UNIVERSITY OF SOUTHAMPTON

On Reasoning about Action and Change in the Fluent Calculus

Helko Lehmann

A thesis submitted for the degree of Doctor of Philosophy

Declarative Systems and Software Engineering
Electronics and Computer Science
Faculty of Engineering and Applied Science

August 2001

UNIVERSITY OF SOUTHAMPTON
ABSTRACT
FACULTY OF ENGINEERING AND APPLIED SCIENCE

Doctor of Philosophy

On Reasoning about Action and Change in the Fluent Calculus
Helko Lehmann

We investigate the possibilities for automatic reasoning about action and change in the Fluent Calculus. To this end, by relating reasoning about action and change in the Fluent Calculus to model checking of dynamic systems, we pursue a systematic approach to analysing Fluent Calculus domains. Motivated by the different properties of Fluent Calculus domains known from the literature we define several Fluent Calculus fragments by syntactic criteria. We distinguish classes of dynamic properties to be inferred, focusing on several versions of planning problems. To apply results concerning the decidability of model checking of dynamic systems to decidability of reasoning about Fluent Calculus domains we establish tight relationships between models of the previously defined Fluent Calculus fragments and well known computational models like finite automata, Petri nets and two-counter machines. Furthermore, we show that dynamic properties, for example the existence of a plan, can be characterised by formulas of modal/temporal logics. Then, for every Fluent Calculus fragment and every class of dynamic properties we investigate the existence of a decision procedure. The results about decidability of all considered planning problems for the Fluent Calculus fragment $\mathcal{FC}_{PL}$ is particularly interesting. In $\mathcal{FC}_{PL}$ domains we can only use constant fluent and action symbols and the executability of actions must not depend on negative preconditions. Despite these restrictions, $\mathcal{FC}_{PL}$ allows the specification of systems with an infinite state space. With the help of our decidability results we develop a partial deduction algorithm to solve conjunctive planning problems for some Fluent Calculus domains. Our algorithm is the first complete reasoning method which can automatically solve conjunctive planning problems for $\mathcal{FC}_{PL}$ domains.

Acknowledgements

Doing a PhD was a journey for me, not only an intellectual and a personal but also a physical one. From 1997 to 1999 I was member of the Knowledge Representation and Reasoning group at the Dresden University of Technology and of the graduate school "Specification of discrete processes and systems of processes by operational models and logics". During this time my supervisor was Steffen Hölldobler. He is responsible for my interest in Artificial Intelligence and Deductive Systems. I am very grateful for his support, also after my decision to leave his group.

Since 1999 I was member of the Declarative Systems and Software Engineering group at the University of Southampton. Clearly, without the help of my new supervisor, Michael Leuschel, this work could never have been finished. His comments and ideas in many discussions were invaluable. With him I co-authored the papers which are basis of my thesis and he proof-read every line of this work. He was also a very challenging squash-partner.

I would like to thank Ulrich Ultes-Nitsche for his helpful comments on my Mini-thesis. As the head of the group Peter Henderson not only contributed to my work with important remarks but also provided a smooth working environment.

I am indebted to all the colleagues at the Knowledge Representation and Reasoning group in Dresden and at the Declarative Systems and Software Engineering group in Southampton. The members of both groups made my time in a research environment enjoyable. I would like to express my particular appreciation towards Sven-Erik Bornscheuer and Yvonne Kalinke, with whom I also co-authored some papers. I am thankful to Robert Walters who, as a native, improved the English in many parts of my thesis. In Dresden Hans-Peter Stoerr, who shared my fate of doing a PhD, was a very inspiring office mate. Stefan Hallerstede, my flat mate in Southampton, submitted his thesis on "Performance-Oriented Refinement" two weeks ago. He contributed to my work by giving advice and distracting me whenever necessary.

Although they did not contribute directly to my thesis, my family and my friends have been most important for my life in Dresden as well as in Southampton. In particular I wish to thank my parents and my grandmother for supporting me throughout my studies in every imaginable way. Among all the friends who turned the time of my studies into an exciting experience I would specially like to thank Lesley Leung. She shared many of the ups and downs during that time and often opened my eyes to the fact that a PhD is just a PhD.

Helko Lehmann
Southampton, August 2001

Contents

Introduction

During the last century the rapid development of computers has led to the automation of numerous tasks that have been previously considered to require "human intelligence". Most of these tasks can be characterised as *explicit* numerical and symbolical calculations in the sense that they are based on well established algorithmical models. For example, the differential calculus had been developed long before the construction of the first computer, that was capable of performing differentiation automatically. However, for most of the tasks humans perform in their everyday life, like oral and written communication, face recognition, or car driving, few satisfying algorithmical models exist. Additionally, our everyday life is subject to change and so too are the targets of our models. It has also been questioned whether some aspects of human behaviour can be modelled by symbol-manipulating machines at all, e.g. [27, 155, 115]. Consequently, the main objectives of research in *Artificial Intelligence (AI)* are 1) to investigate the principal possibilities for developing algorithmic models of certain aspects of human behaviour, and, if possible, 2) to provide such models [54]. Thereby, we will refer to those aspects of human behaviour which are subject of AI research as *intelligent*. This dynamic use of the word "intelligent" is in contrast to its use as an *universal* concept, i.e. a concept which can be justified without referring to human behaviour[1].

The only widely accepted way of determining whether a system models indeed intelligent behaviour is based on variants of the *Turing test* [153]. This test and its variants are designed to take as little internal structure of the system into account as possible. Basically, they refer to the structure that is required for communication with the system, only. As a result two main approaches for modelling intelligent behaviour have been developed. The first, the *neuropsychological*, approach aims to understand the biophysical processes of the "organ" brain, e.g. [132]. According to the hypothesis of *neurocomputing*, if the biophysical processes can be imitated by a computer, then it is also possible to imitate intelligent behaviour, [63, 77]. Following this approach, perceptions which are accessible to introspection, are understood as side effects of the biophysical processes. In contrast to the neuropsychological approach, *classical* AI avoids reference to underlying biophysical processes. Instead, its models are established

[1]We believe that the search for such an universal concept cannot be fruitful.

based on perceptions that are accessible by introspection, e.g. [153, 109]. The following work is dedicated to some aspects of this classical AI approach[2].

Instead of attempting to develop a model which contains concepts for all kinds of perceptions it has been argued, [109, 62], that it is reasonable to restrict the domain to those concepts upon which a vast majority of humans will agree. Later, these "common sense" concepts can be supplemented by more specific ones. "Common sense" concepts include in particular the way humans perceive changes of the world and act accordingly. AI research has concentrated on this aspect from its early beginnings [104]. One of the most important approaches towards understanding and modelling our perception of action and change was introduced in [105, 109], and christened *Situation Calculus*.

According to the classical AI approach intelligent systems consist of two components: an internal representation of knowledge about the world and a reasoning mechanism. The reasoning mechanism controls the behaviour of the system by deriving relevant knowledge from the internal representation. In [109] the two components are called *epistemological* and *heuristic*, respectively. The underlying assumption is called the *knowledge level hypothesis* [113]. Note that due to the distinction between knowledge about the world and the world itself, the classical AI approach implicitly assumes the *existence* of a world. Hence, philosophically speaking, classical AI is based on an *objectivistic* point of view [135].

Also based on the knowledge level hypothesis, *classical logic* has been developed as a model of human reasoning [43]. Thereby it provides "natural" languages to model our perception of action and change formally. In classical logic the relation between actual worlds and representations is called *semantics*. The heuristic component of a logic is called *calculus*. Thereby, the calculus should reflect the behaviour of the system that is possible in the world. Relying on these well established ideas, the name "Situation Calculus" refers to a class of logical second-order languages with extended first-order semantics. The second-order property of Situation Calculus languages is a result of allowing the specification of induction axioms, which require quantification over predicates. In this modern context the word "calculus" in the name "Situation Calculus" is actually misleading as in all recent work the name is not associated with a particular calculus. However, we keep the name for historical reasons.

In fact, most of the research about the Situation Calculus and other logic-based approaches has been dedicated to solving problems of knowledge representation, only. Despite of this deficit the logic-based models, which are usually called models of *Reasoning about Action and Change*, have superseded most of the alternative models proposed in the history of AI research.

Recently, the possibilities for automatic reasoning methods based on knowledge representations in Situation Calculus languages have received increased attention [119, 145]. One of the major reasons for the growing interest is the widespread opinion that the *frame problem* (see Section 2.1), which was consid-

[2]Note that there have also been attempts to relate the two approaches, e.g. [138, 136, 74].

ered to be one of the most fundamental problems of knowledge representation, is solved for some cases. Originating in the work [11] on the *connection method* a simple and elegant way to overcome the frame problem was proposed in [68] and later christened *Fluent Calculus*. Although the Fluent Calculus was first introduced as a logic programming scheme together with *SLDE-resolution* (see Section 1.3) as heuristic component, the association with a particular calculus has been discarded later. As with the Situation Calculus, the name has been kept for historical reasons. Furthermore, the class of logical languages which is defined by the Fluent Calculus can actually be seen as an extension of Situation Calculus languages [150]. At the time of writing little research about possibilities for automatic reasoning within Fluent Calculus domains has been undertaken, which takes the particular properties of knowledge representation in the Fluent Calculus into account. With this thesis we attempt to start filling this gap. However, the recent work of [67] and [69] is strongly related to our approach. We will discuss results of these papers in more detail later in this work. Although some early Prolog implementations have also been used for automatic reasoning, their capabilities are very limited (see Chapter 6).

Due to some of the second-order properties of Fluent Calculus languages and Situation Calculus languages we may apply *Gödel's theorem* [53] to knowledge representations using these languages. According to the theorem we cannot hope to find a calculus that allows us to derive (precisely) all consequences of the represented knowledge, i.e. every (sound) calculus must be *incomplete*. Despite of this inherent limitation of formal systems, the hope persists that many domains do not require the full power of the languages. For such domains complete calculi and effective decision procedures may exist. Consequently, the first goal of this work is to identify classes of Fluent Calculus domains which are "interesting" from a representational as well as from a computational point of view. However, we cannot know in advance how a difference in the representational properties of a domain determines its computational properties. To this end we will represent some well known Fluent Calculus domain classes in a common scheme and take the fragments defined by their syntactical differences as a best bet.

The second goal of this work is to investigate the previously identified Fluent Calculus fragments for possibilities of automatic reasoning. Thereby we will first attempt to prove decidability of reasoning for large classes of problems characterised by well known modal/temporal logics. If these attempts fail, i.e. no automatic reasoning procedures exist for the considered fragment, then we continue by investigating decidability of smaller classes of problems. In the research on Reasoning about Action and Change these latter classes of problems have received much attention under the name *planning problems* (see Section 2.1).

To achieve the second goal we will follow the approach of *model checking* of dynamic systems (see Section 2.7). The aim of model checking is to allow verification of formulas of some modal/temporal logic for a given dynamic system. To understand reasoning about action and change in the Fluent Calculus as a model checking problem, in every Fluent Calculus domain, we will distinguish between knowledge describing the behaviour of a particular dynamic system and properties we wish to infer. Then, depending on the the domain considered, the described dynamic system can be characterised by a well known model of computation, e.g. finite automaton, Petri net, counter machine, while Fluent Calculus formulas describing properties to be verified will be characterised by semantically equivalent formulas of modal/temporal logics. On one hand, this will allow us to prove (disprove) decidability of entailment problems by reducing to (from, respectively) known decision problems in model checking. On the other hand, we hope that the established relationships – between the Fluent Calculus and other well known models and languages – are of interest themselves.

As a third goal we will propose a method for implementing decision procedures for the Fluent Calculus which are particularly suitable to solve *conjunctive planning problems* (see Section 2.1). The method will improve the possibilities for automatic reasoning wrt SLDE-resolution and other previous approaches, which are incomplete even for Fluent Calculus fragments where conjunctive planning problems are decidable. To this end we will develop an *abstract partial deduction* method (see Section 7.1) which is particular suitable for logic programming representations of Fluent Calculus domains.

Structure of the Thesis

According to the main goals, the thesis is structured into three parts. The first part follows the Chapter 1 which briefly introduces notions and notations used throughout the work. This part focuses entirely on knowledge representation in the Fluent Calculus. We will discuss the underlying assumptions and relations to other approaches in detail. Syntax and semantics of the classes of Fluent Calculus domain descriptions considered in this work are defined. Furthermore, we will propose a characterisation of models in terms of *labelled transition systems*. This characterisation is used mainly in the second part, but it proves also fruitful in the first part when relations between some Fluent Calculus fragments are established. The second part is dedicated to the investigation of decidability of reasoning in the Fluent Calculus. In Chapter 3, we introduce some modal/temporal logics of decreasing expressive power and show how formulas of these logics characterise formulas in the Fluent Calculus. In Chapter 4 we show how models of certain Fluent Calculus domains correspond to *Petri nets*. The correspondences are established by means of *bisimulation*. Finally, in Chapter 5 we prove our decidability/undecidability results for each considered Fluent Calculus fragment and reasoning task. To this end our results concerning the relations between models of Fluent Calculus domains and Petri nets (or some other more well-known models of computation) are applied. In the first chapter of the last part of the thesis we show how some Fluent Calculus domains can be represented as *definite* E-*programs*. Then, in Chapter 7 we develop our abstract partial deduction procedure. We prove that it is complete for conjunctive planning problems in a particular Fluent Calculus fragment, once more by employing its relation to Petri nets. The last chapter concludes the work and sketches the possibilities for further research.

Chapter 1

Foundations

In this chapter we summarise essential definitions of logic and logic programming as they will be used throughout this thesis. The given definitions follow standard notions and notations used in the field of Reasoning about Action and Change (RAC) *and related areas of research. Notions and Notations that will be used but which are not commonly known in this field will be introduced in later chapters – usually right before their relations to RAC are discussed.*

1.1 Sorted First-Order Logic

In this section we introduce sorted first-order logic. Our presentation is mainly based on the one in [28]. It is slightly extended by allowing the specification of an ordering relation between between sorts (see below). Such extensions have been investigated in great detail, e.g. in [22, 44, 154].

First of all, we consider a logic to consist of a *syntax*, which characterises a formal language, and a *semantics*. The syntax defines which expressions are allowed. The semantics defines the meaning of these elements and a relation, called *entailment relation*, between them which describes when elements of the language are *logical consequences* of other elements.

Syntax

The syntax of sorted first-order logic is defined by a *signature*:

Definition 1.1.1 *A* signature *is a tuple* $(SORT, \preceq, FUN, REL)$ *where*

1. *$SORT$ refers to a finite set. Each element of $SORT$ is called* a sort.
2. *$\preceq$ is a partial order on the elements of $SORT$,*

3. *FUN is a finite set of elements called* function symbols. *Thereby every function symbol f is associated an expression of the form $f : S_1 \times \cdots \times S_n \to S$ with $S, S_1, \ldots, S_n \in SORT$ for some $n \in \mathbb{N}$.*

4. *REL is a finite set of elements called* predicate symbols. *Thereby every predicate symbol is associated an expression of the form $p : S_1 \times \cdots \times S_n$ with $S_1, \ldots, S_n \in SORT$ for some $n \in \mathbb{N}$.*

SORT, REL, FUN are disjoint. A function symbol $f :\to S$ with $S \in SORT$ is called a constant.

For example, consider the signature $\Sigma_Z = (\{Z, N\}, \{N \preceq Z\}, \{\mathrm{zero} :\to Z, succ : Z \to Z, \mathrm{pred} : Z \to Z\}, \{\mathrm{Less} : Z \times Z\})$.

A sorted first-order language is defined wrt some signature Σ and some *variable declaration*. Variables can be seen as "place-holders" of objects.

Definition 1.1.2 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature and N a finite set. Then a mapping $X : N \to SORT$ is a* variable declaration *wrt Σ. We denote a particular mapping of some variable $x \in N$ to $S \in SORT$ by $(x : S)$. Each element of N is a* variable name *(for short* variable*) wrt X, $N(X)$ denotes the set of all variable names wrt X.*

The objects of a first-order language are represented by the set of *terms* associated with the signature and the variable declaration. The structure of the terms is defined by *FUN*:

Definition 1.1.3 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X a variable declaration wrt Σ and $S \in SORT$. Then the set $T_{S,\Sigma}(X)$ of terms of sort S wrt Σ and X is inductively defined as follows:*

1. *Let $(x : S) \in X$. Then $x \in T_{S,\Sigma}(X)$.*

2. *Let $t_i \in T_{S_i,\Sigma}(X)$ for all $1 \leq i \leq m$ and $(f : S_1 \times \cdots \times S_m \to S) \in FUN$ with $m \in \mathbb{N}$. Then $f(t_1, \ldots, t_m) \in T_{S,\Sigma}(X)$.*

The terms in $T_{S,\Sigma}(\emptyset)$ are called ground.

The set of all terms wrt Σ and X is defined by

$$T_\Sigma(X) = \bigcup_{S \in SORT} T_{S,\Sigma}(X)$$

The variables of a term t *where $t \in T_{S,\Sigma}(X)$ are denoted by* Vars(t) *and inductively defined as:*

1. Vars(x) $= \{x\}$ *for* $(x : S) \in X$

2. *$Vars(f(t_1,\dots,t_n)) = Vars(t_1) \cup \dots \cup Vars(t_n)$ for $f : S_1 \times \dots \times S_n \to S_0 \in FUN$ for some $n \in \mathbb{N}$ and $n > 0$, if $n = 0$ $Vars(f) = \emptyset$.*

For example, the set $T_{Z,\Sigma_Z}(X)$ of terms of sort Z wrt Σ_Z and $X = \{(x : Z)\}$ is defined as $\{zero, succ(zero), pred(zero), succ(succ(zero)), pred(pred(zero)), \dots, x, succ(x), pred(x), succ(succ(x)), pred(pred(x)), \dots, succ(pred(x)), \dots\}$.

While *FUN* defines the structure of objects relations represent "atomic" propositions about objects:

Definition 1.1.4 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature and X a variable declaration wrt Σ. Then the set $A_\Sigma(X)$ of* atoms *wrt Σ and X is defined as follows:*

1. *Let $t_i \in T_{S_i,\Sigma}(X)$ for all $1 \le i \le m$, $m \in \mathbb{N}$ and $(p : S_1 \times \dots \times S_m) \in REL$. Then $p(t_1,\dots,t_m) \in A_\Sigma(X)$.*

The atoms in $A_\Sigma(\emptyset)$ are called ground. *The set of* variables *occurring in an atom $p(t_1,\dots,t_m)$ is defined as $Vars(p(t_1,\dots,t_m)) = \bigcup_{1 \le i \le m} Vars(t_i)$.*

One of the atoms wrt the above signature Σ_Z and X is $Less(succ(succ(zero)), x)$.

More complex formulas are defined by the use of *logical connectors* and *quantifiers*:

Definition 1.1.5 *Let Σ be a signature and X a variable declaration wrt Σ. Then the set $F_\Sigma(X)$ of* formulas *wrt Σ, X is inductively defined as follows:*

1. *Let $\phi \in A_\Sigma(X)$. Then $\phi \in F_\Sigma(X)$.*
2. *Let $\phi_1, \phi_2 \in F_\Sigma(X)$. Then $(\phi_1 \wedge \phi_2)$, $(\phi_1 \vee \phi_2)$, $(\phi_1 \Rightarrow \phi_2)$, $(\phi_1 \Leftrightarrow \phi_2)$, $(\neg\phi_1) \in F_\Sigma(X)$.*
3. *Let $(x : S) \in X$ and $\phi \in F_\Sigma(X)$. Then $\forall(x : S).\, \phi \in F_\Sigma(X)$ and $\exists(x : S).\, \phi \in F_\Sigma(X)$.*

By $atoms(\phi)$ we define the set of atoms which occur in the formula ϕ.

The set of free variables *$Vars(\phi)$ of a formula ϕ is defined as*

1. *if $\phi \in \{(\phi_1 \wedge \phi_2), (\phi_1 \vee \phi_2), (\phi_1 \Rightarrow \phi_2), (\phi_1 \Leftrightarrow \phi_2)\}$ then $Vars(\phi) = Vars(\phi_1) \cup Vars(\phi_2)$*
2. *if $\phi \in \{\exists(x : S).\, \phi_1, \forall(x : S).\, \phi_1\}$ $Vars(\phi) = Vars(\phi_1) \setminus \{x\}$*

A formula ϕ where $Vars(\phi) = \emptyset$ is called closed *or* sentence.

For example, the expression $\exists(x : N).\,(Less(zero, succ(x)) \wedge Less(pred(x), zero))$ is a formula wrt Σ_Z and X.

If ϕ is a formula with a set of free variables X and $\bar{Y}$ a sequence of variable names of X then we define a formula, denoted $\forall\bar{Y}.\,(\phi)$, to be *universally closed* wrt ϕ and $\bar{Y}$ if $\forall\bar{Y}.\,(\phi)$ is the result of adding for every variable x of $\bar{Y}$ a quantifier $\forall(x : S)$, where S is the appropriate sort of x, to ϕ. We define $\forall.\,(\phi)$ to denote the formula where $\bar{Y}$ contains all variables of X. Correspondingly, we also define $\exists.\,(\phi)$ and $\exists\bar{Y}.\,(\phi)$ to denote the *existentially closed* formula wrt ϕ (and $\bar{Y}$).

To increase readability brackets may be omitted by applying the following priorities between logical connectors (ordered wrt decreasing priority): $\neg$, $\forall$, $\exists$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$. Instead of writing $\forall(x_1 : S_1).\,(\forall(x_2 : S_2).\,(\ldots\forall(x_n : S_n).\,\phi\ldots))$ (or $\exists(x_1 : S_1).\,(\exists(x_2 : S_2).\,(\ldots\exists(x_n : S_n).\,\phi\ldots))$, respectively) for $n \geq 1$ we also write $\forall(x_1 : S_1), (x_2 : S_2), \ldots, (x_n : S_n).\,\phi$ ($\exists(x_1 : S_1), (x_2 : S_2), \ldots, (x_n : S_n).\,\phi$, respectively). A sequence of variable declarations $(x_1 : S), (x_2 : S), \ldots, (x_m : S)$ may be abbreviated by $(x_1, x_2, \ldots, x_n : S)$ for $m \geq 1$.

We also try to commit ourselves as much as possible to the following notational conventions:

Names for sorts: starting with capital letter, anonymous: S, S_1, S_2, ...

Names for variables: x, y, z, x_1, x_2, ..., x', x'', ...

Names for sets of variables or variable declarations: X, Y, ...

Names for functions: small letters, more than one, anonymous: f, g, h, f_1, f_2, ...

Names for terms: t, t_1, t_2, ...

Names for atoms: starting with capital letter, anonymous: p, q, r, p_1, p_2, ...

Names for formulas: ϕ, ψ, ϕ_1, ϕ_2, ...

Names for sets of formulas: `A`, `B`, ..., anonymous: L, L_1, ...

