear(0, consume\_rate(p), t) \\
&\gamma^v_{fuel\_level(p), fly^e(p)} \equiv \\
&\qquad y = fuel\_level(p) - (-1) \cdot linear(0, consume\_rate(p), t)
\end{aligned}$$

As a result, the successor state axiom for the numerical fluent $fuel\_level(p)$ is

$$\begin{aligned}
&\Box[a] fuel\_level(p) = y \equiv \\
&\quad \exists t.a = start\big(fly(p), t\big) \land \\
&\qquad y = fuel\_level(p) - linear\big(0, consume\_rate(p), t\big) \lor \\
&\quad \exists t.a = end\big(fly(p), t\big) \land \\
&\qquad y = fuel\_level(p) + linear\big(0, consume\_rate(p), t\big) \lor \\
&\quad fuel\_level(p) = y \land \neg\Big(\exists t.a = start\big(fly(p), t\big) \lor \exists t.a = end\big(fly(p), t\big)\Big)
\end{aligned}$$

So far, it does not seem difficult to translate the continuous effect description in PDDL description to the logic $\mathcal{ES}$. However, if we think about the invariant constraints, we will find that it is not trivial to protect them in the presence of continuous effects.

Remember that in the discretized case, we protect the invariant constraints of durative actions by not allowing actions that violate the constraints to occur. The validity of this approach is based on the assumption that the only way to change the truth value of a formula is by the execution of an action. However, this is not true when we compare continuously changing numerical values. Suppose, for instance, a durative action $\widetilde{A}$ requires $x < 10$ as an invariant, but due to a continuous effect of an executing durative action $\widetilde{B}$, x is linearly increasing from 5 at a rate of 1. So, five time units later, the invariant of $\widetilde{A}$ is violated, yet no action happens at this time point! As a result, it seems that we need a stronger mechanism to protect the invariants from being violated by continuous numerical changes.

Our proposal here is to schedule a force stop at the time point where any invariant condition is predicted to get violated. Since the semantics of PDDL, as defined by Fox and Long, requires the invariant condition to hold in the *open* interval of the duration, ending the durative action on the first spot of a violation ensures that all the conditions hold at least before this time point. In this way, we argue, the invariant condition will be protected.

Technically, this idea is implemented with coercive actions as discussed in Section 5.6. To be specific, we add the following obligatory condition to each end event:

$$Obli(end(\widetilde{A}, t)) \subset Performing(\widetilde{A}) \land \neg Eval[\pi^{o}_{\widetilde{A}}, t] \tag{6.26}$$

which means, whenever the invariant condition $\pi^{o}_{\widetilde{A}}$ does not hold, the action $\widetilde{A}$ is forced to stop.

To see how this helps protect the invariant conditions with continuous fluents, suppose $x = linear(5, 1, 0)$ and the condition of an active durative action $\widetilde{A}$ requires $x < 10$ as an invariant condition. In this case, we have $Obli(end(\widetilde{A}, 5))$, and therefore $\widetilde{A}$ cannot keep running after time 5 to get its invariant condition violated.

It gets even more interesting if we change the invariant of $\widetilde{A}$ from $x < 10$ to $x \leq 10$. In this case, we do not have $Obli(end(\widetilde{A}, 5))$ any more, since $eval(x, 5) = 10$ and still satisfies $\pi^{o}_{\widetilde{A}}$. Nevertheless, we argue that the end event have to occur at latest at time 5. The reason is that $Obli(end(\widetilde{A}, t))$ holds for all $t > 5$. If any action C wants to happen at time $t > 5$, there always exists $5 < t' < t$ satisfying $Obli(end(\widetilde{A}, t'))$, and thus the happening of C without $end(\widetilde{A}, t')$ is not executable. Moreover, even $end(\widetilde{A}, t')$ itself is not possible, since there is another $0 < t'' < t'$ such that $Obli(end(\widetilde{A}, t''))$, and so on. So this forces an $end(\widetilde{A}, t)$ event to happen at some time where $t \leq 5$.

It can be seen, now, that our mechanism successfully protects the invariant condition in the presence of continuous changing effects. One may further ask, whether we can solely use (6.26) for the protection of the invariant conditions, and abandon the idea of protecting them with disallowing possibly violating actions. Unfortunately, we cannot. The reason is that for condition changes in the discretized case, all the changes are made by some actions or events. That means, an action must have occurred and violated an invariant condition, before the $Obli(\cdots)$ gets activated. Although the *end* events happens immediately afterward, or in fact, they happen simultaneously in concept, there is a situation with duration 0 where the invariant condition is violated. As a result, the mechanism in (6.26) fails in the case of discretized effects.

So, the conclusion is that (6.19) and (6.26) together protects the invariant constraints in the presence of both the discrete and continuous effects.

6.4 Timed Initial Literals

In PDDL 2.2, Edelkamp and Hoffmann introduced the concept of timed initial literals, in order to model simple deterministic exogenous events, such as the time window in which a shop is open [EH04]. In this section, we shall concentrate on how they can be mapped to the basic action theory in $\mathcal{ES}$.

The use of timed initial literal is to specify the truth value of a predicate atom, not in the initial situation, but at a certain time point in the future (later than time 0). In $\mathcal{ES}$, the domain evolves in a way defined in the basic action theories, and it is not a standard method to change the truth value of any formula directly with an external "command".

To solve this problem, we resort to coercive actions, which is defined in Section 5.6, since it was introduced to $\mathcal{ES}$ as a standard way to model predetermined exogenous events.

In order to make use of coercive actions, we need to introduce a new action $A_{\langle t,\varphi\rangle}$ that is unique to any existing action symbol in the domain, for each timed initial literal $\langle t,\varphi\rangle$. Both the precondition and the obligatory condition of $A_{\langle t,\varphi\rangle}$ is that the time is exactly the same as specified in the PDDL description, formally,

$$\Box Poss(A_{\langle t,\varphi\rangle}) \equiv time(A_{\langle t,\varphi\rangle}) = t \tag{6.27}$$

$$\Box Obli(A_{\langle t,\varphi\rangle}) \equiv time(A_{\langle t,\varphi\rangle}) = t \tag{6.28}$$

Furthermore, the only effect of $A_{\langle t,\varphi\rangle}$ is to change the truth value of φ to $True$.

As an example, suppose that we have the timed initial literals

```
(:init
  (at 9 (shop-open))
  (at 20 (not (shop-open)))
)
```

Then, we introduce two actions *open_shop* and *close_shop*, with the following axioms for them:

$$
\begin{aligned}
&\Box Poss(open\_shop(t)) \equiv t = 9\\
&\Box Poss(close\_shop(t)) \equiv t = 20\\
&\Box Obli(open\_shop(t)) \equiv t = 9\\
&\Box Obli(close\_shop(t)) \equiv t = 20\\
&\Box [a] ShopOpen \equiv\\
&\quad \exists t.a = open\_shop(t) \vee\\
&\quad ShopOpen \wedge \neg \exists t.a = close\_shop(t)
\end{aligned}
$$

With the help of these two actions, it is guaranteed that the shop gets open at time 9, and becomes closed at time 20, and therefore the timed initial literals are realized.

Chapter 7

Correctness

In Chapter 6, we built a semantic mapping between the PDDL problem description and the basic action theories in the logic $\mathcal{ES}$. Now, we shall prove the correctness of this mapping. We do so in an incremental way, starting from the result of the ADL subset obtained by Claßen *et al.* [CELN07]. Extending their result, we first reconsider the meaning of a *state* in the presence of numerical and temporal properties (Section 7.1). Then we justify, in turn, our treatments of the numerics (Section 7.2), durative actions (Section 7.3) and timed initial literals (Section 7.4).

7.1 The Interpretation of the State

In the ADL subset of PDDL, a state is characterized solely by the truth values of logical atoms. With the introduction of numerical functional fluents, however, a state is determined not only by such a logical state, but also by the values of fluent functions. As a result, we need to extend the interpretation of a state with functional properties.

Definition 7.1. A state I is a tuple $\langle I_P, I_f \rangle$, where I_P is the logical state description, comprised of a finite set of ground atomic predicates that are true in the state, and I_f is the functional state description, comprised of a finite set of formulas with the form $f(\vec{o}) = r$, where $f(\vec{o})$ is a ground function term and r is a ground term in the domain.

Notice that for simplicity, we only consider the closed-world case here, since open-world planning is rarely used in practice (Fox and Long's PDDL 2.1 even does not consider the open-world case) and including it may cause subtle problems in our discussion.

Remember that in Fox and Long's formalism, a state is represented by a triple in $\langle \mathbb{R}, \mathbb{P}(Atms_I), \mathbb{R}_{\perp}^{dim} \rangle$, where the first component is the time of the state, the second is the logical state and the third is the mapping from the primitive numerical expressions to their values. Although their definition seems distinct from ours, we argue that the two are indeed equivalent. To see why, remember that we use the term *now* to represent the time of the current situation in $\mathcal{ES}$, which is similar to the first component in Fox and Long's definition, but is actually a 0-ary functional fluent and thus belongs to I_f according to our definition.

Besides, instead of considering I_f as a mapping from functional terms to their corresponding values, we choose to represent them as a set of formulas. It is easy to see that, under the consistency condition that at most one formula exists to assert the value of a ground functional term, the two interpretations are equivalent. However, with ours, the axiomatizations of the logical state and the functional state become similar.

Now, let us consider the mapping between a Pddl state and its corresponding initial theory. For all the predicates and functions in the Pddl domain, there are counterparts in the basic action theory. However, the reverse is not true: there are additional predicates and functions in the basic action theories, yet they do not appear in the Pddl domain. These properties include, for example, *now*, $Performing(a)$, $\xi^s_{j,i}(\vec{p_j}, \vec{q_i})$, $f^s_{j,i}(\vec{p_j}, \vec{q_i})$, and so on. To differentiate between these two groups of properties, we have the following definition.

Definition 7.2. A *target property* is a property that is defined in the problem domain definition; an *auxiliary property* is a (process-related) property that is not mentioned in the domain definition but defined in the basic action theory. The set of target properties is denoted by $I_P^{(T)}$ and $I_f^{(T)}$ (respectively for predicates and functions), and the set of auxiliary properties is denoted by $I_P^{(A)}$ and $I_f^{(A)}$.

For example, in the vehicle domain, $At(x, y)$ and $fuel\_level(x)$ are target properties, whereas $Performing(a)$, *since*(a) and *now* are auxiliary ones.

To build a direct correspondence between the Pddl state and the initial theory, we map the auxiliary properties in the basic action theory to the Pddl domain, which determine the so-called *meta-state* of a state.

Definition 7.3. A *meta-state* in a Pddl domain, characterized by $\langle I_P^{(A)}, I_f^{(A)} \rangle$, is an extended state description that determines the process-related properties of a state, including

- the time of the state;
- whether each durative action is in progress;

- since when each durative action is in progress;
- the value of fluent functions used in the duration constraint at the start of a durative action;
- the truth value of premise formula for conditional effects at the start and during a durative action;
- the changing rate of continuous fluents.

Recall that the need for defining the auxiliary properties in the basic action theory comes from the limitation of the logic $\mathcal{ES}$ (as well as the situation calculus) that only non-durative actions can be directly represented in it. They are proposed to model the process-related properties. Since the meta-state is obtained from the auxiliary properties that are directly mapped from the basic action theory, they should correctly model the evolution of the processes as well. Theorem 7.4 proves this correctness.

Theorem 7.4. *Suppose we have a plan P, containing m happenings of simple actions and n happenings of durative actions:*

$$\begin{aligned} P \;=\; & \{\big(t_1 : A_1(\vec{o_1})\big), \cdots, \big(t_m : A_m(\vec{o_m})\big)\} \cup \\ & \{\big(s_1 : \widetilde{A}_1(\vec{p_1})[d_1]\big), \cdots, \big(s_n : \widetilde{A}_n(\vec{p_n})[d_n]\big)\} \end{aligned}$$

Then, for a state after some happening at time t, the auxiliary properties determining its meta-state satisfy:

- *now has the same value as t;*
- *$Performing\big(\widetilde{A}_j(\vec{p_j})\big)$ is true if and only if there exists some happening $\big(s : \widetilde{A}_j(\vec{p_j})[d]\big)$ in plan P, satisfying $s \le t < s + d$;*
- *$since\big(\widetilde{A}_j(\vec{p_j})\big) = s$ if there exists some happening $\big(s : \widetilde{A}_j(\vec{p_j})[d]\big)$ in plan P, satisfying $s \le t < s + d$;*
- *$f^s_{j,i}(\vec{p_j}, \vec{q_i})$ has the value of $f_i(\vec{q_i})$ at time s, if $f_i(\vec{q_i})$ appears in the start duration constraint of $\widetilde{A}_j(\vec{p_j})$ and the happening $(s : \widetilde{A}_j(\vec{p_j})[d])$ is in P, satisfying $s \le t < s + d$;*
- *If $\varphi^s_{j,i}$ appears in the premise of a start-end effect in $\widetilde{A}_j(\vec{p_j})$, and the happening $(s : \widetilde{A}_j(\vec{p_j})[d])$ is in P, satisfying $s \le t < s+d$, then $\xi^s_{j,i}(\vec{p_j}, \vec{q_i})$ is true iff $\varphi^s_{j,i}$ is true at time s, where $\vec{q_i}$ are all the free variables in the effect formula that do not appear in $\vec{p_j}$;*

- *If $\varphi^o_{j,i}$ appears in the premise of an overall-end effect in $\widetilde{A}_j(\vec{p_j})$, and the happening $(s : \widetilde{A}_j(\vec{p_j})[d])$ is in P, satisfying $s \leq t < s+d$, then $\xi^s_{j,i}(\vec{p_j}, \vec{q_i})$ is true iff for all happening time t' where $s < t' \leq t$, $\varphi^o_{j,i}$ is true at time t';*
- *The changing rate of the functional fluent f_i is determined by*

$$rate\big(f_i(\vec{q_i})\big) = \Bigg(\sum_{\substack{(s:\widetilde{A}_j(\vec{p_j})[d])\in P\wedge s\leq t<s+d \\ \langle +,f_i(\vec{q_i}),Q_{j,i}\rangle\in\epsilon^o_{\widetilde{A}_j}}} Q_{j,i}\Bigg) - \Bigg(\sum_{\substack{(s:\widetilde{A}_j(\vec{p_j})[d])\in P\wedge s\leq t<s+d \\ \langle -,f_i(\vec{q_i}),Q_{j,i}\rangle\in\epsilon^o_{\widetilde{A}_j}}} Q_{j,i}\Bigg)$$

Proof. These conclusions are obvious from the construction of the successor state axioms discussed in Chapter 6 □

Theorem 7.4 establishes the correspondence between the auxiliary properties in the basic action theory and the process-related properties in the PDDL state with respect to the specific plan. When we later talk about the correctness of the mapping of durative actions, we shall mainly concentrate on proving the equivalence between the standard definition with meta-theoretical structures by Fox and Long and ours with meta-states. But before that, let us first hook the semantics of the non-durative subset, *i.e.* the subset of PDDL with the requirements `:adl` and `:fluent`.