Semantics

A mapping that associates actual sets to *SORT* and actual functions and relations to *FUN* and *REL*, respectively, defines a meaning of ground atoms in a first-order language:

Definition 1.1.6 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature. An* interpretation *I of Σ is a mapping from Σ to*

1. *a family $\{S^I \mid S \in SORT\}$ of sets such that for $S_1, S_2 \in SORT$ $S_1^I \subseteq S_2^I$ iff $S_1 \preceq S_2$,*

2. *a family $\{f^I \mid f \in FUN\}$ of operations*

$$f^I : S_1^I \times \cdots \times S_n^I \to S_0^I \quad \textit{for } f : S_1 \times \cdots \times S_n \to S_0 \in FUN$$

3. *a family $\{p^I \mid p \in REL\}$ of relations*

$$p^I \subseteq S_1^I \times \cdots \times S_n^I \quad \textit{for } p : S_1 \times \cdots \times S_n \in REL$$

The universe *of an interpretation I is defined by*

$$SORT^I = \bigcup_{S \in SORT} S^I$$

For example, we may consider the interpretation I of Σ_Z where $N^I = \mathbb{N}$, $Z^I = \mathbb{Z}$, $zero^I = 0$, $succ^I = \{x \mapsto x + 1 \mid x \in \mathbb{Z}\}$, $pred^I = \{x \mapsto x - 1 \mid x \in \mathbb{Z}\}$ and $Less^I = \{(x, y) \mid x < y \text{ and } x, y \in \mathbb{Z}\}$.

The meaning of a free variable is defined by a mapping to an actual object of appropriate sort:

Definition 1.1.7 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X some variable declaration, I an interpretation of Σ and $t \in T_{S,\Sigma}(X)$. Let $\lambda : N(X) \to SORT^I$ be a mapping such that $\lambda(x) \in S^I$ for $(x : S) \in X$. Then the* value $t^{I,\lambda}$ *is inductively defined as follows*

1. $x^{I,\lambda} = \lambda(x)$ *for* $(x : S) \in X$,
2. $f(t_1, \ldots, t_n)^{I,\lambda} = f^I(t_1^{I,\lambda}, \ldots, t_n^{I,\lambda})$ *for* $f : S_1 \times \cdots \times S_n \to S_0 \in FUN$ *with* $t_i \in T_{S_i,\Sigma}(X)$, $i = 1, \ldots, n$, $n \in \mathbb{N}$.

Now we can assign *truth values*, $\top$ and $\bot$, to formulas wrt some interpretation:

Definition 1.1.8 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X some variable declaration, I an interpretation of Σ and $\phi \in F_\Sigma(X)$. Let $\lambda : N(X) \to SORT^I$ be a mapping such that $\lambda(x) \in S^I$ for $(x : S) \in X$. Then the* value *of $\phi^{I,\lambda}$ is inductively defined as follows*

1. $p(t_1, \ldots, t_n)^{I,\lambda}$ *is* $\top$ *for* $p : S_1 \times \cdots \times S_m \in REL$, $t_i \in T_{S_i,\Sigma}(X)$, $i = 1, \ldots, n$, $n \in \mathbb{N}$, *iff* $(t_1^{I,\lambda}, \ldots, t^{I,\lambda}) \in p^I$, $\bot$ *otherwise,*
2. $(\neg\phi)^{I,\lambda}$ *is* $\top$ *iff* $\phi^{I,\lambda}$ *is* $\bot$,
3. $(\phi_1 \wedge \phi_2)^{I,\lambda}$ *is* $\top$ *iff* $\phi_1^{I,\lambda}$ *is* $\top$ *and* $\phi_2^{I,\lambda}$ *is* $\top$,
4. $(\phi_1 \vee \phi_2)^{I,\lambda}$ *is* $\top$ *iff either* $\phi_1^{I,\lambda}$ *is* $\top$ *or* $\phi_2^{I,\lambda}$ *is* $\top$ *(or both),*
5. $(\phi_1 \Rightarrow \phi_2)^{I,\lambda}$ *is* $\top$ *iff either* $\phi_1^{I,\lambda}$ *is* $\bot$ *or* $\phi_2^{I,\lambda}$ *is* $\top$ *(or both),*

6. $(\phi_1 \Leftrightarrow \phi_2)^{I,\lambda}$ *is* $\top$ *iff* $\phi_1^{I,\lambda}$ *is* $\top$ *and* $\phi_2^{I,\lambda}$ *is* $\top$, *or* $\phi_1^{I,\lambda}$ *is* $\bot$ *and* $\phi_2^{I,\lambda}$ *is* $\bot$,
7. $(\forall(x:S).\,\phi)^{I,\lambda}$ *is* $\top$ *iff* $\phi^{I,\lambda'}$ *is* $\top$ *for all* $\lambda' : N(X) \to SORT^I$ *such that* $\lambda'(y) = \lambda(y)$ *for all* $(y : S') \neq (x : S)$ *of* X,
8. $(\exists(x:S).\,\phi)^{I,\lambda}$ *is* $\top$ *iff there exists a* $\lambda' : N(X) \to SORT^I$ *such that* $\phi^{I,\lambda'}$ *is* $\top$ *and* $\lambda'(y) = \lambda(y)$ *for all* $(y : S') \neq (x : S)$ *of* X.

For example, the formula $\exists(x : N).\,(Less(zero, succ(x)) \wedge Less(pred(x), zero))$ is assigned $\top$ in the interpretation I, since there is a mapping λ which maps x to 0.

Interpretations can be classified according to the values they assign to formulas:

Definition 1.1.9 *Let* $\Sigma = (SORT, \preceq, FUN, REL)$ *be a signature. An interpretation* I *of* Σ *in which some formula* ϕ *is assigned* $\top$ *is called a* model *of* ϕ, *denoted* $I \models \phi$. *If there is an interpretation* I *such that* $I \models \phi$ *then* ϕ *is called* satisfiable, *otherwise* ϕ *is called* unsatisfiable. *A formula* ϕ *is a* consequence *of a set of formulas* $\mathtt{F}$, *denoted* $\mathtt{F} \models \phi$ *if* ϕ *is assigned* $\top$ *in all interpretations where each formula of* $\mathtt{F}$ *is assigned* $\top$. *If* ϕ *is a consequence of* $\emptyset$ *then* ϕ *is called* valid.

For every signature, there exist interpretations where the interpretation of each sort is simply defined by the set of terms of this sort. In these cases the interpretation of the function symbols is given by the corresponding mappings on terms. The only differences between such interpretations result from different interpretations of the relations:

Definition 1.1.10 *Let* $\Sigma = (SORT, \preceq, FUN, REL)$ *be a signature. The* Herbrand-universe $U_\Sigma^{\mathcal{H}}$ *is the set of all ground terms of* Σ:

$$U_\Sigma^{\mathcal{H}} = \bigcup_{S \in SORT} T_{S,\Sigma}(\emptyset)$$

The set of all ground atoms is called the Herbrand-base *of* Σ. *An interpretation* I *of* Σ *is called a* Herbrand-interpretation *if*

1. $S^I = T_{S,\Sigma}(\emptyset)$ *for all* $S \in SORT$
2. $f^I(t_1, \dots, t_n) = f(t_1, \dots, t_n)$ *for all* $t_i \in S_i^I$, $i = 1, \dots, n$ *and for all* $f : S_1 \times \dots \times S_n \to S_0 \in FUN$

An Herbrand-interpretation that is a model for some formula ϕ *is called an* Herbrand-model *of* ϕ.

We may define a calculus formally wrt a logic:

Definition 1.1.11 *A* calculus *C consists of a logic $\mathcal{L}$, a set* A *of formulas of $\mathcal{L}$, called* axioms, *and a set called* inference rules. *The inference rules define a* derivability relation $\vdash$ *between elements of the language.*

Let $\models$ represent the entailment relation of $\mathcal{L}$. Then C is called complete *wrt $\mathcal{L}$ iff whenever* $\mathtt{A} \models \phi$, $\mathtt{A} \vdash \phi$ *and C is called* sound *wrt $\mathcal{L}$ iff whenever* $\mathtt{A} \vdash \phi$, $\mathtt{A} \models \phi$.

First-order logic has many sound and complete calculi, e.g. *Gentzen* and *Hilbert* systems. However, first-order logic is only *semi-decidable*, i.e. the valid formulas are recursively enumerable while the non-valid formulas are not.

For further details on first-order logic in general the reader is referred to, e.g. [42].

Equational Theories

It is often convenient to provide relations which characterise when two objects are equivalent. In this work we will use such relations, called *equations*, in particular to state dependencies between subsequent states of dynamic systems. E.g. in physics, *difference equations* are often used in a very similar way.

An equation is represented by a binary predicate symbol. The properties which ensure that a predicate indeed represents an equivalence relation are ensured by the well known axioms given in the following definition. To emphasise these particular properties it is common to write equation symbols in infix notation.

Definition 1.1.12 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X an appropriate variable declaration wrt Σ, $S \in SORT$ and $=_S : S \times S \in REL$. The Axioms*

$$\forall(x : S).\, x =_S x \qquad \textit{(reflexivity)}$$

$$\forall(x, y : S).\, (x =_S y \Rightarrow y =_S x) \qquad \textit{(symmetry)}$$

$$\forall(x, y, z : S).\, (x =_S y \land y =_S z \Rightarrow x =_S z) \qquad \textit{(transitivity)}$$

$$\begin{array}{l}\forall(x_1 : S_1), \ldots, (x_i, y : S_i), \ldots, (x_m : S_m). \\ \quad (x_i =_S y \Rightarrow f(x_1, \ldots, x_i, \ldots, x_m) =_S f(x_1, \ldots, y, \ldots, x_m))\end{array} \qquad \textit{(substitutivity I)}$$

for all $f : S_1 \times \cdots \times S_m \to S \in FUN$ for $1 \leq i \leq m$, $m \in \mathbb{N}$ with $S_i = S$.

$$\begin{array}{l}\forall(x_1 : S_1), \ldots, (x_i, y : S_i), \ldots, (x_m : S_m). \\ \quad (x_i =_S y \Rightarrow p(x_1, \ldots, x_i, \ldots, x_m) \Leftrightarrow p(x_1, \ldots, y, \ldots, x_m))\end{array} \qquad \textit{(substitutivity II)}$$

for all $p : S_1 \times \cdots \times S_m \in REL$ for $1 \leq i \leq m$ with $S_i = S$, are called the standard equality axioms *wrt $=_S$.*

Note that (substitutivity I) and (substitutivity II) are actually axiom schemata rather then axioms, since they are meant to hold for every function and relation symbol of the signature.

It is often necessary to specify properties additionally to the standard equality axioms. In this work it will be sufficient to consider only properties that can be represented according to the following schema.

Definition 1.1.13 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X a variable declaration wrt Σ and $S \in SORT$. An* equational theory E_S *for a predicate $=_S\, : S \times S \in REL$ consists of the standard equality axioms for $=_S$ and a set of formulas of the form*

$$\forall . (s =_S t)$$

where $s, t \in T_{S,\Sigma}(X)$.

If $\mathrm{E}_S \models s =_S t$ for some terms $s, t \in T_{S,\Sigma}(x)$ we write also $s =_{\mathrm{E}_S} t$.

The equational theory where the set of formulas of the above form is empty is called standard equational theory.

Example 1.1.1 (Σ_Z continued) *Let Σ_Z be extended by the predicate $=_Z$: $Z \times Z$. The formula*

$$\forall (x : Z). pred(succ(x)) =_Z succ(pred(x))$$

together with the standard equality axioms forms an equational theory. □

It has been pointed out that axioms which define properties of equational relations often cause trouble in deductive systems, e.g. if the set of substitutions solving a *unification problem* (see below) is infinite. Consequently, it has been proposed to treat equations in a special way. To this end we distinguish those interpretations which take already the standard equality axioms into account. Similarly, we distinguish interpretations which are models for the considered equational theory.

Definition 1.1.14 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature. An* E-interpretation *of Σ is an interpretation I where for each $S \in SORT$ $=_S{}^I$ with $=_S\, : S \times S \in REL$ is given by some equivalence relation over S^I.*

An Herbrand-E-interpretation *is an E-interpretation I where for each $S \in SORT$ S^I is defined by the congruence classes of an arbitrary congruence relation over $T_{S,\Sigma}(\emptyset)$.*

If an E-interpretation is a model for $\mathrm{E} = \bigcup_{S \in SORT} \mathrm{E}_S$ then it is called E-model. *Correspondingly, an Herbrand-E-interpretation which is model for E is called an* Herbrand-E-model.

A formula ϕ is called E-valid *(*E-unsatisfiable, *respectively) iff ϕ is true (false, respectively) in every E-model.*

Then, since we will only require one non-standard equational theory, we can apply the following result which has been first shown in [72]:

Proposition 1.1.1 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X a variable declaration wrt Σ, $\phi \in F_\Sigma(X)$ and E_S an equational theory for some predicate of Σ and some $S \in SORT$. Then with $\mathrm{E} = \mathrm{E}_S$, $\mathrm{E} \models \phi$ iff ϕ is E-valid and $\mathrm{E} \cup \phi$ is unsatisfiable iff ϕ is E-unsatisfiable.*

Unification

To avoid the difficulties caused by the use of equations it has been shown in [121], following the idea of [131], how equational theories can be successfully built into the *unification procedure*. In this subsection we introduce the notions of unification theory we rely on. The goal of unification is to find for two atoms a set of variable instantiations such that the atoms become identical. Variable instantiations are represented by *substitutions*:

Definition 1.1.15 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature and X a variable declaration wrt Σ. A* substitution *σ wrt Σ and X is a mapping $N(X) \to T_\Sigma(X)$, such that for all $(x : S) \in X$ $\sigma(x) \in T_{S,\Sigma}(X)$ and only for a finite number of $x \in N(X)$ holds $x \neq \sigma(x)$. $\sigma(x)$ is also denoted as $x\sigma$. The set*

$$Dom(\sigma) = \{x \mid x \neq x\sigma\}$$

is called domain *of the substitution σ.*

The restriction *of a substitution σ to a variable declaration Y, denoted $\sigma|_Y$ is defined as $x(\sigma|_Y) = x\sigma$ for $x \in N(Y)$ and $x(\sigma|_Y) = x$ otherwise.*

The composition *of two substitutions σ and θ wrt Σ and X, denoted as $\sigma\theta$, is defined as $x(\sigma\theta) = (x\sigma)\theta$.*

The empty substitution is denoted by ϵ.

A substitution may be applied to a term (or a formula, respectively) and by this transform it into one of its *instances*.

Definition 1.1.16 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X a variable declaration wrt Σ. Then the* application *of a substitution σ wrt Σ and X to a term $t \in T_\Sigma(X)$, written as $t\sigma$ is defined for $t = f(t_1, \ldots, t_n)$ with $f : S_1 \times \cdots \times S_n \to S_0 \in FUN$ as $t\sigma = f(t_1\sigma, \ldots, t_n\sigma)$.*

Furthermore, let $\phi \in F_\Sigma(X)$. Then the application *of σ wrt Σ and X to ϕ, written as $\phi\sigma$ is defined as*

1. *if $\phi = p(t_1, \ldots, t_m) \in A_\Sigma(X)$ then $\phi\sigma = p(t_1\sigma, \ldots, t_m\sigma)$,*
2. *if $\phi = (\neg\phi)$ then $\phi\sigma = \neg(\phi\sigma)$,*
3. *if $\phi = (\phi_1 \; op \; \phi_2)$ then $\phi\sigma = (\phi_1\sigma \; op \; \phi_2\sigma)$ for $op \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$,*
4. *if $\phi \in \{\exists(x : S).\, \phi_1, \forall(x : S).\, \phi_1\}$ then $\phi\sigma \in \{\exists(x : S).\, \phi_1\sigma|_{Dom(\sigma)\setminus\{x\}}, \forall(x : S).\, \phi_1\sigma|_{Dom(\sigma)\setminus\{x\}}\}$, respectively.*

A term t_1 (formula ϕ_1) is called an instance *of some term t_2 (formula ϕ_2) if there is a substitution σ such that $t_2\sigma = t_1$ ($\phi_2\sigma = \phi_1$).*

E.g., $succ(succ(succ(y)))$ is an instance of $succ(x)$ as with substitution $\sigma = \{x \mapsto succ(succ(y))\}$ holds $succ(x)\sigma = succ(succ(succ(y)))$.

Note that the results of applying substitutions according to the above definition may be unexpected. E.g, consider the two formulas $P(x) \wedge \exists(y : Z). Q(x, y)$ and $P(x) \wedge \exists(y' : Z). Q(x, y')$. The formulas are equivalent up to renaming of the non-free variable. However, if we apply the substitution $\sigma = \{x \mapsto y\}$ the resulting formulas are no longer equivalent up to renaming: $(P(x) \wedge \exists(y : Z). Q(x, y))\sigma = P(y) \wedge \exists(y : Z). Q(y, y)$ and $(P(x) \wedge \exists(y' : Z). Q(x, y'))\sigma = P(y) \wedge \exists(y' : Z). Q(y, y')$.

If some formula ϕ is an instance of some formula ψ then ψ is also called *more general* then ϕ. If ϕ is more general then ψ and ψ is more general then ϕ then ϕ and ψ are called *variants* (of each other).

Often it is not feasible to consider the large (or even infinite) number of all possible substitutions. Instead the number of substitutions can be restricted to those that do not produce the same instances if applied. Additionally, different terms resulting from applying different substitutions might also be equivalent if we take some underlying equational theory into account. Hence, to reduce the number of substitutions further we introduce a generalised notion of the "instance-of" relation between substitutions.

Definition 1.1.17 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X a variable declaration wrt Σ, E_S an equational theory for some predicate $=_S$ of Σ with $S \in SORT$ and some $Y \subseteq X$. Then two substitutions σ and θ wrt Σ and X are called* E_S-equivalent *wrt Y, denoted $(\sigma =_{\mathrm{E}_S, Y} \theta)$, if*

$$x\sigma =_{\mathrm{E}_S} x\theta \quad \textit{for all } x \in N(Y).$$

The substitution θ is called E_S-instance *of σ wrt Y, denoted $(\sigma \leq_{\mathrm{E}_S, Y} \theta)$ if there exists a substitution ρ wrt Σ and X, such that*

$$(x\theta =_{\mathrm{E}_S, Y} x\sigma\rho) \quad \textit{for all } x \in N(Y).$$

Now we define the unification problem wrt an equational theory.

Definition 1.1.18 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X a variable declaration and E_S be an equational theory for some predicate $=_S : S \times S$ of Σ. An* E_S-unification problem *consists of two terms $s, t \in T_{S,\Sigma}(X)$ and the question whether there exists a substitution σ with $Dom(\sigma) \subseteq Vars(s, t)$, such that*[1]

$$s\sigma =_{\mathrm{E}_S} t\sigma$$

[1]By $Vars(s, t)$ we denote the set $Vars(s) \cup Vars(t)$.

A substitution, which is solution to an E_S*-unification problem of* s *and* t *is called* E_S-unifier *for* s *and* t*. Two terms are* E_S-unifiable *if there exists an* E_S*-unifier for them.*

A term s *is an* E_S-instance *of a term* t*, denoted* $s \leq_{E_S} t$*, iff there is a substitution* σ *with* $s =_{E_S} \sigma t$*.*

If we consider the equational theory of Example 1.1.1 (denoted by E_Z) then, e.g., $succ(pred(zero)) \leq_{E_Z} pred(succ(x))$. Thereby the E_Z-unifier is $\{x \mapsto zero\}$.

We may classify unification algorithms according to the set of unifiers they compute. Algorithms that compute representations of all possible unifiers are particularly important. For certain equational theories such algorithms may not exist, e.g. if it is not decidable whether two terms are unifiable at all.

Definition 1.1.19 *Let* $\Sigma = (SORT, \preceq, FUN, REL)$ *be a signature,* X *a variable declaration wrt* Σ*,* E_S *an equational theory for some predicate of* Σ *and some* $S \in SORT$*. Furthermore, let* $U_{E_S}(s,t)$ *be the set of all* E_S*-unifiers of some terms* $s, t \in T_\Sigma(X)$*. Then the set* $U \subseteq U_{E_S}(s,t)$ *is called a* complete *set of* E_S*-unifiers, if for all* $\theta \in U_{E_S}(s,t)$*, there exists* $\sigma \in U$ *such that* $\sigma \leq_{E_S, Vars(s,t)} \theta$*.*

If U *is complete and for all* $\theta, \sigma \in U$*,* $\sigma \leq_{E_S, Vars(s,t)} \theta$ *implies* $\sigma =_{E_S, Vars(s,t)} \theta$*, then it is called* minimal.

An unification algorithm P *is called* complete *(*minimal*) if* P *computes* U *(a minimal set of* E*-unifiers) for arbitrary* $s, t \in T_\Sigma(X)$*.*

Note that minimal sets of E_S*-unifiers are always unique up to variable renaming if they exist. Hence, we denote the minimal* E_S*-unifier of* s *and* t *by* $\mu U_{E_S}(s,t)$*. We call a substitution in* $\mu U_{E_S}(s,t)$ *a* most general E_S-unifier *(mgeu) of* s *and* t*.*

An equational theory E_S *is called* decidable *if for any two terms* $s, t \in T_\Sigma(X)$ $s =_{E_S, Vars(s,t)} t$ *is decidable.*

An equational theory E_S *is called* finitary *if for any two terms* $s, t \in T_\Sigma(X)$*, there exists a finite complete set of* E_S*-unifiers.*

Example 1.1.2 *Let* Σ'_Z *contain all elements of* Σ_Z *of Example 1.1.1 and additionally a function* $mult : Z \times Z \to Z$ *and a predicate* $=_Z : Z \times Z$*. Consider the equational theory to be defined by*

$$\forall (x, y, z : Z).\, mult(x, mult(y, z)) =_Z mult(mult(x, y), z)$$

and the standard equality axioms.

Now consider the terms $s = mult(x, zero)$*,* $t = mult(zero, y)$*. The substitution* $\theta = \{x \mapsto zero, y \mapsto zero\}$ *is an* E*-unifier for* s *and* t*, and so is* $\nu = \{x \mapsto mult(zero, z), y \mapsto mult(z, zero)\}$*. Furthermore,* $\{\theta, \nu\}$ *is a complete set of* E*-unifiers and, since* θ *and* ν *are incomparable under* $\leq_{E_S}$ *the set is also minimal.*

□

For further details, e.g. on particular equational theories and unification theory see e.g. [139, 5].

Unification Completeness

Using the standard equality axioms and the axioms of the equational theory for some predicate $=_S$, does not automatically allow us to conclude when two terms are actually not equal. I.e. from $s \neq_{\mathrm{E}_S} t$ for some terms s, t and some equational theory E_S wrt sort S we can not follow that $\mathrm{E}_S \models \neg s =_S t$ (which we also denote as $\mathrm{E}_S \models s \neq_S t$). To this end additional axioms have to be provided. The first correct approach in [137] that is independent of a particular equational theory has lead to the following notion of *unification completeness*:

Definition 1.1.20 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X a variable declaration, E_S an equational theory wrt some predicate $=_S$ of Σ and some $S \in SORT$. A consistent set of formulas E^*_S is called* unification complete *wrt E_S if E^*_S implies the formulas of E_S and for arbitrary $s, t \in T_{S,\Sigma}(X)$ the following holds*

1. *if s and t are not E_S-unifiable, then $\mathrm{E}^*_S \models \neg\exists.\,(s =_S t)$*
2. *for every complete set U of E_S-unifiers for s and t*

$$\mathrm{E}^*_S \models \forall.\,(s =_S t \Rightarrow \bigvee_{\theta \in U} \exists \bar{Z}_\theta.\,(\theta_{=_S}))$$

where $Vars(s,t) = \{y_1, \ldots, y_n\}$, $\theta_{=_S} = y_1 =_{S_1} r_1 \wedge \cdots \wedge y_m =_{S_m} r_m$ with $z_i\theta = r_i$ for all $i = 1, \ldots, m$ and $\bar{Z}_\theta$ is a sequence of variables $z_1, \ldots, z_k$ with $Vars(r_1) \cup \cdots \cup Vars(r_k) = \{z_1, \ldots, z_k\}$.