7.2 The Non-Durative Subset

In Section 6.1.2, we have introduced the conclusion by Claßen *et al.* that their declarative semantics for the ADL subset of PDDL is correct. They proved this conclusion by relating the state updates through PDDL action operators to first-order progression in $\mathcal{ES}$. As mentioned above, their result covers purely logical properties in I_P. With the introduction of fluent functions in Section 6.2, a state is also characterized by the function values in I_f. So we shall extend their correctness proof to incorporate the functional fluents.

The basic idea of the proof is very similar. In the first step, we simply show that, for a basic action theory that is a translation from a PDDL problem description, there exists a correspondence between the relational and functional updates in the state-transitional semantics and first-order progression in $\mathcal{ES}$. However, this is not enough, since PDDL plans, in general, allow concurrent happening of actions in them. As a result, we need to show that even with concurrent happenings, our semantics is equivalent to the standard one. This is done by further showing that the two semantics share the same set of valid plans.

First, in the following Lemma 7.5 and Corollary 7.6, we formalize some simple consequences of the way we construct our basic action theory from the Pddl description. They form an extension to Lemma 6.4 in the Adl subset with function symbols.

Lemma 7.5. *Let I be a* Pddl *state and Σ_0 be the initial theory obtained from I by the construction in Section 6.2.1. Futher, let $\vec{o}$ be object parameters for the fluent ϕ_j (standing for either a fluent predicate F_j or a fluent function f_j), and let α be a precondition formula*[1]*. Then*

1. *$\Sigma_0(I) \wedge \tau_{\phi_j}(\vec{o})$ is satisfiable iff $\vec{o}$ are of the correct types for ϕ_j in the* Pddl *state I.*
2. *$\Sigma_0(I) \cup \Sigma_{num} \models \alpha$ iff α is satisfied in the* Pddl *state I.*

Proof. [2] The first thing to notice is that the translation result Σ_0 (and thus the complete Σ) is satisfiable iff the original Pddl problem is consistent.[3] In the following we will however always assume a consistent Pddl description and therefore a satisfiable Σ_0.

Next, we will also use the property that the set of all sentences of the form (6.8), (6.10) and (6.11) in Σ_0 is equivalent to a finite set of sentences of the form (6.10), including one for the *Object* type. This can easily be seen by the fact that we assumed type definitions to be non-circular; thus we can always "flatten" sentences (6.8) and (6.11) by recursively substituting all $\tau_i(x)$ in their right-hand side by the right-hand side of (6.8) or (6.10) of τ_i. Equivalently, we assume that the Pddl problem description explicitly assigns each object to all the types it belongs to, without making the "detour" of defining the supertypes by means of "either" statements.

1. To show: $\Sigma_0 \wedge \tau_{\vec{\phi_j}}(\vec{o})$ is satisfiable iff the $\vec{o}$ are of the correct types for ϕ_j in the Pddl problem. (ϕ_j here stands either for a function f_j or for a predicate F_j.)

   The "only if" direction:

[1] *Recall that in Section 2.3, we defined* precondition formula *as a first-order formula constructed with only typing, predicate, function and typed object symbols.*

[2] The proof of this lemma, as well as those for Corollary 7.6 and Theorem 7.7, is adapted from the unpublished proofs of [CELN07], with minor extensions for functional fluents.

[3] In closed worlds, inconsistencies can only appear when the tuples in (6.7) or (6.16) do not comply with the type restrictions and definitions in (6.8)–(6.11), (6.12).

Let $w \models \Sigma_0 \wedge \tau_{\vec{\phi_j}}(\vec{o})$ and let $\tau_{j_i}(o_i)$ be one of the conjuncts in $\tau_{\vec{\phi_j}}(\vec{o})$. By assumption, there is some sentence $\tau_{j_i}(x) \equiv (x = o_{j_1} \vee \cdots \vee x = o_{j_{k_i}})$ in Σ_0 such that $w \models \tau_{j_i}(x) \equiv (x = o_{j_1} \vee \cdots \vee x = o_{j_{k_i}})$. Since also $w \models \tau_{j_i}(o_i)$, it must hold that o_i is one of $o_{j_1}, \cdots, o_{j_{k_i}}$. Therefore, o_i is of type τ_{j_i} according to the PDDL problem.

The "if" direction:

Let $\vec{o}$ be of the correct types for ϕ_j according to the PDDL problem. Since Σ_0 is satisfiable, there is some world w with $w \models \Sigma_0$. Let τ_{j_i} be the type for the i-th argument of ϕ_j and $\tau_{j_i}(o_i)$ the respective conjunct in $\tau_{\vec{\phi_j}}(\vec{o})$. By assumption, there is some sentence $\tau_{j_i}(x) \equiv (x = o_{j_1} \vee \cdots \vee x = o_{j_{k_i}})$ in Σ_0 such that $w \models \tau_{j_i}(x) \equiv (x = o_{j_1} \vee \cdots \vee x = o_{j_{k_i}})$ and o_i is one of the $o_{j_1}, \cdots, o_{j_{k_i}}$ defined to be of type τ_{j_i} in the PDDL problem. But then $w \models \tau_{j_i}(o_i)$.

2. To show: $\Sigma_0(I) \cup \Sigma_{num} \models \alpha$ iff $I \models_{PDDL} \alpha$[4], the proof is by induction on the structure of α.

   - $\alpha = (o_1 = o_2)$ for objects o_1 and o_2:

     $$\Sigma_0 \cup \Sigma_{num} \models (o_1 = o_2)$$

     iff (by definition)

     $$\text{for all worlds } w \text{ with } w \models \Sigma_0 \cup \Sigma_{num},\ w \models (o_1 = o_2)$$

     iff (due to the following two facts: Σ_0 is satisfiable, and so is $\Sigma_0 \cup \Sigma_{num}$, by definition, there is at least one such world; besides, the truth value of equations between ground terms is independent of the interpreting world)

     $$o_1 \text{ and } o_2 \text{ are identical constants}$$

     iff (unique names assumption in PDDL)

     $$I \models_{PDDL} (o_1 = o_2)$$

   - $\alpha = (e_1 \otimes e_2)$ where e_i are numerical expressions and $\otimes \in \{=, <\}$: First, notice that only numbers and primitive functions may appear as terms in e_i. In PDDL, this is guaranteed by the standard semantics; in Σ_0, this is ensured by the typing of domain elements. So, suppose $f_1(\vec{q_1}), \cdots, f_m(\vec{q_m})$ are all function terms in e_i. Then, for any number t_l,

     $$\Sigma_0 \models f_l(\vec{q_l}) = t_l$$

[4] We use the notation $I \models_{PDDL} \alpha$ to mean that α is satisfied in the PDDL state I.

iff (since $\Sigma_0 \models (6.16)$)

$$\Sigma_0 \models (\vec{q_l} = \vec{o_1} \wedge t_l = r_1 \vee \cdots \vee \vec{q_l} = \vec{o_{k_j}} \wedge t_l = r_{k_j})$$

iff (since Σ_0 is satisfiable and equality between ground terms is world-independent)

$$\models (\vec{q_l} = \vec{o_1} \wedge t_l = r_1 \vee \cdots \vee \vec{q_l} = \vec{o_{k_j}} \wedge t_l = r_{k_j})$$

iff

$$\langle \vec{q_l}, t_l \rangle \text{ is one of } \langle \vec{o_1}, r_1 \rangle, \cdots, \langle \vec{o_{k_j}}, r_{k_j} \rangle$$

iff (by construction)

$$\big(f_l(\vec{q_l}) = t_l\big) \in I_f$$

iff

$$I \models_{PDDL} \big(f_l(\vec{q_l}) = t_l\big)$$

Meanwhile, for any numerical formulas e_1' and e_2', where e_i' is constructed with only symbols on numbers, $+$ and $*$,

$$\Sigma_{num} \models e_1' \otimes e_2'$$

iff

$$e_1' \otimes e_2' \text{ holds in mathematics}$$

iff

$$I \models_{PDDL} e_1' \otimes e_2'$$

As a result, with the additional fact that $\Sigma_0 \cup \Sigma_{num}$ is satisfiable,

$$\Sigma_0 \cup \Sigma_{num} \models e_1{}_{t_l}^{f_l(\vec{q_l})} \otimes e_2{}_{t_l}^{f_l(\vec{q_l})}$$

iff (combining the above two conclusions)

$$I \models_{PDDL} e_1{}_{t_l}^{f_l(\vec{q_l})} \otimes e_2{}_{t_l}^{f_l(\vec{q_l})}$$

And finally, $\Sigma_0 \cup \Sigma_{num} \models e_1 \otimes e_2$ iff $I \models_{PDDL} e_1 \otimes e_2$.

- $\alpha = F_j(\vec{o})$:

$$\Sigma_0 \cup \Sigma_{num} \models F_j(\vec{o})$$

iff (since $\Sigma_0 \models (6.7)$ and $\Sigma_0 \cup \Sigma_{num}$ is satisfiable)

$$\Sigma_0 \cup \Sigma_{num} \models (\vec{o} = \vec{o_1} \vee \cdots \vee \vec{o} = \vec{o_{k_o}})$$

iff (since $\Sigma_0 \cup \Sigma_{num}$ is satisfiable and truth value of equality between ground term is world independent)

$$\models (\vec{o} = \vec{o_1} \vee \cdots \vee \vec{o} = \vec{o_{k_o}})$$

iff

$$\vec{o} \text{ is one of } \vec{o_1}, \cdots, \vec{o_{k_o}}$$

iff (by construction)

$$F_j(\vec{o}) \in I_P$$

iff

$$I \models_{PDDL} F_j(\vec{o})$$

- The cases $\phi = \phi_1 \wedge \phi_2$ and $\phi = \neg\phi_1$ follow directly by induction
- $\phi = \forall x : \tau.\phi_1$, with x being the only free variable in ϕ_1:

$$\Sigma_0 \cup \Sigma_{num} \models \forall x : \tau.\phi_1$$

iff (by definition)

$$\Sigma_0 \cup \Sigma_{num} \models \forall x.\tau(x) \supset \phi_1(x)$$

iff (by the semantics of $\mathcal{ES}$)

$$\Sigma_0 \cup \Sigma_{num} \models \big(\tau(r) \supset \phi_1\big)^x_r \qquad \text{for all ground terms } r$$

iff (since $\Sigma_0 \models$ (6.10) and $\Sigma_0 \cup \Sigma_{num}$ is satisfiable)

$$\Sigma_0 \cup \Sigma_{num} \models \bigwedge_{o_j \text{ of type } \tau} (\phi_1)^x_{o_j}$$

iff (by induction)

$$I \models_{PDDL} \bigwedge_{o_j \text{ of type } \tau} (\phi_1)^x_{o_j}$$

iff (by the substitutional interpretation of quantification of Pddl)

$$I \models_{PDDL} \forall x : \tau.\phi_1$$

□

Corollary 7.6. *Let I be a* Pddl *state, Σ_0 be the initial theory obtained from I by the construction in Section 6.2.1, $\vec{o}$ be object parameters for the fluent ϕ_j, and $A(\vec{p})$ be a action instance. Then*

1. *$\Sigma_{pre} \cup \Sigma_0 \cup \Sigma_{num} \models Poss\big(A(\vec{p})\big)$ if and only if $\pi_A(\vec{p})$ is satisfied in I and $\vec{p}$ are of correct types.*
2. *$\Sigma_0 \cup \Sigma_{num} \models \gamma^{+}{}_{F_j}{}^{\vec{x_j}}_{\vec{o}}{}^{a}_{A(\vec{p})}$ if and only if $F_j(\vec{o})$ is asserted in the effect of $A(\vec{p})$.*
3. *$\Sigma_0 \cup \Sigma_{num} \models \gamma^{-}{}_{F_j}{}^{\vec{x_j}}_{\vec{o}}{}^{a}_{A(\vec{p})}$ if and only if $F_j(\vec{o})$ is negated in the effect of $A(\vec{p})$.*
4. *$\Sigma_0 \cup \Sigma_{num} \models \gamma^{v}{}_{f_j}{}^{\vec{x_j}}_{\vec{o}}{}^{y}_{r}{}^{a}_{A(\vec{p})}$ for some r, if and only if r is the assignment result for $f_j(\vec{o})$ as an effect of $A(\vec{p})$.*

Proof.

1. To show: $\Sigma_{pre} \cup \Sigma_0 \cup \Sigma_{num} \models Poss\big(A(\vec{p})\big)$ iff $I \models_{PDDL} \pi_A(\vec{p})$ and $\vec{p}$ are of correct types:

$$\Sigma_{pre} \cup \Sigma_0 \cup \Sigma_{num} \models Poss\big(A(\vec{p})\big)$$

iff (by definition)

$$\Sigma_{pre} \cup \Sigma_0 \cup \Sigma_{num} \models \pi^{a}_{A(\vec{p})}$$

iff (by construction)

$$\Sigma_{pre} \cup \Sigma_0 \cup \Sigma_{num} \models \Big(\bigvee_{\text{operators } A_i} \exists \vec{z_i} : \vec{\tau_{A_i}}.a = A_i(\vec{z_i}) \land \pi_{A_i} \Big)^{a}_{A(\vec{p})}$$

iff

$$\Sigma_{pre} \cup \Sigma_0 \cup \Sigma_{num} \models \Big(\bigvee_{\text{operators } A_i} \exists \vec{z_i} : \vec{\tau_{A_i}}.A(\vec{p}) = A_i(\vec{z_i}) \land \pi_{A_i} \Big)$$

iff (by definition)

$$\Sigma_{pre} \cup \Sigma_0 \cup \Sigma_{num} \models \Big(\bigvee_{\text{operators } A_i} \exists \vec{z_i}.\vec{\tau_{A_i}}(\vec{z_i}) \land A(\vec{p}) = A_i(\vec{z_i}) \land (\pi_{A_i})^{\vec{z}}_{\vec{p}} \Big)$$

iff (by unique names in $\mathcal{ES}$, only the A-conjunct applies)

$$\Sigma_{pre} \cup \Sigma_0 \cup \Sigma_{num} \models \vec{\tau_A}(\vec{p}) \land (\pi_A)^{\vec{z_i}}_{\vec{p}}$$

iff

$$\Sigma_{pre} \cup \Sigma_0 \cup \Sigma_{num} \models \vec{\tau_A}(\vec{p}) \text{ and } \Sigma_{pre} \cup \Sigma_0 \cup \Sigma_{num} \models (\pi_A)^{\vec{z_i}}_{\vec{p}}$$

iff (by Lemma 7.5)

$\vec{p}$ are of the correct types for A according to the Pddl problem description and $I \models_{PDDL} \pi_A(\vec{p})$

2. To show: $\Sigma_0 \cup \Sigma_{num} \models \gamma^{+\;\vec{x_j}\;a}_{F_j\;\vec{o}\;A(\vec{p})}$ if and only if $F_j(\vec{o})$ is asserted in the effect of $A(\vec{p})$:
$$\Sigma_0 \cup \Sigma_{num} \models \gamma^{+\;\vec{x_j}\;a}_{F_j\;\vec{o}\;A(\vec{p})}$$
iff (by construction)
$$\Sigma_0 \cup \Sigma_{num} \models \Big(\bigvee_{\gamma^{+}_{F_j,A_i} \in NF(A_i)} \exists \vec{z_i}.a = A_i(\vec{z_i}) \wedge \gamma^{+}_{F_j,A_i} \Big)^{\vec{x_j}\;a}_{\vec{o}\;A(\vec{p})}$$
iff
$$\Sigma_0 \cup \Sigma_{num} \models \bigvee_{\gamma^{+}_{F_j,A_i} \in NF(A_i)} \exists \vec{z_i}.A(\vec{p}) = A_i(\vec{z_i}) \wedge (\gamma^{+}_{F_j,A_i})^{\vec{x_j}\;\vec{z_i}}_{\vec{o}\;\vec{p}}$$
iff (by unique names in $\mathcal{ES}$, only the A-conjunct applies)
$$\Sigma_0 \cup \Sigma_{num} \models (\gamma^{+}_{F_j,A})^{\vec{x_j}\;\vec{z_i}}_{\vec{o}\;\vec{p}}$$
iff (by Lemma 7.5)
$$I \models_{PDDL} (\gamma^{+}_{F_j,A})^{\vec{x_j}\;\vec{z_i}}_{\vec{o}\;\vec{p}}$$
iff (by construction)

$F_j(\vec{o})$ is an effect of $A(\vec{p})$

3. To show: $\Sigma_0 \cup \Sigma_{num} \models \gamma^{-\;\vec{x_j}\;a}_{F_j\;\vec{o}\;A(\vec{p})}$ if and only if $F_j(\vec{o})$ is negated in the effect of $A(\vec{p})$:

Similar to 2.

4. To show: $\Sigma_0 \cup \Sigma_{num} \models \gamma^{v\;\vec{x_j}\;y\;a}_{f_j\;\vec{o}\;r\;A(\vec{p})}$ for some r, if and only if r is the assignment result for $f_j(\vec{o})$ as an effect of $A(\vec{p})$:
$$\Sigma_0 \cup \Sigma_{num} \models \gamma^{v\;\vec{x_j}\;y\;a}_{f_j\;\vec{o}\;r\;A(\vec{p})}$$
iff (by construction)
$$\Sigma_0 \cup \Sigma_{num} \models \Big(\bigvee_{\gamma^{v}_{f_j,A_i} \in NF(A_i)} \exists \vec{z_i}.a = A_i(\vec{z_i}) \wedge \gamma^{v}_{f_j,A_i} \Big)^{\vec{x_j}\;y\;a}_{\vec{o}\;r\;A(\vec{p})}$$

iff

$$\Sigma_0 \cup \Sigma_{num} \models \bigvee_{\gamma^v_{f_j,A_i} \in NF(A_i)} \exists \vec{z_i}.A(\vec{p}) = A_i(\vec{z_i}) \wedge (\gamma^v_{f_j,A_i})^{\vec{x_j}\ y\ \vec{z_i}}_{\vec{o}\ \ r\ \vec{p}}$$

iff (by unique names in $\mathcal{ES}$, only the A-conjunct applies)

$$\Sigma_0 \cup \Sigma_{num} \models (\gamma^v_{f_j,A})^{\vec{x_j}\ y\ \vec{z_i}}_{\vec{o}\ \ r\ \vec{p}}$$

iff (by Lemma 7.5)

$$I \models_{PDDL} (\gamma^v_{f_j,A})^{\vec{x_j}\ y\ \vec{z_i}}_{\vec{o}\ \ r\ \vec{p}}$$

iff (by construction)

$$A(\vec{p}) \text{ assigns to } f_j(\vec{o}) \text{ the value } r.$$

□

With Corollary 7.6, it is easy to construct the new state I' that is updated from $I = \langle I_P, I_f\rangle$ through action operator $A(\vec{p})$. Suppose $\Sigma = \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_0$ is the basic action theory obtained from the PDDL description, and Σ_{num} is the one for axiomatizing numbers. Under the condition $\Sigma_{pre} \cup \Sigma_0 \cup \Sigma_{num} \models Poss\big(A(\vec{p})\big)$, we define four sets $Adds_P$, $Dels_P$, $Adds_f$ and $Dels_f$, all of which are initialized to the empty set and expanded with the following procedure:

1. for all objects $\vec{o}$ and all fluent predicates F_j such that $F_j(\vec{o})$ is type-consistent, if $\Sigma_0 \cup \Sigma_{num} \models \gamma^{+\ \vec{x_j}\ \ a}_{F_j\ \ \vec{o}\ \ A(\vec{p})}$, then add $F_j(\vec{o})$ to the set $Adds_P$;
2. for all objects $\vec{o}$ and all fluent predicates F_j such that $F_j(\vec{o})$ is type-consistent, if $\Sigma_0 \cup \Sigma_{num} \models \gamma^{-\ \vec{x_j}\ \ a}_{F_j\ \ \vec{o}\ \ A(\vec{p})}$, then add $F_j(\vec{o})$ to the set $Dels_P$;
3. for all objects $\vec{o}$ and all fluent functions f_j such that $f_j(\vec{o})$ is type-consistent, if for some r, $\Sigma_0 \cup \Sigma_{num} \models \gamma^{v\ \vec{x_j}\ y\ \ a}_{f_j\ \ \vec{o}\ \ r\ A(\vec{p})}$, then
   - for any $r_0 \neq r$ such that $\Sigma_0 \models f_j(\vec{o}) = r_0$ add $f_j(\vec{o}) = r_0$ to the set $Dels_f$;
   - add $f_j(\vec{o}) = r$ to the set $Adds_f$.

Then the new state description is

$$I' = \langle I'_P, I'_f\rangle$$

where

$$\begin{cases} I'_P = (I_P \setminus Dels_P) \cup Adds_P \\ I'_f = (I_f \setminus Dels_f) \cup Adds_f \end{cases}$$

Here, all the symbols we consider (objects, fluents, operators) are those that appear in the PDDL problem definition. Since there are only a finite number of them, the combination is also finite. The fact that we only check type consistent instances of functions and predicates further restricts the number of atoms to consider. The only potential problem lies in the numbers that appear in the construction. One may ask whether the numbers satisfying the formulas can be found in finite time, since there are countably infinitely many of them. However, we argue that we do not rely on a substitutional method for obtaining their values. In fact, numbers only appear in two special forms. In the case of r_0, it is an existential account, so we only need to see whether $\vec{o}$ coincides with some $\vec{o_j}$ in a finite disjunction in (6.16); for r, remember that in $\gamma^v_{f_j}$, the free variable y always appear at the left-hand side of equations, so the value r can be efficiently calculated with little computational effort by evaluating the right-hand side of the equation. So it is guaranteed that the four sets above are finite, and can always be easily obtained in finite time with the construction above.

We then have the following theorem, saying that a basic action theory obtained from the state description I' is a progression of the original theory I.

Theorem 7.7. *Let I' be the state description obtained from the construction above applied to a given* PDDL *problem description and a ground simple action $r = A(\vec{p})$. Further let*

$$\Sigma_r = \{[r]\psi | \psi \in \Sigma_0(I')\}$$

where $\Sigma_0(I')$ means the result of constructing the initial state axiom from I' instead of I. For all fluent predicates F_k and all fluent functions f_l in the problem description, let the consistency conditions

$$\Sigma_0 \cup \Sigma_{num} \models \neg(\gamma^+_{F_k} \wedge \gamma^-_{F_k})^a_r$$

and

$$\Sigma_0 \cup \Sigma_{num} \models (\forall \vec{o}, t_1, t_2).\big(\gamma^v_{f_l}{}^{\vec{x}\,y}_{\vec{o}\,t_1} \wedge \gamma^v_{f_l}{}^{\vec{x}\,y}_{\vec{o}\,t_2}\big)^a_r \supset (t_1 = t_2)$$

hold. Then Σ_r is a progression of Σ_0 through r.

In order to prove this theorem, we first need the following lemma:

Lemma 7.8. *Let $w \models \Sigma_0 \cup \Sigma_{num}$. Then, for any fluent formula ϕ that only mentions fluent symbols in $\mathcal{F} \setminus \{Poss\}$*

$$w \models \phi \text{ iff } \Sigma_0 \cup \Sigma_{num} \models \phi$$

Proof. The proof is given by induction on the structure of ϕ. Remember that due to the typing constraints in Σ_0, parameters to predicates and functions can only be ground terms of type *Object* or its subtype.

- $w \models f_j(\vec{o}) = t$ iff $\langle \vec{o}, t\rangle$ is one of $\langle \vec{o_1}, r_1\rangle, \cdots, \langle \vec{o_{k_j}}, r_{k_j}\rangle$ in (6.16) iff for every world w' with $w' \models \Sigma_0 \cup \Sigma_{num}$,$w' \models f_j(\vec{o}) = t$.
- $w \models F_j(\vec{o})$ iff $\vec{o}$ is one of $\vec{o_1}, \cdots, \vec{o_{k_o}}$ in (6.7) iff for every w' with $w' \models \Sigma_0 \cup \Sigma_{num}$, $w' \models F_j(\vec{o})$.
- $w \models \tau_i(t)$ iff (using the "type flattening" properties in the proof of Lemma 7.5) t is one of the $o_{j_1}, \cdots, o_{j_{k_j}}$ of type τ_i in (6.10) iff for every w' with $w' \models \Sigma_0 \cup \Sigma_{num}$, $w' \models \tau_i(t)$.
- the cases for $\phi_1 \wedge \phi_2$ and $\neg\phi_1$ follows directly by induction.
- $w \models \forall x.\phi_1$ iff $w \models {\phi_1}^x_r$ for all ground terms r iff (by induction) $\Sigma_0 \cup \Sigma_{num} \models {\phi_1}^x_r$ for all ground terms r iff for all ground term r and all world w' with $w' \models \Sigma_0 \cup \Sigma_{num}$, $w' \models {\phi_1}^x_r$ iff for all w' with $w \models \Sigma_0 \cup \Sigma_{num}$, $w' \models \forall x.\phi_1$.

□

Proof of Theorem 7.7. To show that Σ_r is a progression of Σ_0 through r, we need to prove that Σ_r satisfies the three properties of progression defined in Definition 4.2.

1. All sentences in Σ_r are fluent in $\langle r\rangle$
   Obvious by the construction of Σ_r.
2. $\Sigma \models \Sigma_r$, where $\Sigma = \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_0 \cup \Sigma_{num}$:
   - For (6.8), $\Sigma \models [r]\tau_i(x) \equiv \big(\tau_{i_1}(x) \vee \cdots \vee \tau_{i_{k_k}}(x)\big)$:
     follows directly from (6.6) and (6.8).
   - For (6.9), $\Sigma \models [r]F(x_{j_1}, \cdots, x_{j_{k_j}}) \supset \big(\tau_{j_1}(x_{j_1}) \wedge \cdots \wedge \tau_{j_{k_j}}(x_{j_{k_j}})\big)$:
     let $w \models \Sigma$ and $w \models [r]F_j(\vec{t})$. Then, since $w \models$ (6.5), we have

     $$w \models \big(\gamma^+_{F_j} \wedge \tau_{\vec{F_j}}(\vec{x_j}) \vee F_j(\vec{x_j}) \wedge \neg\gamma^-_{F_j}\big)^{a\ \vec{x_j}}_{r\ \vec{t}}$$

     therefore, we need only to consider the following two cases:

     a) $w \models \big(\gamma^+_{F_j} \wedge \tau_{\vec{F_j}}(\vec{x_j})\big)^{a\ \vec{x_j}}_{r\ \vec{t}}$:
        We have $w \models \tau_{\vec{F_j}}(\vec{x_j})$, and since $w \models$ (6.6), $w \models [r]\tau_{\vec{F_j}}(\vec{x_j})$.

b) $w \models \big(F_j(\vec{x_j}) \wedge \neg\gamma^-_{F_j}\big)^{a\,\vec{x_j}}_{r\,\vec{t}}$:
$w \models F_j(\vec{x_j})$, so since $w \models$ (6.9), $w \models \tau_{\vec{F_j}}(\vec{x_j})$. Similar to the previous case, we have $w \models [r]\tau_{\vec{F_j}}(\vec{x_j})$

- For (6.10), $\Sigma \models [r]\tau_i(x) \equiv (x = o_{j_1} \vee \cdots \vee x = o_{j_{k_i}})$: follows directly from (6.6) and (6.10).
- For (6.11), $\Sigma \models [r]Object(x) \equiv \big(\tau_1(x) \vee \cdots \vee \tau_l(x)\big)$: follows directly from (6.6) and (6.11).
- For (6.12),

$$\Sigma \models [r]\big(f_j(x_{j_1}, \cdots, x_{j_{k_j}}) = y\big) \supset \\ \big(\tau_{j_1}(x_{j_1}) \wedge \cdots \wedge \tau_{j_{k_j}}(x_{j_{k_j}}) \wedge Number(y)\big) :$$

let $w \models \Sigma$ and $w \models [r]f_j(\vec{o}) = t$. Then, since $w \models$ (6.15), we have

$$w \models \big(\gamma^v_{f_j} \wedge \tau_{f_j}(\vec{x_j}) \wedge Number(y) \vee f_j(\vec{x_j}) = y \wedge \neg\gamma_{f_j}\big)^{a\,\vec{x_j}\,y}_{r\,\vec{o}\,\,t}$$

Like the case for fluent predicates, we need to consider the following two cases:

a) $w \models \big(\gamma^v_{f_j} \wedge \tau_{f_j}(\vec{x_j}) \wedge Number(y)\big)^{a\,\vec{x_j}\,y}_{r\,\vec{o}\,\,t}$:
we have $w \models \tau_{\vec{f_j}}(\vec{x_j})$ and $w \models Number(y)$. Since $w \models$ (6.6), $w \models [r]\big(\tau_{\vec{F_j}}(\vec{x_j}) \wedge Number(y)\big)$.

b) $w \models \big(f_j(\vec{x_j}) = y \wedge \neg\gamma_{f_j}\big)^{a\,\vec{x_j}\,y}_{r\,\vec{o}\,\,t}$:
we have $w \models f_j(\vec{x_j}) = y$. Since $w \models$ (6.12), $w \models \big(\tau_{\vec{F_j}}(\vec{x_j}) \wedge Number(y)\big)$. With a similar argument as above, we get $w \models [r]\big(\tau_{\vec{F_j}}(\vec{x_j}) \wedge Number(y)\big)$.