In other words, $s =_S t$ is valid only if this is justified by a unifier in U. $s \neq_S t$ is valid whenever U is empty.

For some equational theories the existence of unification complete theories has been shown [70]:

Proposition 1.1.2 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X a variable declaration, E_S a decidable equational theory wrt some predicate of Σ and some $S \in SORT$, P a complete E_S-unification algorithm. Then there exists a unification complete theory E^*_S for E_S.*

If only Herbrand-models are considered a unification complete theory induces a particular congruence relation on the universe [146]:

Proposition 1.1.3 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X a variable declaration and E_S a decidable equational theory wrt some predicate of Σ and some $S \in SORT$. Let E^*_S be a unification complete theory for E_S. Then every Herbrand-model of E^*_S induces a smallest congruence relation on the Herbrand-universe wrt E_S.*

1.2 Higher-Order Logics

In this work we will also require statements of properties which are not expressible in any first-order language. To this end we will refer occasionally to a second-order language corresponding to some signature. In the following we briefly introduce the necessary extensions of syntax and semantics.

Syntax

Second-order languages allow additional use of variables representing predicates and functions:

Definition 1.2.1 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature. Then a mapping $X : N \to (S_1 \times \cdots \times S_n)$ with $S_1, \ldots, S_n \in SORT$ and $n \in \mathbb{N}$ (a mapping $X : N \to (S_1 \times \cdots \times S_n \to S)$ with $S, S_1, \ldots, S_n \in SORT$ and $n \in \mathbb{N}$) is a* declaration of predicate variables *wrt Σ (a* declaration of function variables wrt Σ*). We denote a particular mapping of some variable $x \in N$ to $S_1 \times \cdots \times S_n$ ($x \in N$ to $S_1 \times \cdots \times S_n \to S$) by $(x : S_1 \times \cdots \times S_n)$ (by $(x : S_1 \times \cdots \times S_n \to S)$). Each element of N is a* predicate variable name *(*function variable name*) wrt X, $N(X)$ denotes the set of all variable names wrt X.*

Correspondingly, the set of terms of a second-order language is inductively defined as in the first-order case but additionally to the application of functions function variables can be used for the construction of terms.

Definition 1.2.2 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature and X consist of a variable declaration and a variable declaration of functions wrt Σ and $S \in SORT$. Then the set $T_{S,\Sigma}(X)$ of terms of sort S wrt Σ and X is inductively defined as in Definition 1.1.3 and additionally:*

3. *Let $t_i \in T_{S_i,\Sigma}(X)$ for all $1 \leq i \leq m$ and $(\phi : S_1 \times \cdots \times S_m \to S) \in X$ with $m \in \mathbb{N}$. Then $\phi(t_1, \ldots, t_m) \in T_{S,\Sigma}(X)$.*

Accordingly, the definitions of *ground* terms, $T_\Sigma(\_)$ and *Vars*(_) are extended. Using predicate variables new atomic sentences can be built:

Definition 1.2.3 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature and X consist of a variable declaration, a variable declaration of functions and a variable declaration of predicates wrt Σ. Then the set $A_\Sigma(X)$ of* atoms *wrt Σ and X is inductively defined as in Definition 1.1.4 and additionally:*

2. *Let $t_i \in T_{S_i,\Sigma}(X)$ for all $1 \leq i \leq m$, $m \in \mathbb{N}$ and $(\Phi : S_1 \times \cdots \times S_m) \in X$. Then $\Phi(t_1, \ldots, t_m) \in A_\Sigma(X)$.*

We also extend *ground* atoms and *Vars*(_) accordingly.

Definition 1.2.4 *Let Σ be a signature and X consist of a variable declaration, a variable declaration of functions and a variable declaration of predicates wrt Σ. Then the set $F_\Sigma(X)$ of* formulas *wrt Σ, X is defined as in Definition 1.1.5 and additionally:*

4. *Let $(\Phi : S_1 \times \cdots \times S_m) \in X$, $m \in \mathbb{N}$ and $\phi \in F_\Sigma(X)$. Then the formulas $\forall(\Phi : S_1 \times \cdots \times S_m).\,\phi$ and $\exists(\Phi : S_1 \times \cdots \times S_m).\,\phi$ are also in $F_\Sigma(X)$.*
5. *Let $(\Phi : S_1 \times \cdots \times S_m \to S_0) \in X$ and $\phi \in F_\Sigma(X)$. Then the formulas $\forall(\Phi : S_1 \times \cdots \times S_m \to S_0).\,\phi$ and $\exists(\Phi : S_1 \times \cdots \times S_m \to S_0).\,\phi$ are also in $F_\Sigma(X)$.*

Again, we assume also the definitions of *free variables* and *closed formulas* to be extended.

Semantics

As in the first-order case an *interpretation* associates actual sets to *SORT* and actual functions and relations to *FUN* and *REL*. However, the meaning of free predicate and function variables has to be defined. Then we can assign *truth values* to formulas wrt some interpretation:

Definition 1.2.5 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X consist of a variable declaration, a variable declaration of predicates and a variable declaration of functions wrt Σ, I an interpretation of Σ and $t \in T_{S,\Sigma}(X)$. Let λ be a mapping such that $\lambda(x) \in S^I$ for $(x : S) \in X$, $\lambda(\Phi) \subseteq S_1^I \times \cdots \times S_m^I$ for $(\Phi : S_1 \times \cdots \times S_m)$ with $m \in \mathbb{N}$ and $\lambda(\Phi)$ is a mapping $S_1^I \times \cdots \times S_m^I \to S_0^I$ for $(\Phi : S_1 \times \cdots \times S_m \to S_0)$ with $m \in \mathbb{N}$. Then the* value *$t^{I,\lambda}$ is inductively defined as in Definition 1.1.7 and additionally:*

3. *$\Phi(t_1, \ldots, t_n)^{I,\lambda} = \Phi^{I,\lambda}(t_1^{I,\lambda}, \ldots, t_n^{I,\lambda})$ for $(\Phi : S_1 \times \cdots \times S_n \to S_0) \in X$ with $t_i \in T_{S_i,\Sigma}(X)$, $i = 1, \ldots, n$, $n \in \mathbb{N}$.*

Let $\phi \in F_\Sigma(X)$. Then the value *of $\phi^{I,\lambda}$ is inductively defined as in Definition 1.1.8 and additionally:*

9. *$\Phi(t_1, \ldots, t_n)^{I,\lambda}$ is $\top$ for $(\Phi : S_1 \times \cdots \times S_m) \in X$, $t_i \in T_{S_i,\Sigma}(X)$, $i = 1, \ldots, n$, $n \in \mathbb{N}$, iff $(t_1^{I,\lambda}, \ldots, t^{I,\lambda}) \in \Phi^{I,\lambda}$, $\bot$ otherwise,*
10. *$\forall(\Phi : \mathcal{S}).\,\phi$ is $\top$ iff $\phi^{I,\lambda'}$ is $\top$ for all λ' such that $\lambda'(\Psi) = \lambda(\Psi)$ for all $(\Psi : \mathcal{S}') \neq (\Phi : \mathcal{S})$ of X where $\mathcal{S}$, $\mathcal{S}'$ denote either $S_1 \times \cdots \times S_m$ with $S_1, \ldots, S_m \in SORT$, or $S_1 \times \cdots \times S_m \to S_0$ with $S_0, S_1, \ldots, S_m \in SORT$.*

The definitions of the other quantifiers can be derived accordingly. Interpretations can be classified according to the values they assign to formulas as in the first-order case.

Modal and Temporal Logics

We will also refer to logics which are, wrt classical first-order logic, extended by additional operators, called *modalities* (originally introduced in [97], with satisfactory semantics in [81]). Some of these logics are particularly important for this work, in particular so called *temporal logics* based on the ideas in [125]. Hence, we will discuss the particular syntax and semantics of the considered logic in more detail in Chapter 3. However, for more information, also on the historical background and the relations between modal and temporal logics, the reader is referred to e.g. [141, 9].

1.3 Logic Programs with Equality

We will consider only logic programs that are extended by an equational theory and do not allow the use of negation.

Definition 1.3.1 *Let Σ be a signature and X a variable declaration. A* definite clause *is an expression of the form*

$$A \leftarrow B_1, \ldots, B_m$$

where A, called the head, *and $B_1, \ldots, B_m$, called the* body, *are atoms of $A_\Sigma(X)$.*

A definite logic program with equational theory *(for short:* definite E-program*) is a pair $(D, \mathtt{E})$ where $\mathtt{E}$ is an equational theory for predicate $=_S$ and D is a finite set of definite clauses where $=_S$ does not appear in the head.*

Since the equational relation $=_S$ should be exclusively defined by the associated equational theory E clauses that define $=_S$, i.e. $=_S$ appears in the head, are disallowed in the program P.

The meaning of a definite logic program with an equational theory is given by the semantics of first-order logic: the atoms in the body of each clause are considered to be connected by conjunction and every clause is assumed to be universally closed. All clauses are connected by conjunction. Furthermore, the standard equational axioms and the axioms of the particular equational theory are added (by conjunction). Then, the semantics of the resulting formula where the considered structures are restricted to be Herbrand-interpretations define the meaning of the program. Using this semantics it has been shown in [72]:

Proposition 1.3.1 *Every definite* E*-program has a smallest Herbrand-*E*-model.*

In the following we will often consider the signature and set of variables associated with a definite E-program to be implicitly defined.

Definition 1.3.2 *Let $(D, \mathtt{E})$ be a definite* E*-program. Then a* definite query *is an expression of the form*

$$\leftarrow A_1, \ldots, A_m$$

where $A_1, \ldots, A_m$ are atoms with $m \geq 0$. $A_1, \ldots, A_m$ are called subgoals *of the query. The case where $m = 0$ is denoted by $\Box$. An* answer substitution *for a definite query G is a substitution σ with $Dom(\sigma) \subseteq Vars(G)$*[2]*. σ is called a* correct answer substitution *wrt $(D, \mathtt{E})$ if $D \cup E \models \forall.\,((A_1 \wedge \cdots \wedge A_m)\sigma)$.*

From Proposition 1.3.1 follows the correspondence between models and correct answer substitutions:

Corollary 1.3.2 *Let $(D, \mathtt{E})$ be a definite* E*-program. Then for the smallest Herbrand-*E*-model $\mathcal{M}$ of $(D, \mathtt{E})$ holds:*

$$\mathcal{M} = \{A \mid A \text{ ground atom, for which } \epsilon \text{ is correct answer wrt } (D, \mathtt{E})\}$$

In the special case where the equational theory E of a definite E-program $(D, \mathtt{E})$ is empty, we call $(D, \mathtt{E})$ a *definite logic program* and refer to it simply by D.

SLDE-resolution

In [72, 47, 66] *SLDE-resolution*[3] has been developed. SLDE-resolution extends the resolution calculus of [130] to deal with definite logic programs that are equipped with an equational theory.

Definition 1.3.3 *Let $(D, \mathtt{E})$ be a definite* E*-program. A* selection rule *R selects from every definite query $\leftarrow A_1, \ldots, A_m$ with $m \geq 0$ a particular subgoal A_k $(1 \leq k \leq m)$.*

An SLDE-resolution *wrt an* E*-program $(D, \mathtt{E})$ and a selection rule R leads from a definite query $G = \leftarrow A_1, \ldots, A_k, \ldots, A_m$ $(m \geq 1)$ to a definite query $G' = \leftarrow (A_1, \ldots, A_{k-1}, B_1, \ldots, B_l, A_{k+1}, \ldots, A_m)\theta$, called an* SLDE-resolvent, *if A_k is the subgoal chosen by R of G and there is a variant $A \leftarrow B_1, \ldots, B_l$ of some clause in D, such that θ is an* E*-unifier of A_k and A.*

After having defined a single resolution step we may also apply SLDE-resolution repeatedly until we arrive at the empty goal.

[2] Where $Vars(\leftarrow A_1, \ldots, A_m) = \bigcup_{i=1}^{m} Vars(A_i)$.

[3] SLDE is an abbreviation for "linear resolution with selection function on definite clauses with equality".

Definition 1.3.4 *Let $(D, \mathtt{E})$ be a definite* E*-Program. Let G_0 be a definite query. Then, a sequence $G_0, G_1, G_2, \ldots$ is called* SLDE-derivation *for $D \cup \{G_0\}$ wrt a selection rule R if for every $i = 1, 2, \ldots$, the query G_i is the result of the application of an SLDE-resolution to G_{i-1}. If such a derivation is finite and ends with $G_n = \square$ $(n \geq 0)$ it is called* SLDE-refutation *of length n for $D \cup \{G_0\}$ wrt R. If $\theta_1, \ldots, \theta_n$ is the sequence of* E*-unifiers used in the corresponding SLDE-resolutions then the substitution $(\theta_1 \ldots \theta_n)\mid_{\mathrm{Vars}(G_0)}$ is called* answer substitution *corresponding to the SLDE-refutation.*

In [47, 66] soundness and completeness of SLDE-resolution has been shown:

Proposition 1.3.3 *Let $(D, \mathtt{E})$ be a definite* E*-program, R a selection rule and G a definite query, then*

1. *Every answer substitution computed by $SLDE-resolution$ for $D \cup \{G\}$ wrt R is a correct answer substitution for G wrt $(D, \mathtt{E})$.*
2. *For every correct answer substitution θ for G wrt $(D, \mathtt{E})$ exists an answer substitution σ computed by SLDE-resolution for $P \cup \{G\}$ wrt R with $(\sigma \leq_S \theta)\mid_{\mathrm{Vars}(G)}$.*

The following data structure is a convenient way to represent SLDE-derivations.

Definition 1.3.5 (SLDE-tree) *Let $(D, \mathtt{E})$ be a definite* E*-Program and G a definite query, R a selection rule. An* SLDE-tree *for $(D, \mathtt{E})$, $\{G\}$ and R is a tree τ satisfying the following conditions:*

1. *each node of τ is a goal,*
2. *the root of τ is G,*
3. *Let $G' = \leftarrow A_1, \ldots, A_k, \ldots, A_m$ $(m \geq 1)$ be a node in τ and let A_k be selected by R. Then, each SLDE-resolvent of G' wrt A_k is a child of that node.*
4. *the empty clause does not have children.*

If there is a non-empty goal in τ where no atom in the body is selected τ is called incomplete.

Note that SLDE-trees are guaranteed to be finitely branching if the equational theory E is finitary.

Note also that *narrowing* [60] is an efficient approach to solve certain equational theories, and can be integrated as part of unification into SLDE-resolution.

For definite logic programs (E-programs where E is empty) we will write SLD-resultion, SLD-tree, SLD-resolvent, SLD-refutation, SLD-derivation to highlight the special case.

Part I

Knowledge Representation

Chapter 2

The Fluent Calculus

In the following section we discuss some of the assumptions on which knowledge representation in the Fluent Calculus is based. Thereby we will also relate it to other AI approaches which model action and change. In contrast to other work in the field we do not intend to provide a purely intuitive way of understanding the notions of fluents, states, situations, and actions. Instead, we attempted to relate them to other abstract concepts known from, e.g., modal logics. The reason for this is that our work focuses on the analysis of the computational properties of Fluent Calculus domains, not on the representational issues. For example, despite of the considerable intuitive difference between the concept fluent in the Situation Calculus and the concept fluent in the Fluent Calculus, we can show that the latter can be used to model the first (but not vice versa). We believe that from an abstract point of view such relations can be recognised easier.

2.1 Representation of Action and Change

To represent knowledge about the real world in a computer, we need some kind of formal language $\mathcal{L}$. We may understand the relation describing which sentences of $\mathcal{L}$ are conclusions of sentences of $\mathcal{L}$ as the semantics of $\mathcal{L}$. Note that the language $\mathcal{L}$ with such a semantics is a logic.

Which Language?

Informally, the Fluent Calculus like the Situation Calculus defines classes of logical second-order languages with extended first-order semantics. Every Situation Calculus language is based on the concepts *action*, *situation* and *fluent*. Fluent Calculus languages extend this foundation by the concept *state*.

The structure the world is regarded to have influences the choice of a particular language for knowledge representation. This choice is not unique – firstly, because humans have not yet succeeded in developing a single consistent theory about the structure of the world that explains all observed phenomena. Instead, several theories, which are represented in different languages and which require different ways of reasoning, are considered to be valid at the same time, even when they occasionally produce contradictory results. Secondly, different languages might be equally expressive, but the complexity of representations of particular kinds of knowledge and the complexity of the reasoning about these representations might vary. E.g., a language A might allow short descriptions of some property a, but require long descriptions of some other property b while a language B allows short descriptions of b but needs long descriptions of a.

To justify the choice of first-order logic for the original Situation Calculus it has been argued in [109] that the representation of the world as a system of interacting discrete finite automata is *metaphysically* adequate, i.e. propositions of interest in common sense reasoning do not contradict each other if the world is actually considered to be a system of interacting discrete finite automata. Thereby, a system of interacting discrete finite automata simply refers to a single discrete finite automaton where the set of states is split into disjoint subsets. Then, each subset S of states together with the set of transitions T leading to or from one of the states of S define a new automaton. The transitions of T which either lead to or originate from an element which is not in S are viewed as *interactions* with other automata of the system.

Finite automata are usually described by providing a complete state/transition table. For automata with many states these tables become very large and the expression of ascertainable knowledge in this way is inappropriate. Another difficulty arises when our knowledge about the world is incomplete. Consequently, in general, our knowledge describes a class of possible automata rather than a single one. To overcome these difficulties the use of first-order logic as representational language has been proposed. The major advantages of first-order logic are its clear model theory and its well-understood proof theory. Additionally, first-order logic is very expressive and most proposed alternative representation languages are no stronger, for an overview see e.g. [14].

Several problems of this approach result from using a system of interacting finite automata as a representation of the world: Finite automata allow only the representation of finite domains. But the boundaries of a considered domain might be unknown or it might be even impossible to explore them. In these cases, humans often assume the size of the domain to be infinite[1].

Fundamental concepts of finite automata are *states* and *transitions*. Thereby, transitions describe a relation between the states. If some part of the real world has to be represented by a finite automaton its states and transitions need to be

[1]Since [109] doesn't contain a formal language definition, it is unclear whether the exclusive use of finite domains was intended. In [107], however, the use of domain specific function symbols and the possibility of reification has been discussed.

identified by observation. This is sometimes impossible. For example, we may attempt to describe the singing of a bird. Although we may in principle assume that each sound uttered by the bird is determined by some particular state of the bird, listening to the bird is not sufficient to identify these states. Instead, we may use different concepts to describe our observation. In case of the singing bird we might choose concepts like *time* and *event*. However, time is often considered to be *continuous*. Consequently, if we want to express knowledge using concepts like time we need to extend our view of the world being a system of interacting finite automata.

Since first-order logic allows the definition of domain–specific recursive functions, representations in the Situation Calculus of [109] are not restricted to finite domains. To enforce an interpretation of a sort to contain only certain constructively defined elements, it has been proposed [127] to extend knowledge representations in the Situation Calculus by second-order induction axioms. Induction axioms permit reasoning about enumerable sets. In general, however, first-order logics with such extensions can be incomplete, i.e. there are consequences of the axioms which can not be proven within these logics [53].

To integrate knowledge which is expressed using different concepts from the ones originally defined in [109], the Situation Calculus has been augmented in several ways. Some of these extensions involve an additional sort which is interpreted in all models as the set of real numbers, e.g. [118, 143, 151]. This sort is used to describe *time* and *space* more accurately. Furthermore, a continuous time-line allows the representation of *events*, *time intervals* between events and other related notions. Based on the concepts of time and events another approach to represent action and change, called *Event Calculus*, has been developed [79]. Relations between the Situation Calculus and the Event Calculus have been discussed in detail, see e.g. [7]. Note however, since real numbers and operations on them are not axiomatised within the Event Calculus or the extensions of the Situation Calculus, respectively, reasoning about continuous domains is restricted.

Using the concepts of state and transition and using the concepts of event and time are based on different assumptions about the world: a description using the notions of states and transitions is effective if dependencies between subsequent states exist, a description using the notions of time and events is effective for systems where few such dependencies exist. However, sometimes it is possible to translate knowledge from one system of concepts into another. On one hand, if time is considered to be discrete, it may be represented in terms of states and transitions by adding a dimension to the state space. Thereby we may restrict the dimension representing time, e.g. to ensure monotonicity. In such a representation events can be understood as particular transitions that depend only on this new dimension[2]. On the other hand, in general states and transitions can not be represented in terms of time and events. This is due to the

[2]This is actually also the way, non–autonomous systems (i.e. systems which are described by difference equations that depend on time explicitly) are translated into autonomous systems (difference equations that do not depend on time explicitly).

assumed single time-line, i.e. a particular sequence of transitions represents one behaviour out of many *possible* ones while a sequence of events represents one *actual* behaviour of the world.

Concepts: States, Situations, Histories

Representations of knowledge in the Situation Calculus (and in the Fluent Calculus) are based on the notion of *situations*. In the original Situation Calculus a situation is considered to be a possible state or a "snapshot" of the world at a certain time. Later, it has been argued, e.g. in [62], that assuming separate systems always have a common state is unnecessarily complicated, particularly if the systems sparsely interact with each other. Moreover, two "snapshots" of the world might look the same although they have been made at different times. Another interpretation of a situation is the time between two actions where the world does not change [7, 117, 58]. In contrast to this, distinguishing "snapshots" or time intervals according to their order of appearance can often be useful. For these reasons, in current versions of the Situation Calculus every situation describes a particular *history* which is understood as a sequence of *actions* performed after an *initial situation*. In principle, different systems have different initial situations – corresponding to the system's first appearance – and, hence, different histories. An interaction of two or more systems is then understood as an "intersection" of their histories, i.e. a relation between situations of the interacting systems. This relation should induce a partial order on the possible histories of all the systems involved. For example, let s_1^0, s_1^1, s_1^2 and s_2^0, s_2^1, s_2^2 denote the subsequent situations of some system S_1 and S_2, respectively. Assume the relation describing the interactions of the two systems to be $\{(s_1^0, s_2^0), (s_1^1, s_2^2), (s_1^2, s_2^1)\}$. The induced partial order is $\{(s_1^0, s_2^0) < (s_1^1, s_2^2), (s_1^0, s_2^0) < (s_1^2, s_2^1)\}$. However, in this work we will only consider single systems without interactions.

In the Situation Calculus, states are associated with a particular situation using the notion of *fluents*[3]. In the Situation Calculus a fluent can be understood as a function f partitioning the set of states into two disjoint sets – the set where f maps to $\top$ (the set where property f *holds*), and the set where f maps to $\bot$ (the set where f does not hold), respectively. For example, the partitioning of the world states into those where the sky looks blue over London and those where the sky does not look blue might be represented by the fluent *blue_sky*. This notion of a fluent also corresponds to the definition of a *modal property*. Other partition functions can be defined in terms of fluents: e.g., the result of the conjunction of two fluents f_1, f_2 corresponds to the set of states resulting from intersecting the set where f_1 holds with the set where f_2 holds. The result of negating a fluent f is the set of states where f does not hold. Other operators for fluents can be defined accordingly.

[3]Note that in the original Situation Calculus fluents are called *propositional fluents* and the term describing the performing of an action is called *situational fluent*, [109].

Since in some versions (e.g., [109, 127]) of the Situation Calculus states are not objects of the language, the association of states to situations is rather implicit: fluents are represented as predicates where one parameter is of sort situation. In this case, first-order logic provides the boolean operations which allow the identification of particular state sets. Other versions (e.g., [98, 6, 99]) provide a dedicated sort used to represent fluents. The process of transferring predicates to the object level is called *reification*, [78, 107, 43]. The Fluent Calculus extends this idea by also representing state sets which result from combining fluents using certain operators, explicitly. The behaviour of these operators must be defined by additional axioms, which in case of the Fluent Calculus form an equational theory.

Note, however, that calculi for first-order logic are in general not sensitive to the specific structure of states if they appear as objects (nor to any class of objects which contain more structure than a general first-order term). Respecting this structure is often very important for adequate reasoning, e.g. if an equational theory has to be considered [121]. Correspondingly, standard calculi have been extended to deal with specific theories, e.g. for logic programming, SLD– and SLDNF–resolution ([130] and [19], respectively) have been extended to respect certain equational theories ([47, 72, 66] and [137, 146], respectively).

On the other hand, it is possible to integrate operators other than those provided by first-order logic directly into the logical language. This approach has led to the development of so called *non-classical* logics, which include, e.g., *sub-structural* logics[4] (introduced by [84, 51]), *modal* and *temporal* logics (introduced by [97] and [124], respectively and their semantics by [81, 82]). Some of these approaches have strong relations to representational and reasoning issues in the Situation Calculus and Fluent Calculus. Therefore, we will investigate and use these relationships to a great extent.

Action and Change

Another Assumption of the Situation Calculus and the Fluent Calculus is that great amounts of human knowledge about the world can be expressed as *causal* relationships between states, i.e. it is assumed that the fluents appearing in a particular state depend on temporally preceding states. Since situations are considered to be histories, time is represented implicitly by the partial ordering on histories: a situation s occurs later than another situation s' iff s contains s'. Moreover, the set of situations and the set of states associated with them is considered to be discrete: if a situation s' precedes a situation s, then there are only finitely many situations which occur after s' and before s.

Causal dependencies are represented by *actions* which represent functions mapping from situations to situations. Associated with actions are axioms describing the relationship between the states which correspond to the situations. The actual change of states according to a particular action is called *execution* of the

[4]See e.g. [134, 129] for introductions on sub-structural logics.

action. Note that, in the original Situation Calculus this relation between states depended also on a parameter describing the agent performing the particular action. Later this parameter has been omitted and current versions of the language do not contain an explicit representation of agents[5].

The most famous representational problem in the field of Reasoning about Action and Change was recognised in [109] and was christened the *(representational) frame problem*. It occurs if a state following the execution of an action can be characterised by similar properties as the state before the execution. In this case, instead of characterising the succeeding state by describing all of its properties, it is expected to be much easier to describe only those properties that do not *persist*. For example, if we want to describe the effects of lifting a cup from a table, we do not want to mention explicitly that the position of the table within the room is uneffected by the action. In the following, we will call such properties and fluents *persistent*, to emphasise the assumption to which they are subject. The axiomatisation which is required to support the specification of this kind of knowledge should be simple, i.e. small and easy to comprehend. Since the performance of a calculus decreases as more axioms have to be considered, the additional axiomatisation should be provided in a way that allows easy integration into a specialised calculus. To distinguish the latter requirement from the first one, which focuses on the representational aspect of the frame problem, it is also referred to as the *inferential frame problem* [13].

The search for a solution to the (representational and inferential) frame problem has led to the rise of a whole new research field called *non–monotonic reasoning*. It is based on the idea that frame axioms can be avoided if the entailment relation of the logic is extended, e.g. [108, 48, 19]. These extensions involve a violation of the *monotonicity property* of the entailment relation in first-order logic: adding knowledge to a database can prevent conclusions which are possible without it. Some of the proposed extensions have led to unexpected models, e.g. the Yale Shooting Problem [59] in the case of *circumscription* of [108][6].

To solve the representational frame problem further assumptions about the structure of knowledge have been made. In the Situation Calculus, persistent fluents are considered to be independent in the sense that the conjunction of any such fluents does not imply another persistent fluent to hold. This assumption enables for representation of states by sets of persistent fluents where the union operation on the sets corresponds to conjunction of the properties. Actions can be represented by functions which remove and add fluents from/to the set, an idea proposed in [38] for a system called *STRIPS*. If the independence is violated and the actions are not defined appropriately the succeeding state after executing an action can not be uniquely determined. E.g. let e, f, g be fluents which hold in some state and let $e \wedge f \Rightarrow g$ a relation between these fluents, then removing g from this state after executing some action implies

[5]For the problems of Reasoning about Action and Change considered in this work such a parameter could easily be added.

[6]A solution to this problem was given by using negation–as–failure for non-monotonic reasoning instead of circumscription [37].

that either e, or f, or e and f has/have to be removed as well to represent the succeeding state consistently. In [127], a procedure is given which transforms axioms describing the effects of the execution of actions into a set of axioms which entail the corresponding persistence assumptions if the persistent fluents are independent[7].

Note that to solve the representational frame problem persistent fluents are often assumed to be independent, even if they are actually dependent. In these cases the dependencies are represented as *indirect effects* of action executions. The representation of indirect effects is the subject of the *ramification problem* as described below.

Another way of dealing with the frame problem is based on the reification of persistent fluents. How these fluents can be combined to represent states must be defined by an additional theory. This can be achieved by providing an equational theory. By integrating this equational theory into the calculus, a computationally more adequate treatment of persistent fluents can be achieved, i.e. the inferential frame problem can be addressed as well. E.g., we may imagine a reified version of the solution to the frame problem given in [127] consisting of one additional sort representing states, together with additional operator symbols. The additional operator symbols must allow to add and remove fluents from a state representation. The effects of executing actions can then be specified in terms of equations. The Fluent Calculus is based on this approach. As we show in Section 4.3, the Fluent Calculus approach is in some sense more expressive than the Situation Calculus approach of [127] as it allows multiple occurences of fluents in state representations. An equational theory defines some operator to be the union of multisets. The multiple occurrence of a fluent rather then just a single fluent forms a modal proposition. Intuitively, the multiple occurrence of a fluent in a state representation corresponds to the multiple occurrence of a *resource* in this state. More precisely, we define a resource to be a set of modal properties with an associated total order. This ordered set is order isomorphic to the natural numbers. In the case of the Fluent Calculus the order is induced by the subset relation on multisets, e.g. for some fluent *orange* representing the availability of an orange, $\{orange, orange, orange\} \subset \{orange, orange, orange, orange\}$ implies that the set of states where $\{orange, orange, orange, orange\}$ holds contains also the set of states where $\{orange, orange, orange\}$ holds, but not vice versa. If a state representation in the Fluent Calculus contains precisely n occurrences of some fluent f then all modal properties where f occurs at most n times are considered to be valid. Whereas all modal properties where f occurs more than n times are considered to be not valid. For example, let $n = 3$ be the number of occurrences of the fluent *orange* in some state. Then the conjunction $\{orange, orange, orange\} \wedge \neg\{orange, orange, orange, orange\}$ is valid, where the modal properties are de-

[7]Reiter's notion of independence is given by the assumption that the following axiom is true: $\neg\exists\vec{x}, a, s. \gamma_F^+(\vec{x}, a, s) \wedge \gamma_F^-(\vec{x}, a, s)$. Here, γ_F^+ and γ_F^- describe the conditions for fluent F to hold and F not to hold in some situation $do(a, s)$, respectively. Both $\gamma_F^+(\vec{x}, a, s)$ and $\gamma_F^-(\vec{x}, a, s)$ may only refer to the values of fluents in situation s, not e.g. $do(a, s)$.

fined by the ordered set $\{\{\}, \{orange\}, \{orange, orange\}, \ldots\}$. Note however, that the solution to the frame problem in the Fluent Calculus relies on a similar assumption as the solution to the frame problem in the situation calculus: modal propositions formed by multisets of different fluents are considered to be logically independent.

Another important representational problem in Reasoning about Action and Change is the *ramification problem* [49]. It occurs if executing actions is considered to have effects which are not described explicitly. On one hand, certain persistent modal properties might be logically dependent although they are treated as independent by the frame axioms (as described above). On the other hand, additional causal dependencies may exist, that are not stated as actions. In [149] it has been proposed to solve this kind of problems in the Fluent Calculus by introducing a post-processing phase after each execution of an action. Clearly, the influence of this phase on the computational properties depends on the structure of the considered causal relations. In this work, we will not discuss the different effects of such dependencies in detail, and consider causal relations to be completely specified by axioms describing the direct effects of actions.

Similarly, the executability of an action might depend on modal properties which are not specified explicitly. We may consider such indirect dependencies because in non-artificial domains it is impossible to specify the preconditions for executing an action completely. For example, we may not be able to start the engine of our car because there is a potato in the tail pipe. However, we do not check such unlikely circumstances before attempting to execute an action unless there is additional evidence. The problem of representing such abnormal preconditions is called the *qualification problem* [106]. As shown in [148], disqualifying evidence for executing an action may be represented as a particular modal property which can be an indirect effect of executing other actions. Initially we may assume no disqualifying evidence, later the modal property representing such evidence might occur as an indirect effect during the post-processing phase used to solve the ramification problem.

Many other extensions have been proposed to integrate different kinds of knowledge about action and change into the Situation Calculus as well as the Fluent Calculus. These extensions include, for instance, in the case of the Fluent Calculus the representation of concurrent and simultaneous actions [15], the explicit representation of what an agent knows and how this knowledge can be improved by actions [152], non-deterministic actions [16], agent goals, rational actions, natural actions, continuous time and continuous change [151]. Many of these extensions may have a major impact on the computational properties of the reasoning method. We will briefly discuss the extensions by non-deterministic effects and the use of a specificity relation in Section 2.8.

What to infer?

In the previous subsections we have described on what concepts the Fluent Calculus is based and how these concepts are used to represent knowledge about action and change. The question of what should be derived from this knowledge has been left open so far. As a first choice it has been proposed, e.g. in [109] for the Situation Calculus, to consider knowledge which can be expressed in the same language as the language used to represent action and change. However, the form of knowledge that intelligent systems may directly acquire by observing the world is different from the form of knowledge they need to control their behaviour. This follows from the assumption that intelligent systems, like humans, actually perform deductive tasks. Furthermore, if the form of knowledge about action and change differs from the form of knowledge used for control we may also represent them using different languages which are optimised for their specific applications. We hope that for such a pair of optimised languages the development of powerful calculi is easier than for the more general language containing both.

In the following we call the language for representing the knowledge which is used for control *query language*. It has been pointed out already in [114] that first-order languages are insufficient to represent some important temporal properties of dynamical systems (represented as programs in [114]). There it has been proposed to extend the language with (second-order) *fixpoint operators*. In the context of the Situation Calculus, a similarly extended query language was introduced in [119] and [144]. In this work, we follow this approach in general. But, since we are interested in achieving lower bounds for undecidability and upper bounds for decidability, respectively, we will distinguish more restricted fragments of the general language. In particular, we define fragments which can be characterised by well known modal/temporal logics. The chosen logics are good candidates, since they have been used successfully for similar tasks – the verification of dynamic systems. For such logics we define the most general problem of reasoning about a Fluent Calculus domain $\mathcal{D}$:

Entailment Problem: Let ϕ be an arbitrary formula of the query logic. Is ϕ a logical consequence of $\mathcal{D}$?

We can hope to be able to decide the entailment problem for weak query languages and weak Fluent Calculus fragments, only. For powerful Fluent Calculus fragments the investigation has to be restricted further. To this end we will focus on a particular class of problems which can be formulated in all query languages considered here:

Planning Problem: Assume, an agent knows certain properties (λ_0) of its current situation and it attempts to reach a situation with certain goal properties (λ_e) by executing certain actions. Then, answering the following question is crucial: "Is there an initial situation with property λ_0 and is there a sequence of actions leading from this situation to some situation with property λ_e?". If the answer to this question is "no" the agent must give up pursuing its goal. If the answer is "yes" and sufficient information about the current situation is available the agent might proceed by computing an appropriate action sequence to reach its goals[8]. If the answer is "yes" but the current situation is not completely known the agent may try to gain more information about its current situation, or execute an action sequence that succeeds in some situations with the given properties, or investigate the *extended planning problem* described below.

In AI research the planning problem has been considered to be one of the most important problems of Reasoning about Action and Change [55]. In fact the *Planning* has established itself as a whole research field focusing on the design of efficient planning systems for domains of increasing complexity, e.g. [75]. Compared with the deductive approach to planning we follow here, many planning algorithms proposed in this way have insufficiently defined semantics. Therefore, despite often being technically less sophisticated[9], the development of deductive planning techniques may help with understanding what planning systems actually compute. Furthermore, in contrast to deductive planning systems, non-deductive planning systems usually assume the initial state to be completely known.

Since solving planning problems is still a very general task, we will concentrate on a more restricted version [18] in the third part of this work, where we develop an automatic reasoning method:

Conjunctive Planning Problem (CPP): Assume λ_0 to be a conjunction of negative or positive modal properties and λ_e a conjunction of positive modal properties. Is there a finite sequence of actions which leads from a situation where λ_0 holds to a situation where λ_e holds?

Another version of the planning problem results from the assumption that the properties of the current situation might not be completely known:

Extended Planning Problem: Assume, that an agent has incomplete knowledge (λ_0) of its current situation and it has to verify whether for *each* situation where property λ_0 exists an action sequence such that the agent reaches its goal λ_e. If this is the case the agent can be sure it is able to reach its goal by an appropriate action sequence. However, due to the uncertainty about the current situation there might be many potential but few successful sequences to choose from[10].

Although in this work we focus on the above versions of the planning problem, many other problems have been considered to be particularly important if we aim to build intelligent systems. Some of them are characterised in the following list:

[8]By applying deductive systems to solve planning problems one usually hopes to find an appropriate action sequence as a side effect. And in some cases (e.g. for $\mathcal{FC}_{PL}$, see Section 2.5 and 5.3) such a sequence is indeed already computed by the considered decision procedure.

[9]E.g., most deductive planning systems generate only *linear* plans, while non-deductive ones can also generate non-linear plans, e.g. [156].

[10]We may also think of an even more extended planning problem: Is there a single plan which can be successfully applied in all situations where λ_0 holds?

Prediction Problem: Given the properties of the initial situation and a finite sequence of actions, which properties hold in the situation after executing this sequence?

Postdiction Problem: Given some properties of some final situation and some properties of situations which occurred while executing some known action sequence, which properties must have been associated with the initial situation?

General Explanation Problem: Like the postdiction problem, but assume the action sequence to be unknown.

In the next section we present our Fluent Calculus framework formally.

2.2 Signature and Domain Independent Axioms

Originally, in [68] the Fluent Calculus was represented as a scheme for specifying dynamic systems as logic programs together with SLDE-resolution as execution mechanism. To allow easier integration of extensions into a single approach and to be able to compare it with the standard approach of the Situation Calculus, a more general execution independent representation has been developed [150]. In this work, we will identify the Fluent Calculus ($\mathcal{FC}$) with this latter representation. Accordingly, the Fluent Calculus is – similar to the Situation Calculus of [117] – a scheme of specifying domains using second order languages.

Definition 2.2.1 *A signature is called a* Fluent Calculus signature *iff it is a tuple* $(SORT, \preceq, FUN, REL)$ *where*

$SORT$: Act, Sit, Fl, St, Obj;
$\preceq$: $Fl \preceq St$;
FUN: $s0 :\rightarrow Sit$;
$do : Act \times Sit \rightarrow Sit$;
$state : Sit \rightarrow St$;
$1^\circ :\rightarrow St$;
$\circ : St \times St \rightarrow St$;
$f_1 : Obj^{i_1} \rightarrow Fl, \dots, f_j : Obj^{i_j} \rightarrow Fl$; ($j \in \mathbb{N}^+$ and $i_1, \dots, i_j \in \mathbb{N}$)
$a_1 : Obj^{k_1} \rightarrow Act, \dots, a_l : Obj^{k_l} \rightarrow Act$; ($l \in \mathbb{N}^+$ and $k_1, \dots, k_l \in \mathbb{N}$)
$g_1 : Obj^{m_1} \rightarrow Obj, \dots, g_o : Obj^{m_o} \rightarrow Obj$; ($o, m_1, \dots, m_o \in \mathbb{N}$)
REL: $<: Sit \times Sit$; $Poss : Act \times Sit$; $\{=_S: S \times S \mid S \in SORT\}$

The sort *Sit* is used to represent the set of *situations*. A situation is a history of action executions starting in some *initial situation*. This is reflected by the structure of terms of the sort *Sit*: Terms of sort *Sit* are recursively defined by applying the function *do* to a term of sort *Act* representing a set of *actions* and a term of sort *Sit*. The initial situation is represented by $s0$. Note that *Sit*, *Act*, *do* and the initial situation $s0$ are also part of the Situation Calculus.

By the function *state* a *state* of the world is associated with every situation. A state may only change if actions are executed. Formally, the function *state* maps to the sort *St* since in $\mathcal{FC}$ domains world states are represented as particular terms. In contrast to this (see also the previous section), in the Situation Calculus the state associated with a situation is implicitly represented by the set of predicates that are valid in this situation. Basic elements of state representations are called *fluents*. Fluents are elements of the sort *Fl*. The properties of the function $\circ : St \times St \to St$ determine how fluents are combined to form state representations. Thereby, the term 1° represents an *empty* state. The introduction of an explicit state representation enables simpler axiomatisation of properties shared by all fluents in all states. For instance, assume that a fluent should not appear or disappear unless it is a direct consequence of executing some action (i.e. the *frame assumption*). This can be easily expressed using an equation describing the dependency between the state after executing an action and the state before executing this action. In particular, this equation is independent of the fluents that are not effected. E.g., the equation $state(s) \circ orange\_juice =_{St} state(do(squash, s)) \circ orange$ can be used to describe that the state associated with the situation after executing the action *squash* contains one more fluent *orange_juice* and one fluent *orange* less than the situation s. In contrast to this, an axiomatisation of the frame assumption in the Situation Calculus requires axioms for each individual fluent. If the number of fluents is large compared to the number of actions in a domain the use of explicit state representations may decrease the size of the axiomatisation dramatically.

The domain independent predicate symbols contain $<$ which is intended to represent the *temporal* order of situations, i.e. the order given by the subsequence relation. *Poss* is a common auxiliary predicate describing the executability of actions in situations, i.e. whether executing an action in some situation leads to a valid successor situation.

The functions $a_1, \ldots, a_l$, which map to sort *Act*, define domain specific actions and the functions $f_1, \ldots, f_j$, which map to *Fl*, define domain specific fluents. With the help of additional elements of sort *Obj* and functions over these elements, the fluents themselves can be structured additionally. To this end we may define arbitrary functions $g_1, \ldots, g_o$, which map to *Obj*. Then objects of sort *Obj* can be used as parameters of fluents. Note that according to the Fluent Calculus signature scheme, functions with domain *Obj* may only depend on elements of *Obj*. The meaning of these elements is domain specific. Similarly, actions may depend on additional domain specific parameters and thereby supply situations with additional structure beyond the properties of the function *do*.

Example 2.2.1 (Brick domain) *We define Σ_b as a Fluent Calculus signature with the domain specific sorts XPos, YPos $\subset$ Obj and the domain specific functions*

$\mathit{brick\_at} : \mathit{XPos} \times \mathit{YPos} \rightarrow \mathit{Fl};$
$\mathit{bar\_at} : \mathit{XPos} \times \mathit{YPos} \times \mathit{XPos} \times \mathit{YPos} \rightarrow \mathit{Fl};$
$\mathit{mv\_brick} : \mathit{XPos} \times \mathit{YPos} \times \mathit{XPos} \times \mathit{YPos} \rightarrow \mathit{Act}.$

XPos and YPos are intended to represent coordinates in a two-dimensional grid. Correspondingly, the fluent brick_at $\in$ *Fl represents the existence of a brick at a certain location, the fluent* bar_at $\in$ *Fl represents the existence of a bar with a pair of coordinates describing start and end positions. The action* mv_brick $\in$ *Act represents the moving of some brick from one location to another. Then, for example, the situation after moving a brick from position location* $(1,1)$ *to location* $(2,2)$ *is represented by the term* $do(\mathit{mv\_brick}(1,1,2,2), s0)$. *The state associated with this situation is represented by* $state(do(\mathit{mv\_brick}(1,1,2,2), s0))$. *By domain specific axioms we will ensure later that such a state can be identified with a term like* $\mathit{brick\_at}(2,2) \circ \mathit{bar\_at}(1,1,3,3)$. □

Example 2.2.2 (Airport) *Consider a very simple bay management and take-off model of an airport. A number of small planes and a number of big planes may be queuing and requesting the permission to land. The fluents* plane_q_s *(*plane_q_b*, respectively) represent the queuing small (big) planes. The arrival of a new small (big) plane in the queue is represented by the action* queue_s *(*queue_b*, respectively). The bay capacity of the airport is modelled by the fluent* bay. *Landing is modelled by the actions* land_b *and* land_s *for big and small planes, respectively. After granting the permission the appropriate space on the airport must be reserved for the landed plane, modelled by the fluents* plane_l_b *and* plane_l_s *for each class of plane. Passengers leaving the airplane are modelled by the fluent* passenger. *The airplane takeoff is described by the actions* take_off_b *(big plane) and* take_off_s *(small plane). The runway is modelled by the fluent* runway. *We define* Σ_a *as a Fluent Calculus signature with the following domain specific functions:*

$\mathit{plane\_q\_s} :\rightarrow \mathit{Fl},\ \mathit{plane\_q\_b} :\rightarrow \mathit{Fl},\ \mathit{plane\_l\_s} :\rightarrow \mathit{Fl},\ \mathit{plane\_l\_b} :\rightarrow \mathit{Fl},$
$\mathit{passenger} :\rightarrow \mathit{Fl},\ \mathit{runway} :\rightarrow \mathit{Fl},\ \mathit{bay} :\rightarrow \mathit{Fl};$
$\mathit{queue\_s} :\rightarrow \mathit{Act},\ \mathit{queue\_b} :\rightarrow \mathit{Act},\ \mathit{land\_b} :\rightarrow \mathit{Act},\ \mathit{land\_s} :\rightarrow \mathit{Act},$
$\mathit{take\_off\_b} :\rightarrow \mathit{Act},\ \mathit{take\_off\_s} :\rightarrow \mathit{Act}.$

□

To simplify the remaining presentation, we define terms of sort *St* that contain multiple copies of one particular fluent term, only. E.g., assume that the sets *XPos* and *YPos* in Example 2.2.1 are both interpreted as the natural numbers. Then the term $\mathit{brick\_at}(2,4) \circ (\mathit{brick\_at}(2,4) \circ \mathit{brick\_at}(2,4))$ contains only copies of $\mathit{brick\_at}(2,4)$.

Definition 2.2.2 *We denote a term of the form* $f \circ (\cdots (f \circ f) \cdots)$ *containing* n *copies of* f *as* f^n. *We define* $f^0 = 1^\circ$.