- For (6.7),

$$\Sigma \models [r]F_j(\vec{x_j}) \equiv (\vec{x_j} = \vec{o_1} \vee \cdots \vee \vec{x_j} = \vec{o_{k_o}}) : \tag{7.1}$$

Let $w \models \Sigma$, then with (6.5), $w, \langle r\rangle \models F_j(\vec{o})$ iff

$$w \models \big(\gamma^+_{F_j} \wedge \tau_{\vec{F_j}}(\vec{x_j}) \vee F_j(\vec{x_j}) \wedge \neg\gamma^-_{F_j}\big)^{a\,\vec{x_j}}_{r\,\vec{o}}$$

So we have the following two cases:

a) $w \models \big(\gamma^+_{F_j} \wedge \tau_{\vec{F_j}}(\vec{x_j})\big)^{a\,\vec{x_j}}_{r\,\vec{o}}$
which, according to Lemma 7.8, is equivalent to

$$\Sigma_0 \cup \Sigma_{num} \models \big(\gamma^+_{F_j} \wedge \tau_{\vec{F_j}}(\vec{x_j})\big)^{a\,\vec{x_j}}_{r\,\vec{o}}$$

iff (by construction) $F_j(\vec{o}) \in Adds_P$ iff $F_j(\vec{o}) \in I'_P$

b) $w \models \big(F_j(\vec{x_j}) \land \neg\gamma^-_{F_j}\big)^{a\ \vec{x_j}}_{r\ \vec{o}}$
which, again by Lemma 7.8, is equivalent to

$$\Sigma_0 \cup \Sigma_{num} \models \big(F_j(\vec{x_j}) \land \neg\gamma^-_{F_j}\big)^{a\ \vec{x_j}}_{r\ \vec{o}}$$

iff

$$\Sigma_0 \cup \Sigma_{num} \models F_j(\vec{x_j}) \text{ and } \Sigma_0 \cup \Sigma_{num} \models \big(\neg\gamma^-_{F_j}\big)^{a\ \vec{x_j}}_{r\ \vec{o}}$$

iff (due to Lemma 7.5, and $\Sigma_0 \cup \Sigma_{num}$ is satisfiable)

$$F_j(\vec{o}) \in I \text{ and } \Sigma_0 \cup \Sigma_{num} \not\models (\gamma^-_{F_j})^{a\ \vec{x_j}}_{r\ \vec{o}}$$

iff (by construction)

$$F_j(\vec{o}) \in I \text{ and } F_j(\vec{o}) \notin \mathit{Dels}_P$$

Therefore, in both cases, $F_j(\vec{o}) \in I'_P$, meaning that $\vec{o}$ is one of $\vec{o_1}, \cdots, \vec{o_{k_o}}$ in (7.1).

- For (6.16),

$$\begin{aligned}\Sigma \models [r]f_j(\vec{x}) = y \equiv \vec{x} = \vec{o_1} \land y = r_1 \lor \\ \cdots \lor \\ \vec{x} = \vec{o_{k_j}} \land r = y_{k_j}\end{aligned} \tag{7.2}$$

Let $w \models \Sigma$, then with (6.15), $w, \langle r\rangle \models f_j(\vec{o}) = t$ iff

$$w \models \big(\gamma^v_{f_j} \land \tau_{f_j}(\vec{x_j}) \land \mathit{Number}(y) \lor f_j(\vec{x_j}) = y \land \neg\gamma_{f_j}\big)^{a\ \vec{x_j}\ y}_{r\ \vec{o}\ \ t}$$

So we have the following two cases:

a) $w \models \big(\gamma^v_{f_j} \land \tau_{f_j}(\vec{x_j}) \land \mathit{Number}(y)\big)^{a\ \vec{x_j}\ y}_{r\ \vec{o}\ \ t}$
which, according to Lemma 7.8, is equivalent to

$$\Sigma_0 \cup \Sigma_{num} \models \big(\gamma^v_{f_j} \land \tau_{f_j}(\vec{x_j}) \land \mathit{Number}(y)\big)^{a\ \vec{x_j}\ y}_{r\ \vec{o}\ \ t}$$

iff (by construction) $\big(f_j(\vec{o}) = t\big) \in \mathit{Adds}_f$ and for any t' with $t' \neq t$, $\big(f_j(\vec{o}) = t'\big) \in \mathit{Dels}_f$
iff $\big(f_j(\vec{o}) = t\big) \in I'_f$ and for any other t', $\big(f_j(\vec{o}) = t'\big) \notin I'_f$

b) $w \models \big(f_j(\vec{x_j}) = y \land \neg\gamma_{f_j}\big)^{a\ \vec{x_j}\ y}_{r\ \vec{o}\ \ t}$
which, again by Lemma 7.8, is equivalent to

$$\Sigma_0 \cup \Sigma_{num} \models \big(f_j(\vec{x_j}) = y \land \neg\gamma_{f_j}\big)^{a\ \vec{x_j}\ y}_{r\ \vec{o}\ \ t}$$

iff

$$\Sigma_0 \cup \Sigma_{num} \models f_j(\vec{x_j}) = t \text{ and } \Sigma_0 \cup \Sigma_{num} \models (\neg\gamma_{f_j})^{a\ \vec{x_j}}_{r\ \vec{o}}$$

due to Lemma 7.5 and the fact that $\Sigma_0 \cup \Sigma_{num}$ is satisfiable, this is equivalent to

$$\big(f_j(\vec{o}) = t\big) \in I \text{ and } \Sigma_0 \cup \Sigma_{num} \not\models (\exists y.\gamma^{v\ a\ \vec{x_j}}_{f_j\, r\ \vec{o}})$$

iff (by construction)

$$\big(f_j(\vec{o}) = t\big) \in I \text{ and for any } t'\ \big(f_j(\vec{o}) = t'\big) \notin \mathit{Adds}_f \cup \mathit{Dels}_f$$

Therefore, in both cases, $\big(f_j(\vec{o}) = t\big) \in I'_f$, meaning that $\langle\vec{o}, t\rangle$ is one of $\langle\vec{o_1}, r_1\rangle, \cdots, \langle\vec{o_{k_o}}, r_{k_o}\rangle$ in (7.2).

3. For every world w_r with $w_r \models \Sigma_r \cup \Sigma_{pre} \cup \Sigma_{post}$, there exists a world w with $w \models \Sigma$, such that

$$w_r[\phi(\vec{o}), r \cdot \sigma] = w[\phi(\vec{o}), r \cdot \sigma]$$

for all $\sigma \in \mathcal{Z}$ and primitive $\phi(\vec{o})$:

First, similar to Lemma 7.8, it can be proven that for any fluent formula ϕ in $\mathcal{F} \setminus \{Poss\}$, if $w_r \models \Sigma_r \cup \Sigma_{num}$, then

$$w_r, \langle r\rangle \models \phi \text{ iff } \Sigma_r \cup \Sigma_{num} \models [r]\phi \tag{7.3}$$

Now, we construct a world w' with

- $w'[f_j(\vec{o}), \langle\rangle] = t$ iff $\big(f_j(\vec{o}) = t\big) \in I_f$
- $w'[F_j(\vec{o}), \langle\rangle] = 1$ iff $F_j(\vec{o}) \in I_P)$
- $w'[\tau_i(t), \langle\rangle] = 1$ iff t is an object of type τ_i according to the Pddl problem description

Then, we let $w = (w')_\Sigma$, *i.e.* w is like w' in the initial situation, and also satisfies $\Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_{num}$ (cf. [LL01]). Clearly, $w \models \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_0 \cup \Sigma_{num}$, so we are only left with the task to show for each primitive ϕ

$$w_r, \langle r\rangle \models \phi \text{ iff } w, \langle r\rangle \models \phi \tag{7.4}$$

Notice that the general property for all successor situations $r \cdot \sigma$ can be justified inductively with the fact that both w and w_r satisfy Σ_{post}.

- $w, \langle r \rangle \models f_j(\vec{o}) = t$
  iff (due to conclusion in Item 2 of this theorem) $\big(f_j(\vec{o}) = t\big) \in I'_f$ iff $\langle \vec{o}, t \rangle$ is one of $\langle \vec{o_1}, r_1 \rangle, \cdots, \langle \vec{o_{k_o}}, r_{k_o} \rangle$ in (7.2) in Σ_r iff $\Sigma_r \models [r]\big(f_j(\vec{o}) = t\big)$ iff (since $\Sigma_r \cup \Sigma_{num}$ is satisfiable) $\Sigma_r \cup \Sigma_{num} \models [r]\big(f_j(\vec{o}) = t\big)$ iff (according to (7.3)) $w_r, \langle r \rangle \models \big(f_j(\vec{o}) = t\big)$
- $w, \langle r \rangle \models F_j(\vec{o})$
  iff (like above) $F_j(\vec{o}) \in I_P$ iff $\vec{o}$ is one of $\vec{o_1}, \cdots, \vec{o_{k_o}}$ in (7.1) in Σ_r iff $\Sigma_r \models [r]F_j(\vec{o})$ iff (since $\Sigma_r \cup \Sigma_{num}$ is satisfiable) $\Sigma_r \cup \Sigma_{num} \models [r]F_j(\vec{o})$ iff (according to (7.3)) $w_r, \langle r \rangle \models F_j(\vec{o})$
- $w, \langle r \rangle \models \tau_i(o)$
  iff (using $w \models$ (6.6)) $w \models \tau_i(t)$ iff (using the "type flattening" property in the proof of Lemma 7.5) o is one of $o_{j_1}, \cdots, o_{j_{k_j}}$ defined to be of type τ_i iff $w, \langle r \rangle \models \tau_i(o)$
- $w, \langle r \rangle \models Poss(o)$
  iff (since $w \models \Sigma_{pre}$) $w, \langle r \rangle \models \pi_o^a$ iff (by induction over the construction of π_o^a, which is a fluent sentence) $w_r, \langle r \rangle \models \pi_o^a$ iff (using $w_r \models \Sigma_{pre}$) $w_r, \langle r \rangle \models Poss(o)$.

$\square$

Theorem 7.7 establishes the correctness of a single-step update to the initial state. An important fact is that the new state description I' has the same form as the original state I, so this theorem can be applied repeatedly through a sequence of actions in a plan, in order to prove that the two semantics share the same set of valid plans.

To proceed with the proof, we first need the following lemma, which ensures that when drawing conclusion about the future with a progressed theory, the common history may be neglected.

Lemma 7.9. *For fluent formulas ϕ and ψ, $\phi \models \psi$ if and only if for any action sequence $r_1, \cdots, r_n$, $[r_1] \cdots [r_n]\phi \models [r_1] \cdots [r_n]\psi$.*

Proof. **The "only if" direction:**

Suppose $\phi \models \psi$, let w be a world satisfying $w \models [r_1] \cdots [r_n]\phi$, and let $z = \langle r_1, \cdots, r_n \rangle$.

We construct a new world w' satisfying

$$\begin{cases} |f(\vec{n})|_{w'}^{\langle\rangle} & = & |f(\vec{n})|_w^z \\ w'[P(\vec{n}), \langle\rangle] & = & w[P(\vec{n}), z] \end{cases} \tag{7.5}$$

where $\vec{n}$ is a vector of ground terms, f is any function symbol and P is any predicate symbol in the domain. Then, for any fluent formula α, by induction on its structure, we have,

$$w'[\alpha, \langle\rangle] = w[\alpha, z] \tag{7.6}$$

Specifically $w'[\phi, \langle\rangle] = w[\phi, z]$, so $w' \models \phi$. Since $\phi \models \psi$, it immediately follows $w' \models \psi$. According to (7.6), $w[\psi, z] = w'[\psi, \langle\rangle]$. So $w \models [r_1] \cdots [r_n]\psi$.

As a result, $[r_1] \cdots [r_n]\phi \models [r_1] \cdots [r_n]\psi$.

The "if" direction:

Suppose $[r_1] \cdots [r_n]\phi \models [r_1] \cdots [r_n]\psi$, let w' be a world satisfying $w' \models \phi$, and let $z = \langle r_1, \cdots, r_n\rangle$.

We construct a new world w satisfying (7.5). Then the conclusion follows in the similar manner as in the "only if" direction.

□

Then, we define how a Pddl plan is mapped into an action sequence in the logic $\mathcal{ES}$.

Definition 7.10 (Grounded serialized plan for simple actions)**.** Let P be a plan, consisting of a finite list of happenings of simple actions.

$$\big(t_1 : A_1(\vec{p}_1)\big), \cdots, \big(t_n : A_n(\vec{p}_n)\big)$$

Without loss of generality, we assume $t_1 \leq \cdots \leq t_n$. Then, the *grounded serialized plan* of P, denoted by $[\![P]\!]$, is the syntactic structure[5]

$$[A_1(\vec{p}_1, t_1)] \cdots [A_n(\vec{p}_n, t_n)]$$

The grounded serialized plan, according to Definition 7.10, is a transformed Pddl plan in the $\mathcal{ES}$ notation. Generally speaking, it is possible to have $t_i = t_{i+1}$, because the actions $A_i(\vec{p_i})$ and $A_{i+1}(\vec{p_{i+1}})$ may happen concurrently at time t_i. In this case, the PDDL plan may have more than one serialization, since the grounded serialized plan may put $[A_i(\vec{p_i}, t_i)]$ either before or after $[A_{i+1}(\vec{p_{i+1}}, t_{i+1})]$.

When proving the correctness of our semantics, we show that the two formalisms always share the same set of valid plans. We do so in two steps: first, we look at the simpler case where no concurrency exists in the plan, and then consider the more complex one where it does exist.

[5]Notice that $[\![P]\!]$ is not a well-formed formula, but only a sequence of bracket-surrounded action terms.

Lemma 7.11. *Given a* PDDL *problem description where* $\widetilde{\mathfrak{A}} = \emptyset$ *and* $TI = \emptyset$*, a plan* P*, without concurrent happening, is valid if and only if*

$$\Sigma \models [\![P]\!]\big(\mathit{Executable} \land \varphi(G)\big)$$

Here, $\Sigma = \Sigma_{pre} \cup \Sigma_{post} \cup \Sigma_0 \cup \Sigma_{num} \cup \Sigma_{exec}$, $\varphi(G)$ is the transformation of the goal specification into a $\mathcal{ES}$ formula, and $[\![P]\!]\big(\mathit{Executable} \land \varphi(G)\big)$ denotes the formula obtained by syntactically concatenating $[\![P]\!]$ and the formula $\big(\mathit{Executable} \land \varphi(G)\big)$.

Proof. Suppose that $\big(t_1 : A_1(\vec{p_1})\big), \cdots, \big(t_n : A_n(\vec{p_n})\big)$ are all the action happenings in P listed in temporal sequence. Since there is no concurrent happening, we have $t_1 < \cdots < t_n$, and $[\![P]\!]$ has the form $[A_1(\vec{p_1}, t_1)] \cdots [A_n(\vec{p_n}, t_n)]$.

The "only if" direction: If the plan is valid, then there is a sequence of states $I_0, I_1, \cdots, I_n$, where I_0 is the initial state, $\big(t_i : A_i(\vec{p_i})\big)$ is possible in I_{i-1}, resulting in state I_i, and G is satisfied in state I_n.

Since $\big(t_1 : A_1(\vec{p_1})\big)$ is possible in I_0, according to Corollary 7.6, we have

$$\Sigma \models \mathit{Poss}\big(A_1(\vec{p_1}, t_1)\big)$$

so

$$\Sigma \models [A_1(\vec{p_1}, t_1)]\mathit{Executable} \tag{7.7}$$

Meanwhile, according to Theorem 7.7,

$$\Sigma \models [A_1(\vec{p_1}, t_1)]\Sigma_0(I_1) \tag{7.8}$$

Thus $[A_1(\vec{p_1}, t_1)]\Sigma_0(I_1)$ is the progression of $\Sigma_0(I_0)$ through $A_1(\vec{p_1}, t_1)$. The derivation in (7.7) and (7.8) can be repeated n times, and by induction, we get

$$\Sigma \models [A_1(\vec{p_1}, t_1)] \cdots [A_n(\vec{p_n}, t_n)]\mathit{Executable}$$

and

$$\Sigma \models [A_1(\vec{p_1}, t_1)] \cdots [A_n(\vec{p_n}, t_n)]\Sigma_0(I_n)$$

Since G is satisfied in I_n, due to Lemma 7.5, $\Sigma_0(I_n) \cup \Sigma_{num} \models \varphi(G)$, applying Lemma 7.9, we get

$$\Sigma \models [A_1(\vec{p_1}, t_1)] \cdots [A_n(\vec{p_n}, t_n)]\big(\mathit{Executable} \land \varphi(G)\big)$$

The "if" direction: suppose that we have

$$\Sigma \models [\![P]\!]\big(\mathit{Executable} \land \varphi(G)\big) \tag{7.9}$$

From the axiom Σ_{exec}, we get, for $k = 1, \cdots, n$,

$$\Sigma \models [A_1(\vec{p_1}, t)] \cdots [A_k(\vec{p_k}, t)] Executable$$

With repeated application of Corollary 7.6, it can be shown that the PDDL plan P is executable, so we define a sequence of states $I_0, I_1, \cdots, I_n$, where I_0 is the initial state, I_i is the state resulting from executing $\big(t_i : A_i(\vec{p_i})\big)$ in I_{i-1}, for each $i = 1, \cdots, n$.

According to Theorem 7.7, $[A_1(\vec{p_1}, t_1)]\Sigma_0(I_1)$ is the progression of $\Sigma_0(I_0)$ through action $A_1(\vec{p_1}, t_1)$, and by induction, we can show that $[\![P]\!]\Sigma_0(I_n)$ is the progression of $\Sigma_0(I_0)$ through $[\![P]\!]$. Lemma 7.9 suggests that this fact, together with (7.9), implies

$$\Sigma_0(I_n) \cup \Sigma_{num} \models \varphi(G)$$

from which Lemma 7.5 tells us that G is satisfied in I_n.

Thus, P is an executable plan leading to a state that satisfies the goal G. By definition, P is valid. □

Now, let us turn to the concurrent happening of actions. According to the semantics definition by Fox and Long, two actions may happen concurrently only if they are non-interfering. The following definition of non-interfering and mutex actions are taken from Definition 12 in [FL03], with a minor modification.[6]

Definition 7.12 (Mutex actions)**.** Two ground actions, a and b are *non-interfering* if

$$\begin{aligned} GPre_a \cap (Add_b \cup Del_b) =& GPre_b \cap (Add_a \cup Del_a) = \emptyset \\ Add_a \cap Del_b =& Add_b \cap Del_a = \emptyset \\ L_a \cap R_b =& R_a \cap L_b = \emptyset \\ L_a \cap L_b \subseteq& L_a^* \cup L_b^* \end{aligned}$$

If two actions are not non-interfering, they are *mutex*.

Here, $GPre_z$ is the set of ground relational atoms that appear in the precondition formula of z, or in the invariant condition of the durative action if z is a start or end event of it. Add_z and Del_z are the sets of ground relational

[6] Compared with the original definition, we add in Definition 7.12 the additional condition that for two actions to be non-interfering, if either of them is the start or end event of a durative action, then the other may not influence any predicate or function in its invariant condition. This condition is necessary for later serializing plans with durative actions. A detailed discussion about this issue is postponed to Chapter 8.

atoms that are asserted as positive and negative literals in the effect of z respectively. L_z is the set of ground functional atoms that appear in the left hand side of assignment effects of z. R_z is the set of ground functional atoms that appear in the right hand side of assignment effects, or appear in the precondition of action z, or in the invariant condition of the durative action if z is the start or end event of it. L_z^* denotes the set of ground functional atoms that appear in the left hand side of an additive assignment effect of z.

Notice that this condition for concurrent execution of simple actions is conservative, in that some intuitively possible cases may be ruled out. For example, consider the two actions a and b with the following definitions

```
(:action a
  :precondition (or P Q)
  :effect (R))
(:action b
  :precondition (P)
  :effect (not P))
```

In a state where P and Q are both true, the intuition may suppose that a and b may happen concurrently, but according to Definition 7.12, they are actually mutex. The reason that Fox and Long proposed this strong condition for non-interfering actions is to make it easily tractable to check whether two actions are mutex. The interesting thing is that this condition also guarantees that concurrent happening of simple actions are serialization safe.

Definition 7.13 (Serialization safe)**.** For a state I and a happening H of two actions a and b, let I' be the state resulting from concurrently executing a and b in I, I'' be the state resulting from first executing a then executing b in I, I''' be the state resulting from first executing b then executing a in I. We say that H is *serialization safe* in I if I', I'' and I''' are the same state. A happening with more than two actions is serialization safe, if the concurrent happening of any two actions in it is serialization safe.

Lemma 7.14. *If a and b are non-interfering actions, then the concurrent happening of them is serialization safe.*

Proof. Suppose that a and b are to be executed in state I.

- *Executability*
  The concurrent happening of a and b is executable in I, iff the precondition $Pre_a \wedge Pre_b$ is satisfied in I, iff both Pre_a and Pre_b are satisfied in I.

Suppose I^* is the state obtained by executing a in I. Since a and b are non-interfering, we have

$$\begin{gathered} GPre_b \cap (Add_a \cup Del_a) = \emptyset \\ L_a \cap R_b = \emptyset \end{gathered}$$

So, Pre_b is true in I^* iff it is true in I. Therefore the concurrent happening of a and b is executable in I, iff it is possible to first execute a and then b in I.

Similarly, we can get that the concurrent happening of a and b is executable in I, iff it is possible to first execute b and then a in I.

- *Effects*

For the logical state, if a and b are executed concurrently, then

$$\begin{gathered} Add_{ab} = Add_a \cup Add_b \\ Del_{ab} = Del_a \cup Del_b \end{gathered}$$

Since

$$Add_a \cap Del_b = Add_b \cap Del_a = \emptyset$$

we have

$$\begin{aligned} I'_P &= (I_P \setminus Del_{ab}) \cup Add_{ab} \\ &= \big(I_P \setminus (Del_a \cup Del_b) \cup (Add_a \cup Add_b)\big) \\ &= \Big(\big((I_P \setminus Del_a) \cup Add_a\big) \setminus Del_b\Big) \cup Add_b \\ &= (I^*_P \setminus Del_b) \cup Add_b \\ &= I''_P \end{aligned}$$

where I^*_P is the logical state obtained by executing a in I. Similarly, we can obtain $I_P = I''_P$.

For the numerical state, if a and b are executed concurrently, since

$$L_a \cap R_b = L_b \cap R_a = \emptyset$$

the primary numerical expressions in $L_b \setminus L_a$ are only influenced by b and not by a; similarly, those in $L_a \setminus L_b$ are only influenced by a and not by b. So the effects on them can be safely serialized. For $L_a \cap L_b$, since

$$L_a \cap L_b \subseteq L^*_a \cup L^*_b$$

they are always additive assignments, and can thus also be serialized without changing the final value of the update.

□

In the following discussion, when we talk about a plan, we always assume that actions that happen concurrently in the plan are non-interfering, and thus the plan is always serialization safe. This assumption is a reasonable one, since plans with concurrent mutex actions are invalid anyway.

Theorem 7.15. *Given a* PDDL *problem description where* $\widetilde{\mathfrak{A}} = \emptyset$ *and* $TI = \emptyset$*, a plan* P*, with no concurrent happening of mutex actions, is valid if and only if*

$$\Sigma \models [\![P]\!]\big(Executable \wedge \varphi(G)\big)$$

for some $[\![P]\!]$*.*

Proof. The case where no concurrent happening exists in P has been justified in Lemma 7.11.

Otherwise, since all concurrent actions in P are non-interfering, according to Lemma 7.14, P is serialization safe. So P is valid iff a serialized plan of P is valid. Thus the problem is reduced to the non-concurrent case, and the conclusion is justified by Lemma 7.11. □

7.3 Durative Actions

The introduction of durative actions complicates the semantics of PDDL, since an action may no longer be considered as an instant state-changing operator, but instead as a process that spans over an interval of time. Furthermore, there may be constraints and conditional effects over the interval.

When formalizing the semantics, Fox and Long transformed durative actions into ground simple actions, and based the definition upon the non-durative subset. This basic idea is somewhat similar to ours here, in that we also represent a durative action with two simple actions – the start and the end events of it.

However, there are intrinsic differences between the two approaches. The definition given by Fox and Long is offline and meta-theoretic in nature. In particular, they transform both the problem definition and the plan into a simple form, and reason about the validity with this form. For example, they split an action with a conditional effect into two simple actions without conditional effects; they transform the happening of a durative action into a sequence of (variably many) happenings of simple actions to guarantee that all the preconditions are satisfied and all the effects are realized. Unfortunately, those transformations are hardly axiomatizable.

In contrast, our definition is declarative, in the sense that we define, for each syntactical construct, what it means in terms of logic. As mentioned above, due to the limitation of $\mathcal{ES}$, we may only use simple actions, and resort to the auxiliary properties to describe the process-related properties in the domain. So to prove the correctness of our definition, it is necessary to show that our definition is equivalent to the standard semantics given by Fox and Long. This is done, again, by showing that the two formalisms share the same set of valid plans. This subsection is devoted to the proof for the subset concerning durative actions.

First, let us extend the definition of grounded serialized plan in Definition 7.10 to incorporate durative actions.

Definition 7.16 (Grounded serialized plan with durative actions)**.** Let P be a plan, possibly with happenings of durative actions. Further let P' be the derived plan from P with only happenings of simple actions as

$$\begin{aligned} P' = &\Big\{\big(t_i : A_i(\vec{p_i})\big) \big| \big(t_i : A_i(\vec{p_i})\big) \in P\Big\} \cup \\ &\Big\{\big(t_i : \widetilde{A}_i^s(\vec{p_i})\big), \big((t_i + d) : \widetilde{A}_i^e(\vec{p_i})\big) \big| \big(t_i : \widetilde{A}_i(\vec{p_i})[d]\big) \in P\Big\} \end{aligned}$$

Then the *grounded serialized plan* of P, denoted by $[\![P]\!]$, is exactly $[\![P']\!]$, the grounded serialized plan of P'.

Intuitively, the grounded serialized plan is just a sequence of bracket-surrounded simple actions written in temporally non-descending order. Each action in the serialized plan corresponds either to a happening of a simple action or to the happening of a start/end event of a durative action in the PDDL plan.

With Definition 7.16, we are able to translate arbitrary PDDL plan into the $\mathcal{ES}$ notation. In order to proceed to build the equivalence between our semantics and the standard one, let us now have a review how the validity of plans with durative actions are defined by Fox and Long:

> Consider a durative action DA without continuous effects and without conditional effects.
>
> The duration field can be separated into DC^{DA}_{start} and DC^{DA}_{end}, the duration constraint for the start and end of DA, with terms being arithmetic expressions and `?duration`. The separation is conducted in the obvious way, placing `at start` conditions into DC^{DA}_{start} and `at end` conditions into DC^{DA}_{end}.
>
> The durative action DA defines three simple actions, DA_{start}, DA_{end} and DA_{inv}. DA_{start} (respectively DA_{end}) has the precondition equal to the conjunction of all the proposition p, such

that `at start` p (`at end` p) is a condition of DA, together with the duration constraint DC^{DA}_{start} (DC^{DA}_{end}). The effect equals to the conjunction of all simple effects e, such that `at start` e (`at end` e) is an effect of DA. DA_{inv} has the precondition equal to the conjunction of all propositions p, such that `over all` p is a condition of DA, and it has an empty effect.

For a plan P, which is a finite collection of timed actions, each either of the form (t, a) for simple actions or of the form $(t, a[t'])$ for durative actions, the following procedure is used to generate $Simplify(P)$, the *induced simple plan* of P:

1. Find the happening sequence, the ordered sequence of time points from the set of times

$$\big\{t|(t,a) \in P \vee (t, a[t']) \in P \vee (t - t', a[t'] \in P\big\}$$

2. The induced simple plan $simplify(P)$ includes the set of pairs defined as follows:

   a) (t, a) for each happening of simple action $(t, a) \in P$;

   b) (t, a_{start}) and $(t+t', a_{end})$ for each happening of durative action $(t, a[t']) \in P$;

   c) $\big((t_i+t_{i+1})/2, a_{inv}\big)$ for each happening of durative action $(t, a[t']) \in P$ and for each i such that $t \leq t_i < t+t'$ where t_i and t_{i+1} are in the happening sequence of P.

Then, a plan, P, is *executable* if $Simplify(P)$ is executable, and P is *valid* if it is executable and the goal is satisfied in the final state produced by $Simplify(P)$.

Notice that if we disallow for both invariant constraints and duration constraints, then, for any durative action $\widetilde{A}$, the induced simple plan does not contain $\widetilde{A}_{inv}$, and the preconditions for $\widetilde{A}$'s start and end events do not mention $DC^{\widetilde{A}}_{start}$ or $DC^{\widetilde{A}}_{end}$. In this case the grounded serialized plan can be considered as a direct syntactical transformation from the induced simple plan, so we have the following simple proposition in Lemma 7.17.