Furthermore, let $|g, f|$ *denote the number of fluents* f *which occur in a term* g *of sort St.*

As it will become clear later, a term g is completely defined up to equivalence wrt the equational theory AC1 (see below) if, for all ground fluents f, $|g, f|$ is given.

Domain Independent Axioms

We define a *domain description* $\mathcal{D}$ for a $\mathcal{FC}$ signature Σ to consist of a certain set of axioms. Some of these axioms are considered to be valid in all domain descriptions for $\mathcal{FC}$ signatures, they are called *domain independent* and introduced in the following. The *domain dependent* axioms are defined as instantiations of a scheme which is presented in Section 2.5 and 2.8. The instantiation depends on the particular considered domain.

The Fluent Calculus relies heavily on the use of equations, e.g. to describe dependencies between state representations. To ensure that a predicate $=_S$: $S \times S$ with $S \in \mathit{SORT}$ represents indeed an equational relation every domain description must contain the standard equality axioms (see Definition 1.1.12) for S. We denote the set of standard equality axioms axioms for all sorts $S \in \mathit{SORT}$ by $\mathtt{E}^=$.

The common basis of all $\mathcal{FC}$ domains, i.e. the way state representations can be combined to form new state representations, is the definition of $\circ$ as a commutative monoid by the set of axioms AC1. Hence, the operator $\circ$ can also be thought of as the multiset union. This enables the solution of the frame problem, since adding and removing of fluents, respectively, can be expressed without explicitly stating which fluents are *not* effected. Furthermore, the corresponding equational theory, i.e. the theory given by the axioms of AC1 together with $\mathtt{E}^=$, can be integrated in the unification procedure.

$$\begin{aligned}
&\forall(x, y, z : \mathit{St}).\, (x \circ y) \circ z =_{St} x \circ (y \circ z),\\
&\forall(x, y : \mathit{St}).\, x \circ y =_{St} y \circ x, \qquad\qquad (\mathtt{AC1})\\
&\forall(x : \mathit{St}).\, x \circ 1^\circ =_{St} x.
\end{aligned}$$

The following proposition (shown e.g. in [100]) is particularly important if we wish to reason about Fluent Calculus domains using standard calculi like resolution:

Proposition 2.2.1 *The equational theory* AC1 *is finitary.*

Each domain description contains axioms defining a causal order over situations. For situations s_1, s_2, $s_1 < s_2$ is intended to be true iff 1) the action sequence leading from the initial situation $s0$ to s_1 is also prefix of the action sequence from $s0$ to s_2, and 2) every action of the sequence is actually executable in the corresponding situation (described by predicate *Poss*). In other words, s_1 is part of the history of s_2.

Using the shortcut $s_1 \leq s_2 \overset{\text{def}}{=} (s_1 < s_2 \vee s_1 =_{Sit} s_2)$ we define:

$$\forall(s_1, s_2 : Sit), (a : Act).$$
$$(s_1 < do(a, s_2) \Leftrightarrow s_1 \leq s_2 \wedge s0 \leq s_1 \wedge Poss(a, s_2)), \qquad (\mathtt{O}_{Sit})$$
$$\forall(s_1, s_2 : Sit). (s_1 < s_2 \Rightarrow \neg s_2 < s_1).$$

On one hand, the axioms ($\mathtt{O}_{Sit}$) do not characterise the desired property sufficiently, since they do not guarantee that different action names represent indeed different actions (i.e. *unique name assumption*). On the other hand, models may contain situations which can not be reached from $s0$ by finite action sequences. Similarly, a fluent may have several names and infinite numbers of fluents may appear in elements of the interpretation of sort St, without violating ($\mathtt{AC1}$) and the standard equality axioms. Additionally, it is often required to derive also inequality of objects, [19, 72, 70].

2.3 Herbrand-$\mathrm{E}_{\mathcal{FC}}$-Models

One way of ensuring both unique name assumptions for all sorts $S \in SORT$ and the possibility to derive inequality (see also the paragraph on unification completeness in Section 1.1), has been proposed in [70]. The approach relies on an set of axioms, called *extended unique name assumptions (EUNA)*. These axioms play a similar role to the *completion semantics* for logic programs of [19] (the corresponding proof procedure is often also called *negation as failure*). There, the *failure* to derive some atom p from some formula ϕ is considered to be a proof for $\neg p$. Correspondingly, the extended unique name assumptions ensure that whenever two terms can not be unified they are considered to be not equal. Since the equational theory $\mathtt{AC1}$ has to be taken into account, a finite axiomatisation of inequality in the style of [19] for the standard equality theory can not be achieved. Hence, a calculus for the Fluent Calculus can only implicitly deal with the axioms of EUNA.

Definition 2.3.1 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a $\mathcal{FC}$ signature and* E *a set of equational theories. For every $\mathrm{E}_S \in \mathrm{E}$ with predicate $=_S \in REL$ and $S \in SORT$ we define the set $\mathrm{E}^*_S = \mathrm{E}_S \cup \mathrm{E}^c_S$ where E^c_S is given as the set of formulas which contains for all $s, t \in T_{S,\Sigma}(X)$*

1. *either $\neg\exists. (s =_S t)$ if s and t are not E_S-unifiable*
2. *or $\forall. (s =_S t \Rightarrow \bigvee_{\theta \in U_{\mathrm{E}_S}(s,t)} \exists \bar{Z}_\theta. (\theta_{=_S}))$*

where $Vars(s) \cup Vars(t) = \{y_1, \ldots, y_n\}$, $\theta_{=_S} = y_1 =_{S_1} r_1 \wedge \cdots \wedge y_m =_{S_m} r_m$ with $z_i\theta = r_i$ for all $i = 1, \ldots, m$ and $\bar{Z}_\theta$ is a sequence of variables $z_1, \ldots, z_k$ with $Vars(r_1) \cup \cdots \cup Vars(r_k) = \{z_1, \ldots, z_k\}$. $U_{\mathrm{E}_S}(s,t)$ denotes a complete set of E_S-unifiers of s and t.

The axioms $EUNA = \bigcup_{S \in SORT} \mathrm{E}^*_S$ are called extended unique name assumptions *wrt the $\mathcal{FC}$ signature Σ.*

Since complete unification algorithms for the equational theory AC1 exist, e.g. [17], the axioms *EUNA* ensure unification completeness according to Definition 1.1.20, i.e. *EUNA* defines inequality of terms in the sense that every Herbrand-model of *EUNA* induces the smallest congruence relation on the Herbrand-universe wrt E. Hence, *EUNA* ensures uniqueness of names if Herbrand-models are considered. Furthermore, in Herbrand-models the interpretations of sorts are inductively closed, e.g., the interpretation of *Sit* contains only elements definable by finite application of *do* on *s*0. However, since the interpretation of *St* should consist only of terms that can be built by (finite) application of $\circ$ on fluents and 1°, we must not allow state terms to be constructed using the function *state*. Instead, the function *state* is defined by domain specific axioms, as described in Section 2.4, and interpreted in the standard way of sorted first-order logic.

Definition 2.3.2 *Let Σ be a Fluent Calculus signature and Σ^- denote the signature Σ without the definition of function state. An interpretation I of Σ is called* Herbrand-$\mathrm{E}_{\mathcal{FC}}$-interpretation *of Σ if I is a Herbrand-*E*-interpretation of Σ^-. A Herbrand-$\mathrm{E}_{\mathcal{FC}}$-interpretation I of Σ is called* Herbrand-$\mathrm{E}_{\mathcal{FC}}$-model *if I is an* E*-model.*

In the remainder of this work we will mean by *model* of a Fluent Calculus domain a model that is also a Herbrand-$\mathrm{E}_{\mathcal{FC}}$-model. However note that in E-programs only pure Herbrand-interpretations can be considered. Therefore, if a Fluent Calculus domain has to be represented as an E-program, the domain specific function *state* has to be represented by a relation. See Section 6.2 where this is discussed in more detail.

Instead of considering Herbrand-$\mathrm{E}_{\mathcal{FC}}$-models, unique name assumptions may be provided explicitly, depending on the considered domain. For example, if we want to specify that the fluent terms f_1 and f_2 must represent different objects, we simply write $f_1 \neq_{St} f_2$. To ensure inductive closure of interpretations of sort *Sit* it has been proposed in [128] to use a second order induction axiom. Similarly, instead of using the rigorous approach of unification completeness, the interpretation of sort *St* may be restricted using the axiomatisation of [142]. Again, the inductive closure of interpretations of sort *St* may be defined explicitly by a second order induction axiom. In [142] the new axiomatisation is called $EUNA^+$. An interpretation of the sorts *Sit*, *St* and of relations $=_{St}$, $=_{Sit}$, which fulfills $EUNA^+$, the induction axiom of [128] and unique name assumptions and domain closure axioms[11] for elements of *Act* and *Fl* is isomorphic to their interpretation in any Herbrand-$\mathrm{E}_{\mathcal{FC}}$-model of the domain description fulfilling *EUNA*. Consequently, the results presented here can be applied also if we use this alternative axiomatisation.

[11]These are axioms ensuring that sorts are interpreted only using explicitly named elements.

2.4 Domain Dependent Axioms

The domain dependent part of a Fluent Calculus domain description consists of *state update axioms*, which describe when and how a state associated with some situation depends on the state associated with the preceding situation. Formally, state update axioms define the function *state*. In this work we will investigate only domain descriptions where the state update axioms fit into the following scheme:

Definition 2.4.1 *Let $\mathcal{D}$ be a domain description for some $\mathcal{FC}$ signature $\Sigma = (SORT, \preceq, FUN, REL)$ and X a variable declaration wrt Σ. Let $(a : Obj^k \rightarrow Act) \in FUN$ and $Y \subseteq X$, $\bar{Y} = Y_1, \ldots, Y_k$ a sequence of the variable names of Y with $(Y_i : Obj) \in Y$ for $i = 1, \ldots, k$. Let $St_a^{-,j}, St_a^{+,j} \in T_{St,\Sigma}(Y)$ for $j = 1, \ldots, m$. A formula in $\mathcal{D}$ is a* state update axiom *iff it has the form:*

$$\begin{array}{l}\forall \bar{Y}. (\forall (s : Sit). (Poss(a(\bar{Y}), s) \Rightarrow \\ \qquad \bigvee_{j=1,\ldots,m} state(do(a(\bar{Y}), s), s) \circ St_a^{-,j} =_{St} state(s) \circ St_a^{+,j}))\end{array} \qquad (\mathtt{SUA}_{\mathcal{D}}^{a})$$

By $\mathtt{SUA}_{\mathcal{D}}$ we denote the set of all state update axioms in $\mathcal{D}$.

Informally, a state update axiom describes the relation between the state associated with s and the state associated with $do(a(\bar{Y}), s)$ if it is possible to execute an instance of action a in situation s. The relation is defined by a disjunction of state equations where for the ith equation $St_a^{-,i}$ represents the fluents to be removed from the state associated with s and $St_a^{+,i}$ represents the fluents to be added. The form of these state equations is crucial for the solution to the frame problem in the Fluent Calculus: only those fluents which are directly effected by executing action a must appear in the state equation. Note that the executability of action a in situation s is determined by the validity of $Poss(a(\bar{Y}), s)$. Thereby, $Poss(a(\bar{Y}), s)$ is defined by so called *precondition axioms*. Precondition axioms are domain dependent and their structure varies with the considered Fluent Calculus fragment. In the following section we will define the Fluent Calculus fragments and the corresponding structure of the precondition axioms we focus on in this work.

In general one might wish to represent also "non-frame" fluents, e.g. fluents whose appearances and disappearances are not directly related to the effects of some action. Of course, computational properties of domain specifications containing non-frame fluents depend on their particular relations to effects of actions and to situations, respectively. So far there is no systematic approach for treating non-frame fluents in $\mathcal{FC}$[12], consequently we focus here on persistent fluents only.

[12] With the exception of fluents that change as *indirect effects* of actions to solve the (representational) *ramification problem*. However, state equations describing these effects are usually considered to be of the above form [147].

2.5 Deterministic Domain Descriptions

In most of the following we consider only domain descriptions where every state update axioms contains precisely one state equation:

$$\begin{aligned}&\forall \bar{Y}.\,(\forall (s : Sit).\,(Poss(a(\bar{Y}), s) \Rightarrow \\ &\qquad state(do(a(\bar{Y}), s), s) \circ St_a^- =_{St} state(s) \circ St_a^+))\end{aligned} \tag{2.1}$$

where St_a^-, St_a^+, a, $\bar{Y}$ are defined as in Definition 2.4.1 for some action $a \in Act$.

Since a formula of the form in Equation 2.1 determines precisely the effects of an action, we call the state update axioms and the corresponding domain descriptions *deterministic*. Deterministic state update axioms are a special case of Definition 2.4.1. We call the state update axioms of Definition 2.4.1 and the corresponding domain descriptions *non-deterministic*. In Section 2.8 we will argue why we focus in this thesis on the deterministic case.

To simplify the statement of modal properties, which will also be required to define precondition axioms in all Fluent Calculus fragments, we define the short-cut

$$Holds(g, s) \overset{\text{def}}{=} \exists (z : St).\, g \circ z =_{St} state(s) \tag{2.2}$$

if $g \in T_{St,\Sigma}(X)$ and $s \in T_{Sit,\Sigma}(X)$. Informally, $Holds(g, s)$ is true iff at least all fluents of g are available in the state associated with situation s. E.g. considering the brick domain of Example 2.2.1, $Holds(bar\_at(2,4) \circ bar\_at(3,4), s)$ is valid if $state(s) = bar\_at(2,4) \circ bar\_at(2,4) \circ bar\_at(3,4)$, but is not valid if $state(s) = bar\_at(2,4) \circ bar\_at(2,4)$. In other words, since $\circ$ describes the multiset union, $Holds(g, s)$ is true iff g is a sub-multiset of $state(s)$.

The fragments we will define in this work are strongly related to schemes that have been proposed to solve certain representational problems in Reasoning about Action and Change. To allow systematic investigation of the implied computational properties we will simplify and generalise the proposed schemes. Despite of these differences all results of this work can be applied equally to the corresponding original schemes.

In the following we represent Fluent Calculus fragments in two groups which we call *propositional* and *non-propositional*, respectively. The distinction is a result of disallowing the use of domain specific sorts (i.e. subsets of Obj) in propositional fragments. Consequently, we may expect propositional fragments to be generally less expressive than their non-propositional counterparts. Furthermore, due to the great expressive power of non-propositional fragments, an investigation of the propositional fragments might be more interesting, since we aim to find boundaries of what can be achieved by automatic reasoning methods.

Non-propositional Fragments

$\mathcal{FC}_L$ and the simple Fluent Calculus

The Fluent Calculus was originally introduced in [68] as an equational logic programming scheme, which enabled for reasoning using SLDE-resolution. Since then it has been extended in several ways; we will refer to this original scheme as *simple Fluent Calculus.*

The fragment $\mathcal{FC}_L$ defined below is – compared to the simple Fluent Calculus – more independent of the applied reasoning method. However, as in simple Fluent Calculus domains, it is not possible in $\mathcal{FC}_L$ to use negation to describe action preconditions. In case of the simple $\mathcal{FC}$ this limitation is a result of applying SLDE-resolution as reasoning method, which can handle definite logic programs, only.

Precondition axioms of $\mathcal{FC}_L$ are slightly more general than in the simple Fluent Calculus, as they may contain a conjunction of *Holds*-predicates rather than only a single one. Note however that the simple $\mathcal{FC}$ could be easily extended to cope with such conjunctions (without modification of the calculus). Furthermore, in Section 5.5 this generalisation will prove to be not essential. The result achieved there does not require the use of conjunctions in precondition axioms. Hence it will be applicable to the simple Fluent Calculus as well.

Definition 2.5.1 *Given a domain description $\mathcal{D}$ for some $\mathcal{FC}$ signature $\Sigma = (SORT, \preceq, FUN, REL)$ and X a variable declaration wrt Σ. $\mathcal{D}$ is called* positive linear *iff for every action $(a : Obj^k \to Act) \in FUN$ there is a state update axiom* $\mathrm{SUA}^a_{\mathcal{D}}$ *of the form:*

$$\forall \bar{Y}. (\forall (s : Sit). (Poss(a(\bar{Y}), s) \Rightarrow \\ state(do(a(\bar{Y}), s), s) \circ St^-_a =_{St} state(s) \circ St^+_a))$$

and there is an axiom of the form

$$\forall \bar{Y}. (\forall (s : Sit). (Poss(a(\bar{Y}), s) \Leftrightarrow Holds(St^-_a, s) \wedge \textstyle\bigwedge_{i=1,\ldots,n_a} Holds(St^=_{a,i}, s))$$

Thereby $Y \subseteq X$, $\bar{Y} = Y_1, \ldots, Y_k$ is a sequence of the variable names of Y with $(Y_i : Obj) \in Y$ for $i = 1, \ldots, k$. $St^-_a, St^+_a, St^=_{a,1}, \ldots, St^=_{a,n} \in T_{St,\Sigma}(Y)$.

The set of all positive linear $\mathcal{FC}$ domain descriptions with appropriate signatures is denoted by $\mathcal{FC}_L$.

Another difference between $\mathcal{FC}_L$ and the simple Fluent Calculus is that according to the latter it is possible to specify more than one clause describing different preconditions and effects of the same action. Thereby, according to the semantics of logic programs, several clauses may be alternatively applied to describe the effects of some action. In $\mathcal{FC}_L$ such domain descriptions can not be represented directly since to this end disjunction in the precondition axioms is required in the Definition 2.5.1. However, such domains may be transformed into $\mathcal{FC}_L$ domains (see Section 2.8 for some details).

The following example is intended to give an intuition of how domains can be specified in $\mathcal{FC}_L$ using precondition axioms and state update axioms.

Example 2.5.1 (Brick domain continued) *Consider the signature Σ_b of Example 2.2.1. Furthermore, let $sx : XPos \to XPos$, $x0 :\to XPos$, $sy : YPos \to YPos$, $y0 :\to YPos$ be part of the signature describing some domain specific objects. Let the action precondition axioms be defined as*

$$\begin{aligned}&\forall (x_1, x_2 : XPos), (y_1, y_2 : YPos), (s : Sit).\\&\qquad (Poss(mv\_brick(x_1, y_1, x_2, y_2), s) \Leftrightarrow Holds(brick\_at(x_1, y_1), s)).\end{aligned}$$

Furthermore, let the state update axiom for mv_brick be defined such that the number of bricks at location (x_1, y_1) is decreased and the number of bricks at location (x_2, y_2) is increased, respectively, if mv_brick is executed:

$$\begin{aligned}&\forall (x_1, x_2 : XPos), (y_1, y_2 : YPos), (s : Sit).\\&\qquad (\,Poss(mv\_brick(x_1, y_1, x_2, y_2), s) \Rightarrow\\&\qquad\ \ state(do(mv\_brick(x_1, y_1, x_2, y_2), s)) \circ brick\_at(x_1, y_1)\\&\qquad\ \ =_{St} state(s) \circ brick\_at(x_2, y_2)\,)\end{aligned}$$

The resulting domain description is clearly a $\mathcal{FC}_L$ domain description. □

In the next example we introduce an abstract domain. It will allow us to simplify the presentation of some technical issues throughout this work.

Example 2.5.2 (Abstract $\mathcal{FC}_L$ domain) *Consider a $\mathcal{FC}_L$ signature Σ_l containing the fluents $F_{\Sigma_l} = \{f1 :\to Fl, f2 : Ob \to Fl, f3 : Ob \to Fl, f4 :\to Fl, f5 :\to Fl\}$ and the actions $A_{\Sigma_l} = \{a1 :\to Act, a2 \to Act, a3 : Ob \to Act, a4 : Ob \to Act, a5 :\to Act, a6 : Act\}$. Thereby, the domain specific sort Ob is defined by $0 :\to Ob$ and $foo : Ob \to Ob$.*

Let

$$\begin{aligned}&\forall (s : Sit).\,(Poss(a1, s) \Leftrightarrow Holds(f1, s))\\&\forall (s : Sit).\,(Poss(a2, s) \Leftrightarrow Holds(f1, s))\\&\forall (y : Ob), (s : Sit).\,(Poss(a3(y), s) \Leftrightarrow Holds(f2, s))\\&\forall (y : Ob), (s : Sit).\,(Poss(a4(y), s) \Leftrightarrow Holds(f3(y), s))\\&\forall (s : Sit).\,(Poss(a5, s) \Leftrightarrow Holds(f4, s))\\&\forall (s : Sit).\,(Poss(a6, s) \Leftrightarrow Holds(f5, s))\end{aligned}$$

be the precondition axioms and

$$\begin{aligned}
&\forall(s : Sit).\,(Poss(a1, s) \Rightarrow state(do(a1, s)) \circ f1 =_{St} state(s) \circ f2(0))\\
&\forall(s : Sit).\,(Poss(a2, s) \Rightarrow state(do(a2, s)) \circ f1 =_{St} state(s) \circ f4)\\
&\forall(y : Ob), (s : Sit).\,(Poss(a3(y), s) \Rightarrow\\
&\quad state(do(a3(y), s)) \circ f2(y) =_{St} state(s) \circ f3(foo(y)) \circ f3(foo(foo(y))))\\
&\forall(y : Ob), (s : Sit).\,(Poss(a4(y), s) \Rightarrow\\
&\quad state(do(a4(y), s)) \circ f3(y) =_{St} state(s) \circ f2(y))\\
&\forall(s : Sit).\,(Poss(a5, s) \Rightarrow state(do(a5, s)) \circ f4 =_{St} state(s) \circ f5^2)\\
&\forall(s : Sit).\,(Poss(a6, s) \Rightarrow state(do(a6, s)) \circ f5 =_{St} state(s) \circ f4)
\end{aligned}$$

the state update axioms of the domain description $\mathcal{D}_l$ wrt signature Σ_l. Note that the subsequent execution of $a3(y)$ and $a4(y)$ may strictly increase the size of the state associated with the initial situation, e.g. the state $f2(0)$ is increased to $f2(foo(0)) \circ f3(foo(foo(0)))$. Thereby, the execution of $a3(y)$ and $a4(y)$ does not only add a new fluent $f3$ but increases also the size of the sub-term parameter 0 of fluent $f2$ to $foo(0)$. □

$\mathcal{FC}_L$ with negative preconditions: $\mathcal{FC}_{LN}$

With the development of SLDENF-resolution [137, 146] it became possible to reason about logic programming representations of domains that allowed the use of negation in action preconditions. Accordingly, we define an extension of fragment $\mathcal{FC}_L$:

Definition 2.5.2 *Given a domain description $\mathcal{D}$ for some $\mathcal{FC}$ signature $\Sigma = (SORT, \preceq, FUN, REL)$ and X a variable declaration wrt Σ. $\mathcal{D}$ is called* linear *iff for every action $(a : Obj^k \rightarrow Act) \in FUN$ there is a state update axiom $\mathtt{SUA}^a_{\mathcal{D}}$ of the form:*

$$\begin{aligned}
\forall\bar{Y}.\,(\forall(s : Sit).\,(&Poss(a(\bar{Y}), s) \Rightarrow\\
&state(do(a(\bar{Y}), s), s) \circ St^-_a =_{St} state(s) \circ St^+_a))
\end{aligned}$$

and there is an axiom of the form

$$\begin{aligned}
\forall\bar{Y}.\,(\forall(s : Sit).\,(Poss(a(\bar{Y}), s) \Leftrightarrow\; &Holds(St^-_a, s) \wedge\\
&\textstyle\bigwedge_{i=1,\ldots,m} Holds(St^=_{a,i}, s) \wedge\\
&\textstyle\bigwedge_{j=1,\ldots,l} \neg Holds(St^n_{a,j}, s)\,)
\end{aligned}$$

Thereby $Y \subseteq X$, $\bar{Y} = Y_1, \ldots, Y_k$ is a sequence of the variable names of Y with $(Y_i : Obj) \in Y$ for $i = 1, \ldots, k$. $St^-_a, St^+_a, St^=_{a,1}, \ldots, St^=_{a,m}, St^n_{a,1}, \ldots, St^n_{a,l} \in T_{St,\Sigma}(Y)$.