Lemma 7.17. *In a domain* $D = \big\langle \mathbf{T}, \mathbf{F}, \mathbf{f}, \mathbf{O}, \langle \mathfrak{A}, \widetilde{\mathfrak{A}} \rangle, I, TI, G \big\rangle$, *if* $TI = \emptyset$ *and* *for each durative action* $\widetilde{A}_j \in \widetilde{\mathfrak{A}}$, *the following conditions are satisfied*

$$\begin{cases} \delta^s_{\widetilde{A}_j} \equiv true \\ \delta^e_{\widetilde{A}_j} \equiv true \\ \pi^o_{\widetilde{A}_j} \equiv true \\ \epsilon^o_{\widetilde{A}_j} = \emptyset \\ \epsilon^e_{\widetilde{A}_j} \text{ has the form } \langle true, true, \varphi^e_{j,i} \rangle \Rightarrow \psi^e_{j,i} \end{cases} \tag{7.10}$$

then a plan P, with no concurrent happening of mutex actions, is valid if and only if

$$\Sigma \models [\![P]\!]\big(Executable \wedge \varphi(G)\big)$$

for some $[\![P]\!]$

Proof. According to Fox and Long, P is a valid plan if and only if the *induced simple plan* of P is executable and the goal is satisfied after executing this plan. Notice that under the conditions in (7.10), the induced simple plan is exactly the same as $[\![P]\!]$, (except for the syntactical differences.) So the executablilty and validity of the plan is reduced to the non-durative case. According to Theorem 7.15, the lemma is proved. □

Lemma 7.17 says that if all the durative actions in the domain mention no duration constraint, invariant condition, inter-temporal effect or continuous effect, and the domain does not have timed initial literals, then the state transitional semantics and our declarative semantics are equivalent in the sense that they share the same set of valid plans.

With Lemma 7.17 as the starting point, we are now ready to investigate the more advanced features of durative actions. This is done in the next four subsections.

7.3.1 Duration Constraint

The treatments of the duration constraint are very similar in the two formalisms. Both definitions split the duration constraint into a start condition and an end condition, and ensure the satisfaction by appending them to the precondition formulas.

The only difference is lies in the start duration condition. In Fox and Long's definition, the start duration condition is checked as part of the precondition of the start event, whereas in ours, the values of the terms mentioned in the start duration condition is remembered, and the condition is not checked until the end event of the durative action.

This difference comes from the way a plan is represented. In Fox and Long's approach, the duration of a durative action is explicitly specified in the plan, so the value is available even before the precondition of its start event is evaluated. In contrast, we represent the plan with a sequence of simple actions, and the duration of a durative action can be obtained only when we get the knowledge when its start event and its end event are executed.

Despite this difference, it is not difficult to prove the following equivalence proposition in Lemma 7.18.

Lemma 7.18. *For a* Pddl *plan P, in which $\big(s : \widetilde{A}_j(\vec{p_j})[d]\big)$ is a happening of a durative action. The duration d satisfies the duration constraint if and only if*

$$\big(\delta^s_{\widetilde{A}_j}(\vec{p_j})\big)^{f_i(\vec{q_i})\qquad duration}_{f^s_i(\vec{p_j},\vec{q_i})\ t-since(\widetilde{A}_j(\vec{p_j}))} \wedge \big(\delta^e_{\widetilde{A}_j}(\vec{p_j})\big)^{duration}_{t-since(\widetilde{A}_j(\vec{p_j}))} \tag{7.11}$$

is satisfied at the happening of $end\big(\widetilde{A}_j(\vec{p_j}), t\big)$ in $[\![P]\!]$.

Proof. Since $\big(s : \widetilde{A}_j(\vec{p_j})[d]\big)$ is a happening in P, according to Definition 7.16, both $start\big(\widetilde{A}_j(\vec{p_j}), s\big)$ and $end\big(\widetilde{A}_j(\vec{p_j}), s{+}d\big)$ are in $[\![P]\!]$. So, according to Theorem 7.4, when the precondition of $end\big(\widetilde{A}_j(\vec{p_j}), t\big)$ is evaluated, $since\big(\widetilde{A}_j(\vec{p_j})\big) = s$, and moreover, $t = s + d$. Thus,

$$t - since\big(\widetilde{A}_j(\vec{p_j})\big) = d$$

For the end duration constraint, $\delta^e_{\widetilde{A}_j(\vec{p_j})}$ is a direct translation from $DC^{\widetilde{A}_j}_{end}$, so according to Lemma 7.5, it is obvious that $\big(\delta^e_{\widetilde{A}_j(\vec{p_j})}\big)^{duration}_{d}$ holds if and only if $DC^{\widetilde{A}_j}_{end}[duration = d]$ holds.

For the start duration constraint, $\delta^s_{\widetilde{A}_j}(\vec{p_j})$ is also a direct translation from $DC^{\widetilde{A}_j}_{start}$. Since $f^s_{j,i}(\vec{p_j}, \vec{q_i})$ are the values of $f_i(\vec{q_i})$ at the start of $\widetilde{A}_j(\vec{p_j})$, the instantiated formula $\big(\delta^s_{\widetilde{A}_j}(\vec{p_j})\big)^{f_i(\vec{q_i})\quad duration}_{f^s_{j,i}(\vec{p_j},\vec{q_i})\ d}$ is exactly the same as $DC^{\widetilde{A}_j}_{start}$ when $\widetilde{A}^s_j(\vec{p_j})$ is activated. So, $\big(\delta^s_{\widetilde{A}_j}(\vec{p_j})\big)^{f_i(\vec{q_i})\quad duration}_{f^s_{j,i}(\vec{p_j},\vec{q_i})\ d}$ holds at the end event iff $DC^{\widetilde{A}_j}_{start}[duration = d]$ holds.

As a result, the condition in the formula (7.11) is satisfied at the end event of $\widetilde{A}_j$, if and only if both $DC^{\widetilde{A}_j}_{start}[duration = d]$ and $DC^{\widetilde{A}_j}_{end}[duration = d]$ are satisfied in their respective happenings. □

7.3.2 Invariant Conditions

As discussed above, when protecting the invariant conditions of a durative action, Fox and Long insert monitoring actions in the interval of the action.

In contrast, we assert that the condition holds at the beginning of the durative action, and no action violates it during its execution. Here, we prove that the two approaches are indeed equivalent in terms of plan validity.

Let us consider the happening of a durative action $\big(t : \widetilde{A}(\vec{x})[d]\big)$, whose invariant condition is $\pi^o_{\widetilde{A}}$. Without loss of generality, suppose that

$$(t_{i_0} : A_{i_0}), (t_{i_2} : A_{i_2}), \cdots, (t_{i_{2n-2}} : A_{i_{2n-2}}), (t_{i_{2n}} : A_{i_{2n}}) \tag{7.12}$$

are all the happenings of non-monitoring actions during the (closed) interval of the happening of $\widetilde{A}(\vec{x})$, where

$$t = t_{i_0} \leq t_{i_2} \leq \cdots \leq t_{i_{2n-2}} \leq t_{i_{2n}} = t + d$$

and

$$\begin{cases} A_{i_0} & = & \widetilde{A}^s(\vec{x}) \\ A_{i_{2n}} & = & \widetilde{A}^e(\vec{x}) \end{cases}$$

then, according to Fox and Long's definition, the monitoring action $\widetilde{A}_{inv}$, with precondition $\pi^o_{\widetilde{A}}$ and empty effect, needs to be placed at time points

$$t_{i_{2k+1}} = \frac{t_{i_{2k}} + t_{i_{2k+2}}}{2}, \quad k = 0, 1, \cdots, n-1$$

In this setting, we have the following lemma:

Lemma 7.19. *In a domain without continuous effects or timed initial literals, the precondition $\pi^o_{\widetilde{A}}$ of the monitoring action $\widetilde{A}_{inv}$ is satisfied at time points $t_{i_{2k+1}}$, if and only if $\mathcal{R}(A_{i_{2k}}, Performing(\widetilde{A}) \supset \pi^o_{\widetilde{A}})$ is satisfied at time points $t_{i_{2k}}$, for $k = 0, 1, \cdots, n-1$.*

Proof. According to the definition of regression, $\mathcal{R}(A_{i_{2k}}, \pi^o_{\widetilde{A}})$ is true in the current situation if and only if $Performing(\widetilde{A}) \supset \pi^o_{\widetilde{A}}$ is true immediately after the execution of $A_{i_{2k}}$ if and only if (from Theorem 7.4, we have $Performing\big(\widetilde{A}(\vec{x})\big)$ holds between t_{i_0} and $t_{i_{2n}}$) $\pi^o_{\widetilde{A}}$ is true immediately after the execution of $A_{i_{2k}}$. Since there is no action between $A_{i_{2k}}$ and $A_{i_{2k+1}}$, and according to the assumption that there is no continuous effect or timed initial literal, nothing changes the truth value of $\pi^o_{\widetilde{A}}$ in the interval, and thus the conclusion in the lemma is drawn. □

Lemma 7.19 proves that the method we proposed in (6.19) in Chapter 6 is equivalent to Fox and Long's approach of adding happenings of guarding actions, in ensuring that valid plans have all invariant conditions satisfied.

7.3.3 Conditional Effects

In this subsection, we shall prove the correctness of our treatment of conditional effects in durative actions. As we shall soon see, since our treatment for the conditional effects is very similar to the one in Fox and Long's semantics definition, the proof here is straightforward.

Notice that the case of intra-temporal conditional effects has already been covered in Lemma 7.17. So in the following discussion, we only concentrate on the inter-temporal conditional effects.

Suppose we have the following end effect for the happening of a durative action $\big(t : \widetilde{A}(x)[d]\big)$:

$$\begin{aligned}&\text{(when (and (at start } \varphi^s_{\widetilde{A}})\\ &\qquad\qquad\text{(over all } \varphi^o_{\widetilde{A}})\\ &\qquad\qquad\text{(at end } \varphi^e_{\widetilde{A}}))\\ &\quad\text{(at end } \psi_{\widetilde{A}}))\end{aligned}$$

which has the normal form

$$\langle \varphi^s_{\widetilde{A}}, \varphi^o_{\widetilde{A}}, \varphi^e_{\widetilde{A}} \rangle \Rightarrow \psi_{\widetilde{A}} \tag{7.13}$$

To define the semantics of the conditional effect of this form, Fox and Long introduce two special new propositions that are unique to any name in the domain. First, for the start-end conditional effect, they introduce the proposition $M^s_{\widetilde{A}}$, with an additional conditional effect `(when (`$\varphi^s_{\widetilde{A}}$`) (`$M^s_{\widetilde{A}}$`))` in the start simple action. Second, for the overall-end conditional effect, they introduce the proposition $M^o_{\widetilde{A}}$, with an additional effect $(M^o_{\widetilde{A}})$ in the start simple action and a conditional effect `(when (and (`$M^o_{\widetilde{A}}$`) (not `$\varphi^o_{\widetilde{A}}$`)) (not (`$M^o_{\widetilde{A}}$`)))` in the monitoring action. Then, to see whether the original inter-temporal conditional effect takes place, they add the intra-temporal conditional effect

`(when (and (`$M^s_{\widetilde{A}}$`) (`$M^o_{\widetilde{A}}$`) (`$\varphi^e_{\widetilde{A}}$`)) `$\psi_{\widetilde{A}}$`)`

to the end simple action.

Remember that we also introduced similar propositions in our semantics definition: the auxiliary fluent predicates $\xi^s_{\widetilde{A}}$ and $\xi^o_{\widetilde{A}}$, with successor state axioms

$$\begin{aligned}&\Box[a]\xi^s_{\widetilde{A}}(\vec{p},\vec{q}) \equiv\\ &\quad \exists t.a = \mathit{start}\big(\widetilde{A}(\vec{p}), t\big) \wedge \varphi^s_{\widetilde{A}} \vee\\ &\quad \xi^s_{\widetilde{A}}(\vec{p},\vec{q}) \wedge \neg\exists t'.a = \mathit{end}(\widetilde{A}(\vec{p}), t')\\ &\Box[a]\xi^o_{\widetilde{A}}(\vec{p},\vec{q}) \equiv\\ &\quad \exists t.a = \mathit{start}(\widetilde{A}(\vec{p}), t) \wedge \mathcal{R}[a, \varphi^o_{\widetilde{A}}] \vee\end{aligned}$$

$$\xi^o_{\widetilde{A}}(\vec{p},\vec{q}) \wedge \neg\exists t'.a = end(\widetilde{A}(\vec{p}),t') \wedge \mathcal{R}[a,\varphi^o_{\widetilde{A}}]$$

Here, $\vec{p}$ and $\vec{q}$ are used to identify the "memory" from different happenings of $\widetilde{A}$'s instances and different conditional effects in $\widetilde{A}$, where $\vec{p}$ are the parameters to the action symbol $\widetilde{A}$ and $\vec{q}$ are all the free variables in $\psi_{\widetilde{A}}$ other than those in $\vec{p}$.

To see whether the effect finally should take place, we further transform the inter-temporal conditional effect in (7.13) into a intra-temporal one by substituting $\varphi^s_{\widetilde{A}}$ and $\varphi^o_{\widetilde{A}}$ with $\xi^s_{\widetilde{A}}(\vec{p},\vec{q})$ and $\xi^o_{\widetilde{A}}(\vec{p},\vec{q})$, respectively. As we can see, all the treatments in the two formalisms are similar. In fact, we have the following lemma:

Lemma 7.20. *For the happening of a durative action* $(t : \widetilde{A}(\vec{p})[d])$ *with conditional effect* $\langle\varphi^s_{\widetilde{A}},\varphi^o_{\widetilde{A}},\varphi^e_{\widetilde{A}}\rangle \Rightarrow \psi_{\widetilde{A}}$*, the truth value of the propositions* $M^s_{\widetilde{A}}$ *and* $M^o_{\widetilde{A}}$ *are pairwise equal to the auxiliary properties* $\xi^s_{\widetilde{A}}(\vec{p},\vec{q})$ *and* $\xi^o_{\widetilde{A}}(\vec{p},\vec{q})$ *in the situation at time* $t+d$*,* i.e. *at the happening of the end event of* $\widetilde{A}(\vec{p})$*.*

Proof. The validity of this lemma is obvious, if we compare the Pddl effect definitions for $M^s_{\widetilde{A}}$ and $M^o_{\widetilde{A}}$ with the successor state axioms for $\xi^s_{\widetilde{A}}$ and $\xi^o_{\widetilde{A}}$. □

It immediately follows from Lemma 7.20 that the two approaches draw same conclusion on whether the premise $\langle\varphi^s_{\widetilde{A}},\varphi^o_{\widetilde{A}},\varphi^e_{\widetilde{A}}\rangle$ enables $\psi_{\widetilde{A}}$ to takes place or not.