The set of all linear $\mathcal{FC}$ domain descriptions with appropriate signatures is denoted by $\mathcal{FC}_{LN}$.

The possibility of reasoning about non-definite programs gained by SLDENF-resolution has been exploited for instance in [70], where it was proposed to represent a specificity relation between action descriptions by using negation. In Section 2.8 we will demonstrate how domain descriptions of [70] can be translated into $\mathcal{FC}_{LN}$ domains and vice versa. Consequently, the computational properties which will be investigated in Section 5.6 wrt $\mathcal{FC}_{LN}$ become almost immediately valid for the Fluent Calculus fragment of [70] as well.

Example 2.5.3 ($\mathcal{FC}_{LN}$ domain descriptions) *Consider the signature and the state update axiom of Example 2.5.1. Note that it is impossible in $\mathcal{FC}_L$ to add the condition $\neg Holds(brick\_at(x_2, y_2), s)$ to the action precondition axiom to ensure that the location (x_2, y_2) is empty before a brick can be moved there. In contrast to that in the fragment $\mathcal{FC}_{LN}$ we may simply define the following precondition axiom to ensure that $mv\_brick(x_1, y_1, x_2, y_2)$ is not executable if there is a brick at the destination:*

$$\begin{aligned}&\forall (s : Sit), (x_1, x_2 : XPos), (y_1, y_2 : YPos).\\ &\qquad (\, Poss(mv\_brick(x_1, y_1, x_2, y_2), s) \Leftrightarrow\\ &\qquad\; Holds(brick\_at(x_1, y_1), s) \wedge \neg Holds(brick\_at(x_2, y_2), s)\,).\end{aligned}$$

□

So far we have defined the two non-propositional fragments $\mathcal{FC}_L$ and $\mathcal{FC}_{LN}$. In both fragments the state update axioms are of the form described in Definition 2.4.1 and both fragments allow precondition axioms to contain a conjunction of $Holds(\_,\_)$ statements. $\mathcal{FC}_{LN}$ is more general in the sense that it also allows the conjunction to contain negative $Holds(\_,\_)$ statements. In the following we define the fragments $\mathcal{FC}_{PL}$, $\mathcal{FC}_{PLN}$ and $\mathcal{FC}_P$.

Propositional Fragments

The remaining fragments we consider in this work do not allow the use of domain specific sorts, i.e. sorts that are interpreted as subsets of the interpretation of *Obj*. Functions denoting fluents and actions are all constants. Consequently, the only variable that occurs in a precondition axiom or a state update axiom is a variable of sort *Sit*. We call Fluent Calculus fragments that are restricted in this way *propositional*. Thereby, the propositional version of $\mathcal{FC}_{LN}$ is denoted by $\mathcal{FC}_{PLN}$ and the propositional version of $\mathcal{FC}_L$ is denoted by $\mathcal{FC}_{PL}$.

To enable straightforward translation from Situation Calculus domains to Fluent Calculus domains, in [150] it has been proposed to provide an axiom restricting states such that a fluent may occur at most once in a state. If we consider a propositional Fluent Calculus equipped with such an axiom as defined below as $\mathcal{FC}_P$, we may expect similar computational properties as shown in [145] for a propositional Situation Calculus[13].

[13]Independently of our work, this has also been investigated in [69]. See Section 5.7 for some details.

Definition 2.5.3 *Given a domain description $\mathcal{D}$ for some $\mathcal{FC}$ signature $\Sigma = (SORT, \preceq, FUN, REL)$ and X a variable declaration wrt Σ.*

$\mathcal{D}$ is called propositional linear *iff $\mathcal{D}$ is linear and, additionally,*

- *all functions of FUN mapping to Fl are constants, and*
- *all functions of FUN mapping to Act are constants.*

The set of all propositional linear $\mathcal{FC}$ domain descriptions is denoted by $\mathcal{FC}_{PLN}$.

$\mathcal{D}$ is called propositional positive linear *iff $\mathcal{D}$ is positive linear and, additionally,*

1. *all functions of FUN mapping to Fl are constants, and*
2. *all functions of FUN mapping to Act are constants.*

The set of all propositional positive linear $\mathcal{FC}$ domain descriptions is denoted by $\mathcal{FC}_{PL}$.

$\mathcal{D}$ is called propositional *iff $\mathcal{D}$ is propositional linear and, additionally,*

1. *$\mathcal{D}$ contains the axiom*
   $\forall (s : Sit), (x, z : St). (state(s) =_{St} x \circ x \circ z \Rightarrow x =_{St} 1^{\circ})$ (NM)
2. *for all state update axioms $\mathtt{SUA}^{a}_{\mathcal{D}}$ and all fluents $f \in T_{Fl,\Sigma}(\emptyset)$:*
   $|St^{-}_{a}, f| \leq 1$, $|St^{+}_{a}, f| \leq 1$, $|St^{=}_{a,i}, f| \leq 1$, $|St^{n}_{a,j}, f| \leq 1$, *for all i, j.*

The set of all propositional $\mathcal{FC}$ domain descriptions is denoted by $\mathcal{FC}_{P}$.

The following two examples illustrate how domains can be specified in $\mathcal{FC}_{PL}$. Again, in the second example we introduce an abstract domain.

Example 2.5.4 (Airport continued) *Consider the signature Σ_a of Example 2.2.2. Assume that new planes may arrive anytime as represented by action queue_b (queue_s, respectively). A plane may only land (actions land_b and land_s) if there is sufficient bay area available on the ground and if the runway is free. Let a small plane get permission to land if one bay is available while for a big plane two bays are necessary. Additionally we assume that a big plane releases five passenger units while a small plane releases three (instead of representing each passenger by one fluent a fluent represents a "passenger unit" where one unit may correspond to some fixed number of passengers). If a plane* *takes off (actions take_off_b and take_off_s) it requires again the runway to be free and it releases the previously acquired bay space. Let*

$$\begin{aligned}
&\forall(s:Sit).\,(Poss(queue\_b,s) \Leftrightarrow \top)\\
&\forall(s:Sit).\,(Poss(queue\_s,s) \Leftrightarrow \top)\\
&\forall(s:Sit).\,(Poss(land\_b,s) \Leftrightarrow Holds(\mathrm{bay}^2 \circ plane\_q\_b \circ \mathrm{runway}, s)\\
&\forall(s:Sit).\,(Poss(land\_s,s) \Leftrightarrow Holds(\mathrm{bay} \circ plane\_q\_s \circ \mathrm{runway}, s))\\
&\forall(s:Sit).\,(Poss(take\_off\_b,s) \Leftrightarrow Holds(plane\_l\_b \circ \mathrm{runway}, s))\\
&\forall(s:Sit).\,(Poss(take\_off\_s,s) \Leftrightarrow Holds(plane\_l\_s \circ \mathrm{runway}, s))
\end{aligned}$$

be the precondition axioms and

$$\begin{aligned}
&\forall(s:Sit).\,(Poss(queue\_b,s) \Rightarrow\\
&\quad state(do(queue\_b,s)) =_{St} state(s) \circ plane\_q\_b)\\
&\forall(s:Sit).\,(Poss(queue\_s,s) \Rightarrow state(do(queue\_b,s)) =_{St} state(s) \circ plane\_q\_s)\\
&\forall(s:Sit).\,(Poss(land\_b,s) \Rightarrow\\
&\quad state(do(land\_b,s)) \circ \mathrm{bay}^2 \circ plane\_q\_b =_{St} state(s) \circ plane\_l\_b \circ \mathrm{passenger}^5)\\
&\forall(s:Sit).\,(Poss(land\_s,s) \Rightarrow\\
&\quad state(do(land\_s,s)) \circ \mathrm{bay} \circ plane\_q\_s =_{St} state(s) \circ plane\_l\_s \circ \mathrm{passenger}^3)\\
&\forall(s:Sit).\,(Poss(take\_off\_b,s) \Rightarrow\\
&\quad state(do(take\_off\_b,s)) \circ plane\_l\_b =_{St} state(s) \circ bay^2)\\
&\forall(s:Sit).\,(Poss(take\_off\_s,s) \Rightarrow\\
&\quad state(do(take\_off\_s,s)) \circ plane\_l\_s =_{St} state(s) \circ \circ bay)
\end{aligned}$$

the state update axioms of the domain description $\mathcal{D}_a$ wrt signature Σ_a.

Note that the subsequent execution of queue_b, land_b *and* take_off_b *may strictly increase the state associated with some initial situation: E.g. if the initial state is* $plane\_q\_b^2 \circ plane\_q\_s \circ \mathrm{runway} \circ \mathrm{bay}^4$*, then after executing the above sequence the state would change to* $plane\_q\_b^2 \circ plane\_q\_s \circ \mathrm{runway} \circ \mathrm{bay}^4 \circ \mathrm{passenger}^3$.

□

Example 2.5.5 (Abstract $\mathcal{FC}_{PL}$ domain) *Consider a $\mathcal{FC}_{PL}$ signature Σ_p containing the fluents $F_{\Sigma_p} = \{f1, \ldots, f5\}$ and the actions $A_{\Sigma_p} = \{a1, \ldots, a6\}$. Let*

$$\begin{aligned}
&\forall(s:Sit).\,(Poss(a1,s) \Leftrightarrow Holds(f1,s))\\
&\forall(s:Sit).\,(Poss(a2,s) \Leftrightarrow Holds(f1,s))\\
&\forall(s:Sit).\,(Poss(a3,s) \Leftrightarrow Holds(f2,s))\\
&\forall(s:Sit).\,(Poss(a4,s) \Leftrightarrow Holds(f3,s))\\
&\forall(s:Sit).\,(Poss(a5,s) \Leftrightarrow Holds(f4,s))\\
&\forall(s:Sit).\,(Poss(a6,s) \Leftrightarrow Holds(f5,s))
\end{aligned}$$

be the precondition axioms and

$$\begin{array}{l}\forall(s:Sit).\,(Poss(a1,s) \Rightarrow state(do(a1,s)) \circ f1 =_{St} state(s) \circ f2)\\ \forall(s:Sit).\,(Poss(a2,s) \Rightarrow state(do(a2,s)) \circ f1 =_{St} state(s) \circ f4)\\ \forall(s:Sit).\,(Poss(a3,s) \Rightarrow state(do(a3,s)) \circ f2 =_{St} state(s) \circ f3^2)\\ \forall(s:Sit).\,(Poss(a4,s) \Rightarrow state(do(a4,s)) \circ f3 =_{St} state(s) \circ f2)\\ \forall(s:Sit).\,(Poss(a5,s) \Rightarrow state(do(a5,s)) \circ f4 =_{St} state(s) \circ f5^2)\\ \forall(s:Sit).\,(Poss(a6,s) \Rightarrow state(do(a6,s)) \circ f5 =_{St} state(s) \circ f4)\end{array}$$

the state update axioms of the domain description $\mathcal{D}_p$ wrt signature Σ_p.

Note that the subsequent execution of $a3$ *and* $a4$ *may again strictly increase the state associated with the initial situation, e.g. the state* $f2$ *is increased to* $f2 \circ f3$. □

The definition of fragment $\mathcal{FC}_P$ explicitly restricts the number of copies of each fluent a state may contain. In $\mathcal{FC}_P$ the structure of the state update axioms must ensure that after executing an action in situation s, where the state associated with s contained every fluent not more than once, the state associated with the succeeding situation contains every fluent also at most once. Otherwise, the domain description does not have a model.

Example 2.5.6 ($\mathcal{FC}_P$ domain description) *Consider the signature Σ_a of Example 2.2.2. Let*

$$\begin{array}{l}\forall(s:Sit).\,(Poss(queue\_b,s) \Leftrightarrow \neg Holds(plane\_q\_b,s))\\ \forall(s:Sit).\,(Poss(queue\_s,s) \Leftrightarrow \neg Holds(plane\_q\_s,s))\\ \forall(s:Sit).\,(Poss(land\_b,s) \Leftrightarrow Holds(bay \circ plane\_q\_b \circ runway,s)\\ \forall(s:Sit).\,(Poss(land\_s,s) \Leftrightarrow Holds(bay \circ plane\_q\_s \circ runway,s))\\ \forall(s:Sit).\,(Poss(take\_off\_b,s) \Leftrightarrow Holds(plane\_l\_b \circ runway,s))\\ \forall(s:Sit).\,(Poss(take\_off\_s,s) \Leftrightarrow Holds(plane\_l\_s \circ runway,s))\end{array}$$

be the precondition axioms and

$$\begin{array}{l}\forall(s:Sit).\,(Poss(queue\_b,s) \Rightarrow\\ \quad state(do(queue\_b,s)) =_{St} state(s) \circ plane\_q\_b)\\ \forall(s:Sit).\,(Poss(queue\_s,s) \Rightarrow state(do(queue\_b,s)) =_{St} state(s) \circ plane\_q\_s)\\ \forall(s:Sit).\,(Poss(land\_b,s) \Rightarrow\\ \quad state(do(land\_b,s)) \circ bay \circ plane\_q\_b =_{St} state(s) \circ plane\_l\_b)\\ \forall(s:Sit).\,(Poss(land\_s,s) \Rightarrow\\ \quad state(do(land\_s,s)) \circ bay \circ plane\_q\_s =_{St} state(s) \circ plane\_l\_s)\\ \forall(s:Sit).\,(Poss(take\_off\_b,s) \Rightarrow\\ \quad state(do(take\_off\_b,s)) \circ plane\_l\_b =_{St} state(s) \circ bay)\\ \forall(s:Sit).\,(Poss(take\_off\_s,s) \Rightarrow\\ \quad state(do(take\_off\_s,s)) \circ plane\_l\_s =_{St} state(s) \circ bay)\end{array}$$

be the state update axioms of the domain description $\mathcal{D}_o$ wrt the signature Σ_a. These state update axiom ensure the above property by adding a fluent only if it does not occur in the state associated with the previous situation. If the state associated with $s0$ *contains only one copy of the fluents* $bay, plane\_l\_b, plane\_l\_s$ *and at most one copy of each of the fluents* $runway, plane\_q\_b, plane\_q\_s$, *then it can be shown by induction that executing the given actions may not lead to a situation where the associated state contains multiple copies of some fluent, i.e. a model of the domain description exists (for the domain considered in [68] this has been done explicitly).*

Note that in $\mathcal{FC}_P$ it is not possible to model the number of arriving passengers and the number of queueing planes since they are not bounded (in the $\mathcal{FC}_P$ domain we decided to discard the fluent passenger and to restrict the number of queueing planes to one for each size of plane. On the other hand, in the $\mathcal{FC}_{PL}$ domain the number of landed planes is bounded by the available bay area which in turn is bounded by the state associated with $s0$. *Although such finite boundaries could be hardwired in $\mathcal{FC}_P$ (e.g., with four different fluents we could encode up to 16 different numbers) we decided to weaken the model further by allowing only one plane (small or big) to be at the airport. A similar problem occurs if the airport to be modelled has more than one runway.* □

Note that there are obvious relations depicted in Figure 2.1 between the $\mathcal{FC}$ fragments of Definitions 2.5.1, 2.5.2 and 2.5.3. The symbol $\subseteq$ denotes that every domain description of the fragment to the left of $\subseteq$ is also a domain description of the fragment to the right of $\subseteq$. For instance, every $\mathcal{FC}_{PL}$ domain description is also a $\mathcal{FC}_L$ domain description and a $\mathcal{FC}_{LN}$ domain description. If we compare the fragments based purely on syntactical criteria then $\mathcal{FC}_{LN}$ is the most expressive fragment we defined so far.

$$\begin{tikzcd}
 & \mathcal{FC}_{LN} & \\
\mathcal{FC}_L \arrow[ur, "\subseteq"] & & \mathcal{FC}_{PLN} \arrow[ul, "\subseteq"'] \\
\mathcal{FC}_{PL} \arrow[u, "\subseteq"'] \arrow[urr, "\subseteq"'] & & \mathcal{FC}_P \arrow[u, "\subseteq"']
\end{tikzcd}$$

Figure 2.1: Syntactical relations between $\mathcal{FC}$ fragments.

In Section 2.7 we will develop a method that allows to compare particular domain descriptions based on the models they describe. Then we may also compare $\mathcal{FC}$ fragments based on the sets of models they can describe. This will provide a more interesting meaning to the "expressibility" of a $\mathcal{FC}$ fragment.

A Normal Form for Modal Properties

For propositional domains we may substitute the conjunction of *Holds*-statements in each precondition axiom by an equivalent single *Holds*-statement. Due to the equational theory `AC1` which must be satisfied by every model $\mathcal{M}$ of any Fluent Calculus domain, if $g_1, \ldots, g_m$ are state terms containing only variable free fluents $f_1, \ldots, f_n$, and s is any situation in $\mathcal{M}$ then:

$$\begin{array}{l} \mathcal{M} \models \mathit{Holds}(f_1^{\max(|g_1,f_1|,\ldots,|g_m,f_1|)} \circ \cdots \circ f_n^{\max(|g_1,f_n|,\ldots,|g_m,f_n|)}, s) \\ \text{iff} \quad \mathcal{M} \models \bigwedge_{1\leq i\leq m} \mathit{Holds}(g_i, s). \end{array} \tag{2.3}$$

This follows from the fact that whenever $\mathit{Holds}(g, s)$ is valid then $\mathit{Holds}(g', s)$ is also valid if $\mathcal{M} \models \exists(z : St).\, g' \circ z =_{St} g$, i.e. if the multiset represented by g contains the multiset represented by g' as a sub-multiset. A conjunction of statements $\mathit{Holds}(g_i, s)$ is equivalent to the statement $\mathit{Holds}(g, s)$ where g represents the multiset which contains each multiset represented by some g_i but not more.

Consequently, it is sufficient to consider precondition axioms of $\mathcal{FC}_{PL}$ to contain a single *Holds*-statement, only:

$$\forall(s : Sit).\, (\mathit{Poss}(a, s) \Leftrightarrow \mathit{Holds}(St_a^- \circ St_a^=, s)) \tag{2.4}$$

where $St_a^-, St_a^= \in T_{St,\Sigma}(Y)$.

Example 2.5.7 *Consider the domain description $\mathcal{D}_a$ of Example 2.5.4. The third state update axiom could have been equivalently represented as:*

$$\begin{array}{l} \forall(s : Sit).\, (\mathit{Poss}(land\_b, s) \Leftrightarrow \\ \qquad \mathit{Holds}(bay^2, s) \wedge \mathit{Holds}(bay \circ plane\_q\_b, s) \wedge \mathit{Holds}(runway, s)) \end{array}$$

□

However, if the fluents do not need to be variable free, as for the fragments $\mathcal{FC}_L$ and $\mathcal{FC}_{LN}$, it is not always possible to find a single *Holds*-statement representing a conjunction of *Holds*-statements in this way. E.g., the conjunction $\mathit{Holds}(f(x, y), s) \wedge \mathit{Holds}(f(succ(z), succ(z)), s)$ can only be represented by $\mathit{Holds}(f(succ(z), succ(z)) \circ f(x, y), s)$ if the disequation $f(succ(z), succ(z)) \neq f(x, y)$ is fulfilled. This disequation is satisfied by the following infinite set T of instances of $f(x, y)$: $\{f(0, succ(x)), f(succ(x), 0), f(succ(0), succ(succ(x))), f(succ(succ(x)), succ(0)), \ldots\}$. None of the elements of T is an instance of another element and the generalisation of any two elements has instances which do not satisfy the disequation. Therefore, the disequation can not be represented by a finite disjunction of equalities and we can not substitute every precondition axiom of $\mathcal{FC}_L$ or $\mathcal{FC}_{LN}$ by a finite set of axioms in the style of Formula 2.4.

Due to the equational theory `AC1` the negative *Holds*-predicates appearing in action precondition axioms of $\mathcal{FC}_{PLN}$ can be rewritten using the following relation where g is a state term containing only variable free fluents $f_1, \ldots, f_n$ and $\mathcal{M}$ a model:

$$\begin{array}{l} \mathcal{M} \models \neg Holds(g, s) \\ \text{iff} \quad \mathcal{M} \models \bigvee_{1 \leq i \leq n} \neg Holds(f_i^{|g, f_i|}, s). \end{array} \tag{2.5}$$

To understand the modal properties described by boolean combinations of *Holds*-statements, it is helpful to view a single *Holds*-statement as a conjunction of lower bounds on the number of fluents, e.g. $Holds(f_1 \circ f_2 \circ f_2, s)$ is valid in all situations s where the associated state contains at least one copy of f_1 and at least two copies of f_2. In contrast to this, a negative *Holds*-statement represents a disjunction of upper bounds, e.g. $\neg Holds(f_1 \circ f_2 \circ f_2, s)$ is valid in all situations s where the associated state contains no copy of f_1 or at most one copy of f_2. To simplify the notation of a conjunction of upper bounds we introduce the following shortcut:

$$\overline{Holds(g, s)} \stackrel{\text{def}}{=} \bigwedge_{f \in Fl \wedge |g, f| > 0} \neg Holds(f^{|g, f|}, s) \tag{2.6}$$

Informally, $\overline{Holds(g, s)}$ is true if the number of copies of a fluent f occurring in the state associated with s is smaller then the number of copies in g, for all fluents in g. For example, if we consider again the brick domain, $\overline{Holds(brick\_at(2, 4) \circ brick\_at(2, 3), s)}$ is valid if $state(s) = brick\_at(2, 5)$, but is not valid if $state(s) = brick\_at(2, 4)$.

Since every boolean combination of $Holds(g, s)$ formulas can be transformed into disjunctive normal-form, it follows using Equations 2.3 and 2.5 that every boolean combination of $Holds(g, s)$ formulas where the fluents of g are variable free can be transformed into a disjunctive normal-form of *Holds*- and $\overline{\textit{Holds}}$-

statements ($f_1, \dots, f_n$ are the fluents that appear in the state terms):

$$\begin{array}{l}
\mathcal{M} \models \bigvee\limits_{0 \le h \le k} (\bigwedge\limits_{0 \le i \le l} \mathit{Holds}(g_i^h, s) \wedge \bigwedge\limits_{0 \le i \le m} \neg \mathit{Holds}(\bar{g}_i^h, s)) \\
\text{iff} \\
\mathcal{M} \models \bigvee\limits_{0 \le h \le k} (\bigwedge\limits_{0 \le i \le l} \mathit{Holds}(g_i^h, s) \wedge \\
\qquad \bigvee\limits_{\substack{0 \le i_0 \le n \\ \bar{g}_{i_0}^h \neq 1^\circ}} \cdots \bigvee\limits_{\substack{0 \le i_m \le n \\ \bar{g}_{i_m}^h \neq 1^\circ}} \bigwedge\limits_{0 \le j \le m} \neg \mathit{Holds}(\bar{g}_{i_j}^h, s)) \\
\text{iff} \\
\mathcal{M} \models \bigvee\limits_{0 \le h \le k} \bigvee\limits_{\substack{0 \le i_0 \le n \\ \bar{g}_{i_0}^h \neq 1^\circ}} \cdots \bigvee\limits_{\substack{0 \le i_m \le n \\ \bar{g}_{i_m}^h \neq 1^\circ}} \\
\quad (\mathit{Holds}(f_0^{max(|g_0^h, f_0|, \dots, |g_l^h, f_0|)} \circ \cdots \circ f_n^{max(|g_0^h, f_n|, \dots, |g_l^h, f_n|)}, s) \\
\quad \wedge \\
\quad \overline{\mathit{Holds}(f_0^{min_{>0}(|\bar{g}_{i_0}^h, f_0|, \dots, |\bar{g}_{i_m}^h, f_0|)} \circ \cdots \circ f_n^{min_{>0}(|\bar{g}_{i_0}^h, f_n|, \dots, |\bar{g}_{i_m}^h, f_n|)}, s)})
\end{array} \tag{2.7}$$

where we denoted $f_{i_j}^{|\bar{g}_j^h, f_{i_j}|}$ by $\bar{g}_{i_j}^h$. The function $\min_{>0}$ computes the minimal element of a set, but considers only elements that are greater then 0 and if all elements are 0 the result is 0.

The Equation 2.7 enables us to rewrite any formula that consists only of *Holds*(g, s) statements referring to the same situation s into a disjunction of sets of upper and lower bounds for fluent occurrences in the state associated with s.

Since an action precondition axiom which defines *Poss* by an arbitrary boolean combination of *Holds*(g, s) can be transformed using Equation 2.7 into the form

$$\forall \bar{Y}. (\forall (s : \mathit{Sit}). (\mathit{Poss}(a(\bar{Y}), s) \Leftrightarrow \Delta_1 \vee \cdots \vee \Delta_n))$$

for some finite n where each Δ_i contains neither implicit nor explicit disjunction, the domain description may be transformed into a domain description without use of disjunctive action precondition axioms by introducing a new action and corresponding state update and action precondition axioms for each Δ_i. Consequently, it is sufficient in this work to consider precondition axioms of $\mathcal{FC}_{PLN}$ where neither the explicit nor implicit (by $\neg$*Holds*(g, s) according to Equation 2.5) use of disjunction is allowed (see also Section 2.8):

$$\begin{array}{l}
\forall \bar{Y}. (\forall (s : \mathit{Sit}). (\mathit{Poss}(a(\bar{Y}), s) \Leftrightarrow \\
\qquad \mathit{Holds}(St_a^-(\bar{Y}) \circ St_a^=(\bar{Y}), s) \wedge \overline{\mathit{Holds}(St_a^n(\bar{Y}), s)}))
\end{array} \tag{2.8}$$

So far we presented how dynamic systems can be described as Fluent Calculus domains. In the following section we define the language which is used to describe temporal properties of such systems. We call this language the *query language* $\mathcal{QL}$. The semantics of $\mathcal{QL}$ is defined using the semantics of Fluent Calculus domains.

2.6 The Query Logic

Since we want to use the query logic to reason about temporal properties of a system, the propositions whose value depends only on the value of a single situation variable are particularly important. In the query logic such propositions are considered to be atomic. Formally, we define the set of these atomic propositions as the set of first-order formulas with a free variable of sort *Sit*. Additionally, all other variables occuring in such a formula are of sort different than *Sit* and they must be bounded by some quantifier:

Definition 2.6.1 *Let X be a variable declaration wrt some Fluent Calculus signature Σ. We define $X_{Sit} \equiv \{(x : Sit) \mid (x : Sit) \in X\}$ and $\overline{X_{Sit}} = X \setminus X_{Sit}$, for any $(y : Sit)$,*

$$F_{(y:Sit),\Sigma}(X) = \{\phi \mid \phi \in F_\Sigma(\overline{X_{Sit}} \cup \{(y : Sit)\}) \wedge Vars(\phi) = \{(y : Sit)\}\}$$

For example, if we consider the signature Σ_b of Example 2.2.1, the formula

$$\forall(y_1 : YPos).\exists(y_2 : XPos).\,Holds(brick\_at(y_2, y_1), x)$$

of $F_{(x:Sit),\Sigma_b}(\{x, z_1, z_2\})$ describes that in situation x in every row of the grid there exists at least one brick.

Let $\phi \in F_{(x:Sit),\Sigma_b}(()Y)$, then by $\phi[t]$, where t is a term of sort *Sit*, we denote $\phi\{x \mapsto t\}$, .

The set $F_{(x:Sit),\Sigma}(X)$ is very powerful in the sense that an element may contain arbitrary first-order formulas. The value of these sub-formulas does not need to depend on the situation variable x. Since first-order logic is only semi-decidable we can not expect a logic which uses all elements of $F_{(x:Sit),\Sigma}(X)$ as atomic propositions to be decidable. Consequently, to help to establish low bounds of undecidability, we define subsets of $F_{(x:Sit),\Sigma}(X)$. These subsets are particularly important for the propositional fragments of the Fluent Calculus:

Definition 2.6.2 *Let Σ be a Fluent Calculus signature and Y be a variable declaration and $(x : Sit) \in Y$ a variable of sort Sit. Then we define*

$$L_{(x:Sit),\Sigma}(Y) = \{\exists\bar{Z}.\,(Holds(g, x)) \mid g \in T_{St,\Sigma}(Y) \wedge Z = Vars(g)\}.$$

We define $\Lambda_{L_{(x:Sit),\Sigma}(Y)}$ as the set of formulas constructed from propositions of $L_{(x:Sit),\Sigma}(Y)$ by applying boolean operations.

For example, $\exists(z : XPos).\, Holds(brick\_at(z, 4), x)$ is an element of the language $L_{(x:Sit),\Sigma_b}(Y)$ for the signature Σ_b of Example 2.2.1 and $(z : St), (x : Sit) \in Y$.

Since every element ϕ of $L_{(x:Sit),\Sigma_b}(Y)$ contains precisely one free variable named x at the same position, we omit this variable when we refer to elements of $L_{(x:Sit),\Sigma_b}(Y)$, e.g, instead of $\exists(z : XPos).\, Holds(brick\_at(z, 4), x)$ we write simply $\exists(z : XPos).\, Holds(brick\_at(z, 4))$. Since in most contexts it is clear to which variable of sort *Sit* we refer to, we often omit the explicit definition of a name and write $L_\Sigma(Y)$ or simply L_Σ if Y contains no other variables.

The syntax of the most general query language we consider is given in the following definition. The atomic propositions of the logic are given by any subset of $F_{(x:Sit),\Sigma}(X)$. The query language is equivalent to the one of [119, 144] for the Situation Calculus. However, here we define its semantics using the semantics of Fluent Calculus domain descriptions. Note that we use an additional sort $\mathcal{B}$ in the defintion of the syntax. In the most general case defined below the interpretation of $\mathcal{B}$ is given by the power-set of the interpretation of sort *Sit*.

Definition 2.6.3 *Let Σ be a Fluent Calculus signature, X a variable declaration wrt Σ, Y a variable declaration of the form $(y : \mathcal{B})$. Let $L \subseteq F_{(x:Sit),\Sigma}(X)$. The* query logic *$\mathcal{QL}$ over Σ wrt X and Y and L is defined as follows: The set of* queries *is the smallest set containing*

1. *$x \in y$, for $(x : Sit) \in X$ and $(y : \mathcal{B}) \in Y$,*
2. *$x_1 \xrightarrow{a} x_1$, for $(x_1 : Sit), (x_2 : Sit) \in X$, $a \in T_{Act,\Sigma}(\emptyset)$,*
3. *$x_1 \leq x_2$, for $(x_1 : Sit), (x_2 : Sit) \in X$,*
4. *p for $p \in L$ and $(x : Sit) \in X$,*
5. *$\phi_1 \wedge \phi_2$, $\neg\phi$, $\exists(x : Sit).\, \phi$, $\exists(y : \mathcal{B}).\, \phi$, for queries ϕ_1, ϕ_2, ϕ with $(x : Sit) \in X$, $(y : \mathcal{B}) \in Y$.*

Let $\mathcal{M}$ be a model of some domain description of Σ, ϕ a query, $\mathcal{B}^{\mathcal{M}} = \mathcal{P}(Sit^{\mathcal{M}})$, $v_{Sit} : X_{Sit} \rightarrow Sit^{\mathcal{M}}$ and $v_{\mathcal{B}} : Y \rightarrow \mathcal{B}^{\mathcal{M}}$ valuations. Then

$\mathcal{M}, v_{Sit}, v_{\mathcal{B}} \models x \in y$	*iff*	$v_{Sit}(x) \in v_{\mathcal{B}}(y)$
$\mathcal{M}, v_{Sit}, v_{\mathcal{B}} \models x_1 \xrightarrow{a} x_1$	*iff*	$\mathcal{M} \models (v_{Sit}(x_1) < do(a, v_{Sit}(x_1)) \wedge v_{Sit}(x_2) =_{Sit} do(a, v_{Sit}(x_1)))$
$\mathcal{M}, v_{Sit}, v_{\mathcal{B}} \models x_1 \leq x_2$	*iff*	$\mathcal{M} \models v_{Sit}(x_1) \leq v_{Sit}(x_2)$
$\mathcal{M}, v_{Sit}, v_{\mathcal{B}} \models p$	*iff*	$\mathcal{M} \models p[v_{Sit}(x)]$
$\mathcal{M}, v_{Sit}, v_{\mathcal{B}} \models \phi_1 \wedge \phi_2$	*iff*	$\mathcal{M}, v_{Sit}, v_{\mathcal{B}} \models \phi_1$ *and* $\mathcal{M}, v_{Sit}, v_{\mathcal{B}} \models \phi_2$
$\mathcal{M}, v_{Sit}, v_{\mathcal{B}} \models \neg\phi$	*iff*	$\mathcal{M}, v_{Sit}, v_{\mathcal{B}} \not\models \phi$
$\mathcal{M}, v_{Sit}, v_{\mathcal{B}} \models \exists(x : Sit).\, \phi$	*iff*	*there is* $v'_{Sit} : X_{Sit} \rightarrow Sit^{\mathcal{M}}$ *s.t.* $\mathcal{M}, v'_{Sit}, v_{\mathcal{B}} \models \phi$ *and* $v'_{Sit}(z) = v_{Sit}(z)$ *for all* $(z : Sit) \in X_{Sit}$ *with* $z \neq x$.
$\mathcal{M}, v_{Sit}, v_{\mathcal{B}} \models \exists(x : \mathcal{B}).\, \phi$	*iff*	*there is* $v'_{\mathcal{B}} : Y \rightarrow \mathcal{B}^{\mathcal{M}}$ *s.t.* $\mathcal{M}, v_{Sit}, v'_{\mathcal{B}} \models \phi$ *and* $v'_{\mathcal{B}}(z) = v_{\mathcal{B}}(z)$ *for all* $(z : \mathcal{B}) \in Y$ *with* $z \neq x$.

Using this query language we may now precisely state the planning problem of Section 2.1 as determining the satisfiability of the following formula:

$$\exists(s : Sit).\,(\lambda_0[s] \wedge \exists(s' : Sit).\,(s' \geq s \wedge \lambda_e[s'])) \tag{2.9}$$

where $\lambda_0, \lambda_e \in \Lambda_{L_{(x:Sit),\Sigma}(Y)}$ for some variable declaration Y wrt some Fluent Calculus signature Σ and $(x : Sit) \in Y$.

Alternatively, we may also use the equivalent formula

$$\begin{aligned} &\exists(s : Sit).\,(\lambda_0[s] \wedge \lambda_e[s]) \vee \\ &\exists(s : Sit).\,(\lambda_0[s] \wedge \exists(s' : Sit).\,(s' > s \wedge \lambda_e[s'])) \end{aligned} \tag{2.10}$$

which consists of two sub-formulas that can be independently checked. The sub-formula on the first line is satisfied in some model $\mathcal{M}$ if there is a situation in $\mathcal{M}$ where both λ_0, and λ_e, are fulfilled. The sub-formula on the second line is satisfied in some model $\mathcal{M}$ if there is a situation in $\mathcal{M}$ where λ_0 holds and from where it is possible to reach a *different* situation (by executing a non-empty sequence of actions) where λ_e holds.

The conjunctive planning problem can be represented in the same way but λ_e is restricted to be a conjunction of atoms of $L_{(x:Sit),\Sigma}(Y)$.

The extended planning problem can be expressed as the validity of the formula:

$$\forall(s : Sit).\,(\lambda_0(s) \Rightarrow \exists(s' : Sit).\,(s' \geq s \wedge \lambda_e(s'))) \tag{2.11}$$

The Figure 2.2 illustrates the relation between the problems described above. E.g., if we succeed in developing a calculus which allows to decide the entailment problem we may apply this calculus to the other problems as well. Note furthermore that if λ_0 characterises precisely which fluents appear in the state associated with s then we may apply a calculus solving the extended planning problem to solve the planning problem. On the other hand, if we show that a planning problem with such a λ_0 is undecidable we can conclude that the extended planning problem is also undecidable.

$$\begin{tikzcd}
& \text{entailment problem} & \\
\text{planning problem} \arrow[ur, "\subset"] & & \text{extended planning problem} \arrow[ul, "\subset"'] \\
\text{conjunctive planning problem} \arrow[u, "\subset"] & &
\end{tikzcd}$$

Figure 2.2: Relations between the problem classes.

2.7 Models and Transition Systems

Highly specialised logics have been developed to describe the dynamic properties of systems. Many of them are derived from *modal logics* introduced in [97]. Modal logics extend classical logic by additional operators, so called *modalities.* In [83] it has been proposed to consider transition systems as models of such logics. Some of the more specialised logics derived from modal logics are often also called *temporal logics* [124, 123, 122, 2, 86, 31]. Modal logics can be used in two different ways:

1. Modal formulas can be used to define the dynamic properties of a transition system. In this case, if we want to verify whether a transition system fulfills some dynamic property of interest, we have to derive the modal formula representing this property from the modal formulas defining the system. Often, however, the properties used to define the transition system and the properties which have to be verified require logical languages of different expressive power. Then the development of an appropriate modal logic can be very difficult, since on one hand, the logic must be sufficiently expressive for both definition of the transition system and representation of properties of interest. On the other hand, to simplify the development of efficient calculi, the logic should not be more expressive than required.
2. In the *model checking* approach [20] the dynamic system can be defined in a different language than the one used to describe the properties to be verified. But it must be ensured that the semantics of the formula to be verified and the semantics of the system definition are given by common classes of dynamic systems. We may then verify a dynamic property by checking the intersection of the set of models associated with the system description with the set of models associated with the formula to be verified for emptiness.

In this work we will follow the second approach by relating domain descriptions to well known models of computation and formulas describing dynamic properties, e.g. the planning problem, to standard temporal logics. As the common class of models we will consider a combination of labelled transition systems (graphs where transitions carry action labels) and Kripke structures (graphs where nodes are labelled by propositions). This results in the following definition of a P-valued transition system as a mathematical representation of the possible behaviours of a dynamic system. Thereby P is a parameter denoting the set of propositions used to label the nodes of the transition system.

Definition 2.7.1 *A tuple* $\Theta = (S, \rightarrow, A, \alpha)$ *is called a* P-valued transition system *if* S *is a non-empty set of* states, A *is a non-empty set of* transition labels *and* $\rightarrow$ *is a family of relations such that for each* $t \in A$, $\xrightarrow{t} \in \rightarrow$ *and* $\xrightarrow{t} \subseteq S \times S$. α *is a mapping* $P \times S \rightarrow \{\top, \bot\}$ *where* P *is a non-empty set called* propositions.

A path *π in Θ is defined as a (possibly infinite) sequence of states such that for any two subsequent states z, z' there exists $t \in A$, such that $z \xrightarrow{t} z'$. By π_n the n-th element of π is denoted for $n \in \mathbb{N}$. The subsequence of π starting in π_n is denoted by $\pi_{\geq n}$. A path π is called a* run *iff π is of length ω or, if π^* is the final state of π, i.e. there is no $z \in S$ and $t \in A$ such that $\pi^* \xrightarrow{t} z$.*

Θ is called rooted *if there is a $z_0 \in S$ such that to every $z \in S$ exists a path $z_0 z_1 \dots z \dots$. Then, z_0 is called the* root[14] *of Θ. A run π is called a* branch *of Θ iff Θ is rooted and π_0 is the root.*

To compare transition systems we use the notion of strong bisimulation (e.g. [64]). Strong bisimulation is of particular importance in this work, since the validity of any formula of modal logics we will consider here is invariant under strong bisimulation, i.e. proving a property for one system is sufficient to establish it for all strongly bisimilar systems. Since all bisimulations considered here are strong, we omit the word "strong". Since we want to be able to compare transition systems where the nodes are labelled by elements of different sets of propositions, we have to extend the common definition of bisimulation slightly by including also mappings Ψ_1, Ψ_2 between these sets. If we want to show that a P_1-valued transition system and a P_2-valued transition system are bisimilar then we have to show that both mappings, $\Psi_1 : P_1 \to P_2$ and $\Psi_2 : P_2 \to P_1$, are isomorphisms. Note that such bisimulations are also called *zig-zags*, [8].

Definition 2.7.2 (bisimulation) *Let $\Theta_1 = (S_1, \{\xrightarrow{a}_1 \mid a \in A_1\}, A_1, \alpha_1)$, $\Theta_2 = (S_2, \{\xrightarrow{a}_2 \mid a \in A_2\}, A_1, \alpha_2)$ be P_1-valued and, respectively, P_2-valued transition systems. Let Φ be a relation $\Phi \subseteq S_1 \times S_2$. Φ is called a* bisimulation *if there exist mappings $\Psi_1 : P_1 \to P_2$ and $\Psi_2 : P_2 \to P_1$ such that $(z_1, z_2) \in \Phi$ implies*

1. *for all $l \in P_1$, $\alpha_1(l, z_1) = \top$ iff $\alpha_2(\Psi_1(l), z_2) = \top$,*
2. *for all $l \in P_2$, $\alpha_2(l, z_2) = \top$ iff $\alpha_1(\Psi_2(l), z_1) = \top$,*
3. *with $z_1 \xrightarrow{t_1}_1 z_1'$, exists $z_2 \xrightarrow{t_1}_2 z_2'$ such that $(z_1', z_2') \in \Phi$*
4. *with $z_2 \xrightarrow{t_2}_2 z_2'$, exists $z_1 \xrightarrow{t_2}_1 z_1'$ such that $(z_1', z_2') \in \Phi$*

$z_1 \in S_1$ and $z_2 \in S_2$ are called *bisimilar*, written $z_1 \sim z_2$, if there is a bisimulation Φ such that $(z_1, z_2) \in \Phi$. Θ_1 and Θ_2 are called *bisimilar* if there is a bisimulation Φ such that for all $z_1 \in S_1$ there exists $z_2 \in S_2$ such that $z_1 \sim z_2$ and for all $z_2 \in S_2$ there exists $z_1 \in S_1$ such that $z_1 \sim z_2$. Note, that for rooted transition systems it is sufficient to show the existence of a bisimulation for the root states.

In the following we aim to characterise models of domain descriptions of the Fluent Calculus by transition systems. According to Definition 2.7.1, a transition

[14]Note that according to this definition a transition system may have several roots, e.g. if the system contains two states z_1, z_2 and there are transitions from z_1 to z_2 and from z_2 to z_1, respectively.

system contains states, transition labels, relations describing transitions between states and state dependent propositions. An obvious choice might identify a state of the transition system with a state in the Fluent Calculus model (an element of the interpretation $St^{\mathcal{M}}$ of the sort St in the model $\mathcal{M}$). However, state update axioms may in general also depend on the execution history[15] which is not reflected in the structure of elements of $St^{\mathcal{M}}$. Hence, we follow a more powerful approach by generating the *unravelled* (cf. [141]) transition system, i.e. elements of $Sit^{\mathcal{M}}$ are considered to be states of the transition system. The set of transition labels is identified with $Act^{\mathcal{M}}$ – the set of actions in model $\mathcal{M}$. We define the transition relation to contain a pair $(s, do^{\mathcal{M}}(a, s))$ of situations if it is possible to execute some action a in s. Finally, the mapping from situation dependent propositions of $F_{(x:Sit),\Sigma}(X)$ in some situation s to $\{\top, \bot\}$ reflects the validity of the proposition in s of the model $\mathcal{M}$.

Definition 2.7.3 *Let $\mathcal{M}$ be a model of some domain description $\mathcal{D}$ wrt the Fluent Calculus signature Σ. Let X be a variable declaration and $L \subseteq F_{(x:Sit),\Sigma}(X)$. Then we define $K(\mathcal{M}, L) = (Sit^{\mathcal{M}}, \rightarrow_{\mathcal{M}}, Act^{\mathcal{M}}, \alpha_{\mathcal{M}})$ where $\rightarrow_{\mathcal{M}}$ is the set of relations $\xrightarrow{a}_{\mathcal{M}}$ with $(s, s') \in \xrightarrow{a}_{\mathcal{M}}$ iff $s < do^{\mathcal{M}}(a, s)$ and $s' =_{Sit} do^{\mathcal{M}}(a, s)$, for $s, s' \in Sit^{\mathcal{M}}$ and $a \in Act^{\mathcal{M}}$.[16] For $\phi \in L$ and $s \in Sit^{\mathcal{M}}$ let $\alpha_{\mathcal{M}}(\phi, s) = \top$ iff $\mathcal{M} \models \phi\{x \mapsto s\}$.*

Now we show that significant parts of a model of a domain description can be characterised as a transition system. This transition system has the form of a tree:

Lemma 2.7.1 *Given a domain description $\mathcal{D}$ for the Fluent Calculus signature Σ, a variable declaration X wrt Σ, a model $\mathcal{M}$ of $\mathcal{D}$, and $L \subseteq F_{(x:Sit),\Sigma}(X)$.*

1. *$K(\mathcal{M}, L)$ is a L-valued transition system,*
2. *$K(\mathcal{M}, L)$ is a (possibly infinitely branching) tree rooted in $s0^{\mathcal{M}}$.*

Proof 2.7.1 The first proposition follows directly from definition of $K(\mathcal{M}, L_\Sigma)$ and Definition 2.7.1.

Since only Herbrand-$\mathrm{E}_{\mathcal{FC}}$-models are considered, the set of situations is inductively defined by $do^{\mathcal{M}}$ and $s0^{\mathcal{M}}$. From the axioms $\mathtt{O}_{Sit}$ follows that $s0^{\mathcal{M}}$ is root and ancestor wrt. $<$ of every other $s \in Sit^{\mathcal{M}}$ which can possibly be reached by executing actions. The axioms $\mathtt{O}_{Sit}$ ensure that $<$ is interpreted as an order on the execution history. Furthermore, from the extended unique name assumption follows that two situations $s_1, s_2 \in Sit^{\mathcal{M}}$ are equivalent iff they consist of identical action sequences (note that these sequences are always finite). Since every situation $s \in Sit^{\mathcal{M}}$ which is different from $s0^{\mathcal{M}}$ is of the form $do^{\mathcal{M}}(a, s')$, for $a \in Act^{\mathcal{M}}$ and $s' \in Sit^{\mathcal{M}}$, s has an unique predecessor s' wrt $<^{\mathcal{M}}$. □

[15]Note that such systems do not fulfil the *Markov property*.