7.3.4 Continuous Effects

In order to define the semantics of continuous effects of durative actions, Fox and Long resort to differential equations. Definition 7.21 (originally Definition 22 in [FL03]) is their definition of the *continuous update function.*

Definition 7.21 (Continuous update function)**.** Let C be a set of ground continuous effects for a planning instance, I, and $St = (t, S, \mathbf{X})$ be a state. The *continuous update function*, defined by C for state St, is the function $f_C : \mathbb{R} \rightarrow \mathbb{R}^n$, where n is the dimension of the planning problem (the number of fluent function instances), such that

$$\frac{d\,f_C}{d\,t} = g$$

and

$$f_C(0) = \mathbf{X}$$

where g is the update function generated for an action a with

$$NP_a = \big\{(\langle\texttt{op}\rangle\ \texttt{P Q})|\big(\langle\texttt{op}\rangle\ \texttt{P (}*\ \texttt{\#t Q)}\big) \in C\big\}$$

Intuitively, suppose the state St starts at t_i and ends at t_{i+1}, then for any $t_i \leq t \leq t_{i+1}$, $f_C(t - t_i)$ is the vector of the values of all function instances in the domain at time t. Here, $\mathbf{X} = f_C(0)$ is the vector of the initial values of all function instances at t_i, the start of the state St, and g is the changing rate. Since only linear changes are allowed in PDDL and there is no other happening between t_i and t_{i+1}, g, the derivative of f_C, is a vector of constant numbers.

Recall that according to Theorem 7.4, the changing rate of any numerical property $f_k(\vec{q_k})$ satisfies

$$rate\big(f_k(\vec{q_k})\big) = \Bigg(\sum_{\substack{(s:\widetilde{A}_j[d]) \in P \wedge s \leq t < s+d \\ \langle +, f_k(\vec{q_k}), Q_{j,k}\rangle \in \epsilon^o_{\widetilde{A}_j}}} Q_{j,k}\Bigg) - \Bigg(\sum_{\substack{(s:\widetilde{A}_j[d]) \in P \wedge s \leq t < s+d \\ \langle -, f_k(\vec{q_k}), Q_{j,k}\rangle \in \epsilon^o_{\widetilde{A}_j}}} Q_{j,k}\Bigg)$$

It is easy to see that there exists a correspondance between the changing rate $rate\big(f_k(\vec{q_k})\big)$ in the basic action theory and the derivative of the continuous update function $\frac{d f_C}{dt}$ in Definition 7.21. Lemma 7.22 characterizes this correspondance.

Lemma 7.22. *Suppose t_i and t_{i+1} are the times for two neighboring happenings in a plan with continuous durative actions. Let f_C be the continuous update function on $[t_i, t_{i+1})$, which corresponds to the the fluent numerical properties $f_1(\vec{q_1}), \cdots, f_n(\vec{q_n})$. Further, let l_k be the linear objects assigned to $f_k(\vec{q_k})$ at time t_1. Then,*

$$\frac{d f_C}{dt} = \langle rate(l_1), \cdots, rate(l_n)\rangle$$

Proof. This conclusion is obvious from Definition 7.21 and Theorem 7.4. □

Considering the fact that the time for function f_C is relative to the latest happening time, whereas $f_k(\vec{q_k})$ has the global time, we have the following lemma to build the connection between the two.

Corollary 7.23. *In the setting of Lemma 7.22, if*

$$f_C(0) = \langle eval\big(l_1, t_i\big), \cdots, eval\big(l_n, t_i\big)\rangle \tag{7.14}$$

then

$$f_C(t - t_i) = \langle eval\big(l_1, t\big), \cdots, eval\big(l_n, t\big)\rangle, \qquad t \in [t_i, t_{i+1})$$

Proof. For any $t \in [t_i, t_{i+1})$, we have

$$\begin{aligned}
& f_C(t - t_i) \\
=& f_C(0) + \int_{t_i}^{t} \frac{d f_C}{d t} dt \qquad \left(\text{since } \frac{d f_C}{d t} \text{ is constant in } [t_i, t_{i+1})\right) \\
=& f_C(0) + \frac{d f_C}{d t} \cdot (t - t_i) \qquad (\text{since (7.14) and Lemma 7.22}) \\
=& \langle \mathit{eval}\big(l_1, t_i\big), \cdots, \mathit{eval}\big(l_n, t_i\big) \rangle \\
& \quad + \langle \mathit{rate}\big(l_1\big) \cdot (t - t_i), \cdots, \mathit{rate}\big(l_n\big) \cdot (t - t_i) \rangle \\
=& \Big\langle \mathit{eval}\big(l_1, t_i\big) + \mathit{rate}\big(l_1\big) \cdot (t - t_i), \cdots, \\
& \quad \mathit{eval}\big(l_n, t_i\big) + \mathit{rate}\big(l_n\big) \cdot (t - t_i) \Big\rangle \\
& \qquad\qquad\qquad\qquad (\text{comparing the forms}) \\
=& \langle \mathit{eval}\big(l_1, t\big), \cdots, \mathit{eval}\big(l_n, t\big) \rangle
\end{aligned}$$

□

Since l_k models the function $f_k(\vec{q_k})$ in the interval $[t_i, t_{i+1})$ in the basic action theory, Corollary 7.23 implies that concerning the continuous update, as long as the continuous fluents have correct values in the state at t_i, they are also correct throughout this state, and in particular, immediately before the beginning of the next state at t_{i+1}. Here, a "correct value" means that the evaluation result of the linear object in our approach is equal to the one obtained from the continuous update functions in Fox and Long's definition.

With this conclusion for single-step update, we are now able to prove that our approach correctly models the *trace* of the plan.

Definition 7.24 (Trace)**.** Let D be a planning instance that includes continuous durative actions, P be a plan for D, SP be the simplified plan of P, $\{t_i\}_{i=0,\cdots,m}$ be the happening sequence for SP and I_0 be the initial state for D. Further let Cts be the *systems of continuous effects*

$$Cts = \{(C, t_i, t_{i+1}) | C \text{ is the set of active continuous effects over } (t_i, t_{i+1})\}$$

Then, the *trace* for P is the sequence of states $\{I_i\}_{i=0,\cdots,m}$ defined as follows:

- If there is no element $(C, t_i, t_{i+1}) \in Cts$, then I_{i+1} is the state resulting from applying the happening at t_i in the simple plan SP to the state I_i;
- If $(C, t_i, t_{i+1}) \in Cts$, then let T_i be the state formed by substituting $f_C(t_{i+1} - t_i)$ for the numeric part of state I_i, where f_C is the continuous

update function defined by C for state I_i [7]. Then I_{i+1} is the state resulting from applying the happening at t_i in the simple plan SP to the state T_i. If f is undefined for any element in Cts, then so is the trace.

Definition 7.24 (originally Definition 24 in [FL03]) states that the procedure to obtain the successor state of a state I_i is to first apply the continuous numerical updates, if any, and then perform the add and delete operations as usual. The following Lemma 7.25 shows that our solution in Section 6.3.3 correctly captures Fox and Long's notion of the trace.

Lemma 7.25. *In the setting of* Definition 7.24*, suppose* Σ *is the basic action theory derived from* D*, and* $A_1, \cdots, A_n$ *are the sequence of actions in the simplified plan of* P*. Then for any functional fluent* $f_k(\vec{q_k})$*, any number* r *and any state* I_i*,*

$$f_k(\vec{q_k}) = r$$

holds in state I_i *if and only if*

$$\Sigma \models [A_1] \cdots [A_i] eval(f_k(\vec{q_k}), t_i) = r$$

Proof.
Due to Lemma 7.5, $f_k(\vec{q_k}) = r$ holds in I_0 if and only if $\Sigma \models eval(l_k, t_0) = r$.

First, consider the state immediately before I_1 at time t_1. According to Definition 7.24, the value of $f_k(\vec{q_k})$ at (immediately before) time t_1 is $\{f_C(t_1 - t_0)\}_k$, where f_C is the continuous update function on the interval $[t_0, t_1)$, and $\{f_C(t_1 - t_0)\}_k$ stands for the k-th dimension of the vector $f_C(t_1 - t_0)$.

According to Corollary 7.23, $\{f_C(t_1 - t_0)\}_k$ equals to $eval(f_k(\vec{q_k}), t_1)$. So, if we construct a new state I_1^-, which is the same as I_0, except that the values of all numerical function $f_k(\vec{q_k})$ is changed from $eval(f_k(\vec{q_k}), t_0)$ to $eval(f_k(\vec{q_k}), t_1)$. From this construction, I_1^- serves as the intermediate state T_0 in Definition 7.24.

Now, let us consider the application of A_1 in state I_1^-. Since the resulting state I_1 is at the same time as I_1^-, there is no longer continuous numerical

[7] The continuous update function is defined on an interval that is closed on the left but open on the right. So strictly speaking, $f_C(t_{i+1} - t_i)$ is undefined. Here, we understand $f_C(t_{i+1} - t_i)$ as

$$\lim_{t' \to t_{i+1}^-} f_C(t' - t_i)$$

that is, the value of f_C immediately before time t_{i+1}.

change to consider. According to Theorem 7.7, $[A_1]\Sigma_0(I_1)$ is the progression of $\Sigma_0(I_1^-)$.

As a result, $f_k(\vec{q_k}) = r'$ holds in I_1 if and only if $\Sigma \models [A_1]eval(l_k, t_1) = r'$.

With an induction on the length of the plan, the proposition in the lemma is proved. □

Lemma 7.25 shows that our basic action theory correctly models the trace of plans of a domain, possibly with continuous durative actions. Now we are only left with the satisfaction of invariant conditions with the presence of continuous effects.

Lemma 7.26. *Let D be a planning instance with continuous durative actions and invariant conditions, Σ be the basic action theory obtained from D as described in Chapter 6, Σ' be the same as Σ except that it does not take the invariants into account, P be a plan for D. Suppose that $\Sigma' \models [\![P]\!]Executable$. Further, let $\{(t_i : A_i)\}_{i=1,\cdots,m}$ be all the happenings in the derived simple plan of P, and Q_i be the conjunction of all the invariant conditions between t_i and t_{i+1}. Then all Q_i are satisfied in their corresponding interval if and only if $\Sigma \models [\![P]\!]Executable$.*

Proof. **The "only if" direction**

Suppose all Q_i are satisfied in their corresponding interval. Notice that each Q_i is a conjunction of the invariant conditions $\pi^o_{\widetilde{A}_{l_1}}, \cdots, \pi^o_{\widetilde{A}_{l_k}}$ of the actions that are in progress $\widetilde{A}_{l_1}, \cdots, \widetilde{A}_{l_k}$. So, according to Lemmas 7.5 and 7.9,

$$\Sigma \models [A_1]\cdots[A_i]Performing(\widetilde{a}) \supset Eval[\pi^o_{\widetilde{a}}, t]$$

which leads to

$$\Sigma \models [A_1]\cdots[A_i]\neg Obli\big(end(\widetilde{a}), t\big)$$

Since $\Sigma' \models [\![P]\!]Executable$ and Σ differs from Σ' only with the additional *Obli* conditions, but none of them are true at any time in the duration of the plan, We know that $\Sigma \models [\![P]\!]Executable.$

The "if" direction

Suppose that there is some $t \in [t_i, t_{i+1})$ such that Q_i is false at t. Then according to Lemma 7.5, for some durative action $\widetilde{A}$,

$$\Sigma \models [A_1]\cdots[A_i]Performing(\widetilde{A}) \wedge \neg Eval[\pi^o_{\widetilde{A}}, t]$$

As a result, we have

$$\Sigma \models [A_1]\cdots[A_i]Obli\big(end(\widetilde{A}), t\big)$$

so

$$\Sigma \models [A_1] \cdots [A_i] \Big(Obli\big(end(\widetilde{A}), t\big) \wedge t < time(A_{i+1}) \Big)$$

Thus, we have

$$\Sigma \models [A_1] \cdots [A_i] \neg Poss(A_{i+1})$$

According to Σ_{exec}, we have

$$\Sigma \not\models [\![P]\!] Executable$$

□

7.4 Timed Initial Literals

To define the meaning of timed initial literals in the standard semantics of Pddl 2.2, Edelkamp and Hoffmann introduced additional happenings to the temporal plan. Each of the additional happening $(t : A_{til})$ corresponds to a timed initial literal $\langle t_0, \varphi_0 \rangle$ in the problem definition. The happening time t equals to t_0 as specified in the timed initial literal, and the action A_{til} is new and unique, with the precondition TRUE and the single effect asserting the specified literal φ_0 [EH04].

The semantics that we introduced in Chapter 6 is hooked on their definition. We forced the insertion of similar auxiliary actions in the plan by specifying the actions as obligatory at the very time points. The correct insertion is guaranteed by the following Lemma 7.27.

Lemma 7.27. *Suppose $\langle t_0, \varphi_0 \rangle \in TI$ is a timed initial literal in a problem definition D. If the obligatory condition*

$$\Box Obli\big(A_{\langle t_0, \varphi_0 \rangle}(t)\big) \equiv (t = t_0)$$

is defined in the basic action theory $\Sigma(D)$, then for every executable plan P, the time of whose last happening t_{end} satisfying $t_{end} > t_0$, $A_{\langle t_0, \varphi_0 \rangle}(t_0)$ is in P.

Proof. Since $t_{end} > t_0$, there must be a happening $A_i(\vec{x_i}, t_i)$ in P, such that $t_0 < t_i \leq t_{end}$, and there is no other happening $A_k(\vec{x_k}, t_k)$ in P with $t_0 < t_k < t_i$.

Suppose $A_{\langle t_0, \varphi_0 \rangle}(t_0)$ is not in P, then at time t_i, we have

$$Obli\big(A_{\langle t_0, \varphi_0 \rangle}(t_0)\big) \wedge now < t_0 < t_i$$

which contradicts the precondition of $A_i(\vec{x_i}, t_i)$ that has the form as shown in (5.36). This further contradicts with the assumption that P is executable. As a result, $A_{\langle t_0, \varphi_0 \rangle}(t_0)$ must exist in P. □

Since for each timed initial literal $\langle t_0, \varphi_0 \rangle$, the corresponding introduced action $A_{\langle t_0, \varphi_0 \rangle}(t_0)$ exists in any executable plan according to Lemma 7.27, the correct semantics of timed initial literals is guaranteed further by the original definition by Edelkamp and Hoffmann.

7.5 Conclusion

So far, we have proved the individual features in our semantic mapping. From these fragments, it is easy to obtain that our semantics for the subset of PDDL with time and concurrency, as defined in Chapter 6, is a correct one. This correctness is based on the fact that for a PDDL problem definition, our semantics and the standard one determine the same set of valid plans. This is formalized in the following Theorem 7.28.

Theorem 7.28. *Given a* PDDL *problem description*

$$D = \langle \mathbf{T}, \mathbf{F}, \mathbf{f}, \mathbf{O}, \mathbf{A}, I, TI, G \rangle$$

a plan P, with no concurrent happening of mutex actions, is valid if and only if

$$\Sigma \cup \Sigma_{exec} \models [\![P]\!]\big(\mathit{Executable} \land \varphi(G)\big)$$

for some $[\![P]\!]$.

Proof. The validity of this theorem follows immediately from Theorem 7.15 and Lemmas 7.17–7.27. □

Chapter 8

Conclusion and Future Work

8.1 The Result

The major contribution of this thesis lies in two aspects.

First, we have studied the possible extensions to the logic $\mathcal{ES}$, which is approximately in parallel to the ones in the situation calculus [Rei01]. This enables $\mathcal{ES}$ to reason about, among other things, numerics, time, concurrency and coerciveness, such that many realistic features in a domain can be modeled in the logic. The extensions also show the generality and expressiveness of $\mathcal{ES}$. Furthermore, due to the nice features of $\mathcal{ES}$ (*e.g.* situation properties are defined in the semantics, a fixed domain of discourse is assumed, *etc.*), the formulation becomes simpler than in the classical situation calculus.

Second, we have built a semantic mapping between a subset of PDDL with time and concurrency and our extended version of basic action theories in $\mathcal{ES}$, and have proved the correctness of this mapping. This serves as a declarative semantics for this subset of PDDL. Unlike the state-transitional semantics, which rely on meta-theoretic structures in the definition [FL03], ours here defines what each element in the language means in a purely logical notion. The advantage is that, with this declarative semantics, it is possible to bridge planning and action formalisms, such that one approach can benefit from the results of the other. For example, as we shall go a little deeper in Section 8.2.3, our result makes it possible to integrate external planners in action languages such as GOLOG.

8.2 Future Work

Due to the time constraints and the author's knowledge level, a few topics are still left open. Here, we give a brief introduction to several directions of future work, and some preliminary ideas on them.

8.2.1 The Reoccurance Problem

In general, it is possible to have two instances of a same durative action happen concurrently. For example, suppose we have a durative action $refuel(v)$ which fills a tank with petrol at a rate of v. Then,

$$\{(1 : refuel(30)[3]), (2.5 : refule(30)[5])\}$$

may be a valid plan according to the semantics given by Fox and Long. This plan has the meaning that a refueling process starts at time 1 with duration 3, and another starts at time 2.5 with duration 5; the refueling rate of both processes is 30.

Unfortunately, this plan is not executable if we map it to the basic action theory. Intuitively, the reason is like this: after the virtual simple action $start(refuel(30), 1)$, $Performing(refuel(30))$ becomes true. At time 2.5, when the start event of the second process $start(refuel(30), 2.5)$ is to be activated, $Performing(refuel(30))$ remains true, since $end(refuel(30), 4)$ has not been executed by that time. So due to the durative action axioms in $\Sigma_{dura} \subset \Sigma$, we have

$$\Sigma \models [start(refuel(30), 1)]\neg Poss\Big(start(refuel(30), 2.5)\Big)$$

As we can see, a valid plan in Pddl becomes unexecutable, and thus invalid, in our translated basic action theory. This is called the *reoccurrance problem.*

The cause of the reoccurrance problem is that we identify durative actions only by their names. According to the semantics of $\mathcal{ES}$, $refuel(30)$ refers to one single element in the domain, so it is not possible to distinguish whether we mean the first process or the second, purely from this name. If the second happening were $(2.5 : refuel(30.001)[5])$, for example, then this problem would not occur.

In the previous chapters, we have implicitly assumed that such an ambiguity does not exist in the Pddl domains, but in general this assumption does not necessarily hold.

To solve this problem, we can simply introduce a unique identifier of each happening of the durative action. In the refueling example, for instance, the concurrent refueling process may come from two distinct pipes to the tank, so if the domain designer identify the two processes with their corresponding pipe name, and define the refuel action as $refuel(id, v)$, where id denotes the pipe and v is the rate, then the problem is avoided. However, this solution requires the modification of the domain definition. It is interesting to study, given a domain, potentially with reoccurrance problem, how we can automatically assign unique identifiers to the processes, when we construct the basic action theories from it.

8.2.2 Integrating True Concurrency

In the development of this thesis, one of our guidelines was to stick to the existing syntax and semantics of the logic $\mathcal{ES}$, and only modify the basic action theory to extend the expressiveness. One consequence of this decision is that we choose to use interleaved concurrency.

As we can see from the previous chapters, the interleaved account for concurrency is almost always sufficient for capturing the semantics for the Pddl. This result owes a lot to the strong condition of mutex actions defined by Fox and Long, in that actions that interact with each other cannot happen simultaneously according to the standard semantics of Pddl.

However, two exceptions exist, where interleaved concurrency fails. One is the case where an obligatory action is scheduled at the same time as the happening of other actions; the other is where an action influences the invariant condition of a durative action whose start event happens simultaneously with it. Now, let us illustrate them with two examples.

As an example to the first problem, consider the plan

$$\{(5 : A_1), (10 : A_2)\}$$

and the additional condition $\Sigma \models \Box \mathit{Obli}\big(B(5)\big)$. Ideally, we expect to have

$$\Sigma \models [A_1(5)][A_2(10)]\neg \mathit{Executable} \tag{8.1}$$

since $[B(5)]$ is not in the action sequence. Unfortunately, even with Σ_{Obli}, (8.1) may fail. To see why, notice that when $A_1(5)$ is about to activate, $\mathit{now} = 0$ and $\mathit{time}\big(A_1(5)\big) = 5$, but there is no obligatory action a' such that $0 < \mathit{time}(a') < 5$, provided $B(5)$ is the only obligatory action in the domain; similarly, when $A_2(10)$ is about to activate, $\mathit{now} = 5$ and $\mathit{time}\big(A_2(10)\big) = 10$, but again there is no obligatory action scheduled in the *open* interval $(5, 10)$. As a result, our solution to coerciveness fails in this case.[1] That is why we introduced Assumption 5.1 in Chapter 5 to temporarily avoid the problem.

[1] One may argue that we should modify Equation (5.36) to

$$\Box \mathit{Poss}(a) \supset \neg\big(\exists a'.\mathit{Obli}(a') \wedge \mathit{now} \leq \mathit{time}(a') \leq \mathit{time}(a)\big)$$

to also take the end points of the interval into account. However, this does not solve the problem either. Let us denote the modified basic action theory with Σ'. For the first $\leq$ in the above equation, notice that

$$\Sigma' \models [A_1(5)][B(5)]\neg \mathit{Poss}\big(A_2(10)\big)$$

As for the second problem, let us consider the following two actions:

```
(:durative-action A
  :parameters ()
  :duration ()
  :condition (over all (P))
  :effect ()
)
(:action B
  :parameters ()
  :condition ()
  :effect (P)
)
```

Intuitively, A is a durative action that has a single precondition that P holds in its duration; B is a simple action that asserts P as its effect. Assume that `(not P)` holds initially, then the plan

$$\{(t_1 : A[d_1]), (t_1 : B)\} \tag{8.2}$$

is executable, since although the invariant condition formula of A was false before t_1 and the start event of A alone does not assert it, the simultaneous simple action B does so. If we consider the basic action theory Σ derived from this domain and the grounded simple plan of (8.2), however, we shall have

$$\begin{cases} \Sigma \models [B(t_1)][start(A, t_1)][end(A, t_1 + d_1)]Executable \\ \Sigma \models [start(A, t_1)][B(t_1)][end(A, t_1 + d_1)]\neg Executable \end{cases}$$

The cause of this paradox is the interleaved view of concurrency. In the second sentence above, there is a situation of duration 0 between the start event of A and the execution of B, where the invariant condition of A is violated. In the previous chapters, to avoid such a problem from occurring in our discussion, we strengthened Fox and Long's definition of *mutex actions* in Definition 7.12 to consider actions like $B(t_1)$ and $start(A, t_1)$ as mutex. However, they are in fact

since

$$\Sigma' \models [A_1(5)][B(5)]\Big(Obli\big(B(5)\big) \wedge now = time\big(B(5)\big) \leq 10\Big)$$

Then, for the second $\leq$, we have the counter example

$$\Sigma' \models [B(5)]\neg Poss\big(A_1(5)\big)$$

since

$$\Sigma' \models [B(5)]\Big(Obli\big(B(5)\big) \wedge now \leq time\big(B(5)\big) = time\big(A_1(5)\big)\Big)$$

non-interfering according to the standard semantics. As a result, a stronger formulation of concurrency is needed to capture the original semantics.

Although we do not give a formal proof here, it can be shown that if the situation calculus with true concurrency is used, as formalized by Reiter [Rei96], the both problems above can be solved. The question is, then, how we can reproduce this result in $\mathcal{ES}$, by extending it with true concurrency.

One natural way to accommodate true concurrency is to denote happenings with sets of actions. This is also the approach in Reiter's formalism. However, in order to use this approach, at least the two problems need to be considered:

1. *The syntax and the semantics of $\mathcal{ES}$ need to be changed.*
   In its current form, the language requires that the [] operator take a simple action as its argument; with true concurrency, the argument becomes a set of simple actions. As for semantics, the sequence of actions z becomes a sequence of sets of actions. So far, it is not clear how the extended syntax and semantics can be formally defined.

2. *Allowing for general sets makes the domain uncountable*
   A direct idea is to consider general sets as elements in the domain, but unfortunately, it makes the domain uncountable. This can be illustrated simply by a domain with all the subsets of an infinite set. This fact leads to a dilemma. On one hand, Reiter showed that the case of infinite actions happening concurrently must be taken seriously; on the other hand, a countably infinite domain is the foundation of many nice properties of $\mathcal{ES}$.

As a result, reasoning with true concurrency in the logic $\mathcal{ES}$ does not follow trivially from the existing work in the situation calculus. We believe that it is an interesting topic for further research.

8.2.3 Embedding External Planners in Golog

An ultimate goal of a bigger project containing the topic here is to bring the research in the areas of planning and action formalisms to a convergence, such that the work in one area may benefit from the ready results in the other. For example, one may embed external planners in the action languages, such as Golog. As introduced in Section 3.6, Golog is a powerful language that enables its user to write programs that constrain the search for an executable plan to a goal in a flexible way. However, when general planning is needed, Golog can only resort to its non-deterministic choice of actions, whose performance is far from competitive to the state-of-the-art planners.

Recently, Claßen *et al.* proposed a general method to integrate Adl planners, such as FF, into Golog programs, whose underlying basic action theory is a translation of an Adl problem description [CELN07]. Their result is based on the declarative semantics of the Adl subset of Pddl in the same paper. The general idea is like this: during the execution of the Golog program, when a goal that requires general planning is faced, a special procedure transforms the goal into a Pddl problem description, and activates the external planner to generate a plan. With the declarative semantics, it is guaranteed that valid plans in both cases are the same, so the returned plan is a valid one, and can thus be used directly in the Golog program.

As the major result, a declarative semantics for Pddl 2, a much larger subset of Pddl, is obtained in this thesis. The natural question is, can we extend the work in the Adl subset, and integrate external planners in the temporal concurrent Golog? We are optimistic in the prospect, and believe that the practical implementation of this idea will be an exciting future work on this topic.

Bibliography

[Can91] G. Cantor. Über ein elementare frage der mannigfaltigkeitslehre. 1891. 52

[CELN07] J. Claßen, P. Eyerich, G. Lakemeyer, and B. Nebel. Towards an integration of golog and planning. *IJCAI*, 2007. to appear. 2, 17, 48, 70, 95, 99, 136

[DGLL00] Giuseppe De Giacomo, Yves Lespérance, and Hector Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000. 2, 33

[EH04] Stefan Edelkamp and Joerg Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. report00195, Institut für Informatik, Universität Freiburg, January 21 2004. 1, 9, 15, 92, 129

[FL03] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124, 2003. 1, 8, 12, 13, 53, 114, 124, 127, 131

[FN71] R. Fikes and N. J. Nillson. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208, 1971. 1, 5

[FR97] Lin Fangzhen and Ray Reiter. How to progress a database. *Artificial Intelligence*, 92:131–167, 1997. 30

[GHK+98] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl—the planning domain definition language, 1998. 1, 8, 9

[GL00] Henrik Grosskreutz and Gerhard Lakemeyer. cc-Golog: Towards more realistic logic-based robot controllers. In *NMR-00*, 2000. 2, 33, 38

[GL01] Henrik Grosskreutz and Gerhard Lakemeyer. On-line execution of cc-Golog plans. In *IJCAI-01*, 2001. 2

[GL05] Alfonso Gerevini and Derek Long. Plan constraints and preferences in PDDL3. Technical report, University of Brescia, 2005. 9

[Lif86] Vladimir Lifschitz. On the semantics of strips. In Michael P. Georgeff and Amy Lansky, editors, *Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufmann, Los Altos, California, 1986. 2, 6

[LL01] Hector J. Levesque and Gerhard Lakemeyer. *The Logic of Knowledge Bases.* MIT Press, 2001. 110

[LL04] G. Lakemeyer and H. J. Levesque. Situations, si! situation terms, no! In *9th Conf. on Principles of Knowledge Representation and Reasoning (KR2004).* AAAI Press, 2004. 2, 43

[LL05] Gerhard Lakemeyer and Hector J. Levesque. Semantics for a useful fragment of the situation calculus. In *IJCAI-05*, 2005. 2, 43

[LR95] F. Lin and R. Reiter. How to progress a database ii: The strips connection. In *In Proc. IJCAI-95*, 1995. 67

[LRL$^+$97] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997. 2, 40

[McC63] John McCarthy. Situations, actions, and causal laws. Technical Report Memo 2, Stanford Artificial Intelligence Project, Stanford University, 1963. 2, 23

[Ped89] Edwin P. D. Pednault. Adl: exploring the middle ground between strips and the situation calculus. In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, pages 324–332, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. 1, 6

[Ped94] Edwin P. D. Pednault. Adl and the state transition model of action. *Logical Computing*, 4(5):467–512, 1994. 6

[Pin94] Javier Pinto. *Temporal Reasoning in the Situation Calculus.* PhD thesis, Department of Computer Science, University of Toronto, Toronto, Canada, January 1994. 2, 33, 37, 38, 56, 65

[PR99] Fiora Pirri and Ray Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 46(3):261–325, 1999. 27

[Rei91] Ray Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, 1991. 25

[Rei96] R. Reiter. Natural actions, concurrency and continuous time in the situation calculus. In *In Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)*, pages 2–13, Cambridge, Massachusetts, U.S.A., November 1996. 2, 33, 56, 65, 135

[Rei01] Raymond Reiter. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001. 2, 23, 29, 131

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003. 1

[Tar51] A. Tarski. A decision method for elementary algebra and geometry. 1951. 53