[16]Equally, we can define $(s, s') \in \xrightarrow{a}_{\mathcal{M}}$ iff $\mathcal{M} \models Poss(a, s)$ and $s' =_{Sit} do^{\mathcal{M}}(a, s)$, without considering the definition of $<$.

Note that we cannot associate a particular transition system with a domain description $\mathcal{D}$ as $\mathcal{D}$ does not determine the value of the function *state* for the initial situation $s0$ in each model of $\mathcal{D}$. However, once the interpretation of $state(s0)$ is given the interpretation of *state* for all other situations is completely characterised by the precondition and state update axioms of $\mathcal{D}$. Consequently, $\mathcal{D}$ has as many models as there are interpretations of $state(s0)$.

The above lemma associates sets of transition systems with domain descriptions. By using the notion of bisimulation we may classify transition systems and, hence, also the domain descriptions describing these systems. Furthermore, the lemma provides the semantics we need to use well known modal/temporal logics as query languages for the Fluent Calculus.

2.8 Other Fluent Calculus fragments

Here we show under which circumstances non-deterministic Fluent Calculus domain descriptions can be represented as deterministic ones and which properties are preserved. The result allows us in many cases to reduce planning problems for non-deterministic $\mathcal{FC}$ domains to planning problems for deterministic $\mathcal{FC}$ domains. We achieve the result by investigating the transition systems associated with $\mathcal{FC}$ domains.

In [70] the simple Fluent Calculus has been extended by a *specificity relation* over the state update axioms (see below for a definition). We demonstrate how domains of this extended simple Fluent Calculus can be translated into $\mathcal{FC}_{LN}$ domains. Thereby, the translation does not alter the set of transition systems associated with a domain description. We also show that there are $\mathcal{FC}_{LN}$ domains that cannot be represented as simple Fluent Calculus domains with specificity relation.

Non-deterministic Domain Descriptions

According to Section 2.4 the effect of a state update axiom of a non-deterministic domain descriptions may be described by a disjunction of state equations:

$$\Gamma = \Gamma_1 \vee \cdots \vee \Gamma_m \tag{2.12}$$

where $\Gamma_1, \ldots, \Gamma_m$ represent state equations. Thereby the effect of such a non-deterministic state update axiom is determined by one of the state equations Γ_j, $j = 1, \ldots, m$.

If $Vars(\Gamma_1)$,..., $Vars(\Gamma_m)$ are all disjoint[17], this non-deterministic domain description may be transformed into a deterministic domain description by intro-

[17]If they are not disjoint we have to solve disequations as described on Page 49. Thereby Γ can be transformed into a disjunction of Γ_j such that their free variables are disjoint. However, this disjunction might then consist of an infinite number of elements and so does the resulting deterministic domain description.

ducing a new action and corresponding state update and action precondition axioms for each possible effect equation. In the following we define a mapping from a non-deterministic domain description $\mathcal{D}$ to a deterministic domain description $\mathcal{D}$.

Definition 2.8.1 *Let $\mathcal{D}$ be some domain wrt the Fluent Calculus signature $\Sigma = (SORT, \preceq, FUN, REL)$ and X some variable declaration wrt Σ. For every state update axiom $\mathtt{SUA}^a_{\mathcal{D}}$, let $\Gamma^a = \Gamma^a_1 \vee \cdots \vee \Gamma^a_{m_a}$ denote the effect of $\mathtt{SUA}^a_{\mathcal{D}}$ where each Γ^a_j, $j = 1, \ldots, m_a$, denotes a state equation*

$$state(do(a(\bar{Y}), s), s) \circ St^{-,j}_a =_{St} state(s) \circ St^{+,j}_a$$

Thereby $(a : Obj^k \to Act) \in FUN$, $Y \subseteq X$, $\bar{Y} = Y_1, \ldots, Y_k$ is a sequence of the variable names of Y with $(Y_i : Obj) \in Y$ for $i = 1, \ldots, k$. $St^{-,1}_a$, $St^{+,1}_a$,...,$St^{-,m}_a$, $St^{+,m}_a \in T_{St,\Sigma}(Y)$. Furthermore, let

$$\forall. (\forall (s : Sit). (Poss(a(\bar{Y}), s) \Leftrightarrow \Delta))$$

denote the precondition axiom for action a. We associate a domain $\mathcal{D}^d$ with $\mathcal{D}$ as follows: We remove every pair of precondition axiom and state update axiom $\mathtt{SUA}^a_{\mathcal{D}}$ where the number of disjuncts $m_a > 1$ in $\mathtt{SUA}^a_{\mathcal{D}}$ and add for every $j = 1, \ldots, m_a$ a new pair of precondition axiom and (deterministic) state update axiom:

$$\forall. (\forall (s : Sit). (Poss(a_j(\bar{Y}), s) \Leftrightarrow \Delta)) \tag{2.13}$$

$$\forall. (\forall (s : Sit). (Poss(a_j(\bar{Y}), s) \Rightarrow \Gamma^a_j)) \tag{2.14}$$

Thereby $a_j : Obj^k \to Act$ is a new action for each $j = 1, \ldots, m_a$.

Example 2.8.1 (Non-deterministic domain descriptions) *Consider a signature of $\mathcal{FC}_P$ containing the fluents f, g, h and the action a. Let*

$$\forall (s : Sit). (Poss(a, s) \Leftrightarrow Holds(f, s))$$

be the precondition axiom and

$$\begin{aligned}\forall (s : Sit). (\, Poss(a, s) \Rightarrow (&state(do(a, s)) \circ f =_{St} state(s) \circ g \vee \\ &state(do(a, s)) \circ f =_{St} state(s) \circ h)\,)\end{aligned}$$

the (non-deterministic) state update axiom of the domain description. After applying the transformation of Formula 2.13 the deterministic domain description consists of the precondition axioms

$$\forall (s : Sit). (Poss(a_1, s) \Leftrightarrow Holds(f, s))$$
$$\forall (s : Sit). (Poss(a_2, s) \Leftrightarrow Holds(f, s))$$

and the state update axioms according to Formula 2.14

$$\forall(s : Sit).\,(Poss(a_1, s) \Rightarrow state(do(a_1, s)) \circ f =_{St} state(s) \circ g)$$
$$\forall(s : Sit).\,(Poss(a_2, s) \Rightarrow state(do(a_2, s)) \circ f =_{St} state(s) \circ h)$$

for the new actions a_1 and a_2. While the deterministic domain description has only one model where the state associated with the initial situation consists of one copy of fluent f, the non-deterministic domain description has two.

Models of the non-deterministic domain domain description, where the arrows with label l denote the transition caused by executing action l according to Definition 2.7.1:

$$\begin{tikzcd} state(s0) =_{St} f \arrow[d, "a"] & state(s0) =_{St} f \arrow[d, "a"] \\ state(do(a, s0)) =_{St} g & state(do(a, s0)) =_{St} h \end{tikzcd}$$

Model of the deterministic domain description:

$$\begin{tikzcd} & state(s0) =_{St} f \arrow[dl, "a_1"'] \arrow[dr, "a_2"] & \\ state(do(a_1, s0)) =_{St} g & & state(do(a_2, s0)) =_{St} h \end{tikzcd}$$

Note that in the deterministic case there exists an action sequence leading to a situation s where $\lambda_e = Holds(g, s)$ is true for all initial situations where $\lambda_0 = Holds(f, s0)$ (extended planning problem). In the second model of the non-deterministic case no such action sequence exists. □

The new domain description $\mathcal{D}^d$ is equivalent to the old $\mathcal{D}$ in the following sense:

Proposition 2.8.1 *Let Σ be a Fluent Calculus signature and Y be a variable declaration and $(x : Sit) \in Y$ a variable of sort Sit. Let $\mathcal{D}$ be a non-deterministic domain description wrt Σ which can be transformed by Definition 2.8.1 and $\mathcal{D}^d$ the corresponding result. Let $\lambda_0, \lambda_e \in \Lambda_{L_{(x:Sit),\Sigma}(Y)}$. Then there is a model $\mathcal{M}$ of $\mathcal{D}$, a situation $s^0_{\mathcal{M}} \in Sit^{\mathcal{M}}$ and an action sequence $a_0, \ldots, a_n$ with $a_i \in Act^{\mathcal{M}}$ for $i = 0, \ldots, n$ such that $\mathcal{M} \models \lambda_0(s^0)$ and $\mathcal{M} \models \lambda_e(do(a_n, \ldots do(a_0, s^0) \ldots))$ iff there is a model $\mathcal{M}^d$ of $\mathcal{D}^d$, a situation $s^0_{\mathcal{M}^d} \in Sit^{\mathcal{M}^d}$ and an action sequence $a^d_0, \ldots, a^d_n$ with $a^d_i \in Act^{\mathcal{M}^d}$ such that $\mathcal{M}^d \models \lambda_0(s^0)$ and $\mathcal{M}^d \models \lambda_e(do(a^d_n, \ldots do(a^d_0, s^0) \ldots))$.*

Proof 2.8.1 Since the initial situations in models of $\mathcal{D}$ and models of $\mathcal{D}^d$ are not restricted we may find for every situation $s_{\mathcal{M}}$ of a model $\mathcal{M}$ of $\mathcal{D}$ a model $\mathcal{M}^d$ and a situation $s_{\mathcal{M}^d}$ such that $\mathit{state}^{\mathcal{M}}(s_{\mathcal{M}}) =_{St} \mathit{state}^{\mathcal{M}^d}(s_{\mathcal{M}^d})$ (for example the model with initial situation $s0^{\mathcal{M}^d} = s_{\mathcal{M}^d}$ where per definition $\mathit{state}^{\mathcal{M}}(s_{\mathcal{M}}) =_{St} \mathit{state}^{\mathcal{M}^d}(s0^{\mathcal{M}^d})$). Clearly, the same proposition holds vice versa for every situation in models of $\mathcal{D}^d$.

Let $s_{\mathcal{M}}$ be a situation of some model $\mathcal{M}$ wrt $\mathcal{D}$ and $\mathcal{M}^d$ be a model wrt $\mathcal{D}^d$ with situation $s_{\mathcal{M}^d}$ such that $\mathit{state}^{\mathcal{M}}(s_{\mathcal{M}}) =_{St} \mathit{state}^{\mathcal{M}^d}(s_{\mathcal{M}^d})$. Then, for every situation $s'_{\mathcal{M}} = do^{\mathcal{M}}(a, s_{\mathcal{M}})$ resulting from executing some action a in $s_{\mathcal{M}}$ there exists a situation $s'_{\mathcal{M}^d} = do^{\mathcal{M}^d}(a^d, s_{\mathcal{M}^d})$ in $\mathcal{M}^d$ resulting from executing some action a^d in situation $s_{\mathcal{M}^d}$ such that $\mathit{state}^{\mathcal{M}}(s'_{\mathcal{M}}) =_{St} \mathit{state}^{\mathcal{M}^d}(s'_{\mathcal{M}^d})$ if $\mathit{state}^{\mathcal{M}}(s_{\mathcal{M}}) =_{St} \mathit{state}^{\mathcal{M}^d}(s_{\mathcal{M}^d})$. This is due to the fact that the above construction ensures that for every possible effect Γ_i of some action a in $\mathcal{D}$ exists an action a_i where the corresponding state update axiom in $\mathcal{D}^d$ has precisely the effect Γ_i. Furthermore the precondition axioms describing when the state update axiom for a has to be applied is equivalent to the one for applying a_i. Then if $\mathit{state}^{\mathcal{M}}(s^0_{\mathcal{M}}) =_{St} \mathit{state}^{\mathcal{M}^d}(s^0_{\mathcal{M}^d})$ for situations $s^0_{\mathcal{M}} \in \mathit{Sit}^{\mathcal{M}}$, $s^0_{\mathcal{M}^d} \in \mathit{Sit}^{\mathcal{M}^d}$, by induction over the structure of situations follows that for every situation $s^n_{\mathcal{M}} = do^{\mathcal{M}}(a_n, \ldots do^{\mathcal{M}}(a_0, s^0_{\mathcal{M}}) \ldots)$ in $\mathcal{M}$ exists a situation $s^n_{\mathcal{M}^d}$ in $\mathcal{M}^d$ with $s^n_{\mathcal{M}^d} = do^{\mathcal{M}^d}(a^d_n, \ldots do^{\mathcal{M}^d}(a^d_0, s^0_{\mathcal{M}^d}) \ldots)$ such that $\mathit{state}^{\mathcal{M}}(s^n_{\mathcal{M}}) =_{St} \mathit{state}^{\mathcal{M}^d}(s^n_{\mathcal{M}^d})$. Since λ_0 (λ_e) may only depend on the state associated with some situation it can be either true for both, $s^n_{\mathcal{M}}$ and $s^n_{\mathcal{M}^d}$, or false for both, only.

On the other hand, let $s_{\mathcal{M}^d}$ be a situation of some model $\mathcal{M}^d$ wrt $\mathcal{D}^d$ and $\mathcal{M}$ be a model wrt $\mathcal{D}$ with situation $s_{\mathcal{M}}$ such that $\mathit{state}^{\mathcal{M}^d}(s_{\mathcal{M}^d}) =_{St} \mathit{state}^{\mathcal{M}^d}(s_{\mathcal{M}})$. Then for every situation $s'_{\mathcal{M}^d} = do^{\mathcal{M}^d}(a^d, s_{\mathcal{M}^d})$ resulting from executing some action a^d in $s_{\mathcal{M}^d}$ there exists a model $\mathcal{M}'$ of $\mathcal{D}$ and some situation $s'_{\mathcal{M}'} = do^{\mathcal{M}'}(a, s_{\mathcal{M}'})$ in $\mathcal{M}'$ and $s_{\mathcal{M}'} = do^{\mathcal{M}}(a, s_{\mathcal{M}})$ such that $\mathit{state}^{\mathcal{M}'}(s_{\mathcal{M}'}) =_{St} \mathit{state}^{\mathcal{M}}(s_{\mathcal{M}})$ and $\mathit{state}^{\mathcal{M}'}(s'_{\mathcal{M}'}) =_{St} \mathit{state}^{\mathcal{M}^d}(s'_{\mathcal{M}^d})$. Accordingly if $\mathit{state}^{\mathcal{M}}(s^0_{\mathcal{M}}) =_{St} \mathit{state}^{\mathcal{M}^d}(s^0_{\mathcal{M}^d})$ for situations $s^0_{\mathcal{M}} \in \mathit{Sit}^{\mathcal{M}}$, $s^0_{\mathcal{M}^d} \in \mathit{Sit}^{\mathcal{M}^d}$, by induction follows that for every situation $s^n_{\mathcal{M}^d}$ in $\mathcal{M}^d$ with $s^n_{\mathcal{M}^d} = do^{\mathcal{M}^d}(a^d_n, \ldots do^{\mathcal{M}^d}(a^d_0, s^0_{\mathcal{M}^d}) \ldots)$ there exists a set of models $\mathcal{M}_0, \ldots, \mathcal{M}_n$ where $s^j_{\mathcal{M}_j} = do^{\mathcal{M}_j}(a_j, s^{j-1}_{\mathcal{M}_j})$ and $s^{j-1}_{\mathcal{M}_j} = s^{j-1}_{\mathcal{M}_{j-1}}$ such that $\mathit{state}^{\mathcal{M}_j}(s^j_{\mathcal{M}_j}) =_{St} \mathit{state}^{\mathcal{M}^d}(s^j_{\mathcal{M}^d})$ and $\mathit{state}^{\mathcal{M}_j}(s^{j-1}_{\mathcal{M}_j}) =_{St} \mathit{state}^{\mathcal{M}_{j-1}}(s^{j-1}_{\mathcal{M}_{j-1}})$ for all $1 \leq j \leq n$. Since the effects described by state update axioms only depend on the state associated with the immediately preceding situation, there is also a single model $\mathcal{M}$ of $\mathcal{D}$ with situation $s^n_{\mathcal{M}} = do^{\mathcal{M}}(a^n, \ldots do^{\mathcal{M}}(a^0, s^0_{\mathcal{M}}) \ldots)$ where $\mathit{state}^{\mathcal{M}}(s^j_{\mathcal{M}}) =_{St} \mathit{state}^{\mathcal{M}^d}(s^j_{\mathcal{M}^d})$ for all $0 \leq j \leq n$. □

As a consequence of Proposition 2.8.1 we may reduce planning problems (see Section 2.1) for some non-deterministic domain description (those which can be transformed using Definition 2.8.1) to planning problems for the corresponding deterministic domain description. However, note that the executable action sequences in the deterministic domain description do not reflect the abilities of the agent as specified in the non-deterministic domain description: In the non-deterministic domain the agent cannot choose which of the state equations applies when he executes an action. The choice is "external", i.e. it depends on the world the agent is interacting with. In contrast to this, in the deterministic domain the agent may actually choose the state equation by executing the appropriate action. Hence, an action sequence which is a solution to some planning problem for the deterministic domain is successful in the world only if the choices of the agent correspond with the actual behaviour of the world.

Furthermore, Proposition 2.8.1 cannot be generalised to hold for the extended planning problem of Section 2.1. The solution of the extended planning problem depends on *all* models corresponding to the initial situations fulfilling λ_0. On one hand, every model of the deterministic domain description $\mathcal{D}^t$ contains all possible situations and their associated states that are reachable by some action sequence from the initial situation and, consequently, also those where λ_e is fulfilled (if there are any). On the other hand, states associated with situations by a non-deterministic choice are apparent in some models of $\mathcal{D}$ but not in others, as illustrated by Example 2.8.1.

Disjunctions in Preconditions

If disjunctions are allowed in precondition axioms as described in Section 2.5 a very similar transformation to the one of Definition 2.8.1 can be applied: Consider the domain description contains a precondition axiom of the form

$$\forall.\,(\forall(s:\mathit{Sit}).\,(\mathit{Poss}(a(\bar{Y}),s)\Leftrightarrow\Delta_1\vee\cdots\vee\Delta_n))$$

Thereby $(a:\mathit{Obj}^k\rightarrow\mathit{Act})\in\mathit{FUN}$, $Y\subseteq X$, $\bar{Y}=Y_1,\ldots,Y_k$ is a sequence of the variable names of Y with $(Y_i:\mathit{Obj})\in Y$ for $i=1,\ldots,k$. $\Delta_1,\ldots,\Delta_n$ contain neither explicit nor implicit disjunctions as described in Section 2.5.

If $\mathit{Vars}(\Delta_1),\ldots,\mathit{Vars}(\Delta_n)$ are all disjoint this domain description may be transformed into a domain description without disjunctions in precondition axioms by introducing a new action and corresponding state update and action precondition axioms for each Δ_i: Let $\mathtt{SUA}^a_{\mathcal{D}}$ be a state update axiom of some domain $\mathcal{D}$ wrt some action a. Let Δ of the precondition axiom for a be of the form as defined above and $\mathit{Vars}(\Delta_i)\cap\mathit{Vars}(\Delta_j)=\emptyset$ for all pairs $i,j\in\{1,\ldots,n\}$ and $i\neq j$. Let Γ denote the state equation of $\mathtt{SUA}^a_{\mathcal{D}}$. Then we transform $\mathcal{D}$ into $\mathcal{D}^d$ by removing $\mathtt{SUA}^a_{\mathcal{D}}$ from $\mathcal{D}$ and adding for every $i=1,\ldots,n$ a new pair of precondition and state update axioms

$$\forall.\,(\forall(s:\mathit{Sit}).\,(\mathit{Poss}(a_i(\bar{Y}),s)\Leftrightarrow\Delta_i)) \tag{2.15}$$

$$\forall.\,(\forall(s:\mathit{Sit}).\,(\mathit{Poss}(a_i(\bar{Y}),s)\Rightarrow\Gamma)) \tag{2.16}$$

Thereby $a_i:\mathit{Obj}^k\rightarrow\mathit{Act}$ is a new action for every $i=1,\ldots,n$.

This transformation is less critical than the one from non-deterministic domain descriptions to deterministic domain descriptions, since a state associated with some situation is reachable in $\mathcal{D}$ iff it is a reachable state associated with some situation in $\mathcal{D}^d$. In other words, it is irrelevant which of the preconditions Δ_i enabled for changing the state associated with some situation, i.e. every reachable state of every model of $\mathcal{D}$ has an equivalent state in a corresponding model of $\mathcal{D}^d$ and vice versa.

Example 2.8.2 (Disjunctive preconditions) *Consider a $\mathcal{FC}_P$ signature containing the fluents f, g, h and the action a. Let*

$$\forall (s : Sit).\, (Poss(a, s) \Leftrightarrow Holds(f \circ h, s) \vee Holds(g \circ h, s))$$

be the precondition axiom and

$$\forall (s : Sit).\, (Poss(a, s) \Rightarrow state(do(a, s)) \circ h =_{St} state(s))$$

the state update axiom of the domain description. After applying the above transformation the deterministic domain description consists of the precondition axioms

$$\forall (s : Sit).\, (Poss(a_1, s) \Leftrightarrow Holds(f, s))$$
$$\forall (s : Sit).\, (Poss(a_2, s) \Leftrightarrow Holds(g, s))$$

and the state update axioms

$$\forall (s : Sit).\, (Poss(a, s) \Rightarrow state(do(a, s)) \circ h =_{St} state(s))$$
$$\forall (s : Sit).\, (Poss(a, s) \Rightarrow state(do(a, s)) \circ h =_{St} state(s))$$

for the new actions a_1 and a_2.

Consider the three models of the original domain description where the state associated with the initial situation consists of $f \circ h$, $g \circ h$, and $f \circ g \circ h$, respectively:

$$\begin{tikzcd}
state(s0) =_{St} f \circ h \arrow[d, "a"] & state(s0) =_{St} g \circ h \arrow[d, "a"] & state(s0) =_{St} f \circ g \circ h \arrow[d, "a"] \\
state(do(a, s0)) =_{St} f & state(do(a, s0)) =_{St} g & state(do(a, s0)) =_{St} f \circ g
\end{tikzcd}$$

For each of the above models there exists also a model of the deterministic domain description:

$$\begin{tikzcd} \mathit{state}(s0) =_{St} f \circ h \arrow[d, "a_1"] & \mathit{state}(s0) =_{St} g \circ h \arrow[d, "a_2"] \\ \mathit{state}(\mathit{do}(a_1, s0)) =_{St} f & \mathit{state}(\mathit{do}(a_2, s0)) =_{St} g \end{tikzcd}$$

$$\begin{tikzcd} & \mathit{state}(s0) =_{St} f \circ g \circ h \arrow[dl, "a_1"'] \arrow[dr, "a_2"] & \\ \mathit{state}(\mathit{do}(a_1, s0)) =_{St} f \circ g & & \mathit{state}(\mathit{do}(a_2, s0)) =_{St} f \circ g \end{tikzcd}$$

The only differences between models of the original domain and the models of the corresponding deterministic domain follow from the changed action names. Consequently, if a formula of the query logic does not refer to a particular action name it is valid (satisfiable) in the original domain iff it is valid (satisfiable) in the deterministic domain. This applies, for example, to all planning problems and also all extended planning problems. However note, that we can not establish bisimularity. □

The Simple Fluent Calculus with Specificity

In [70] the simple Fluent Calculus has been extended to address the following representational issue: Even if an action description in the simple Fluent Calculus proposes valid effects in general and incorrect effects for only few exceptional situations, the whole action description has to be modified to cope with these minor cases. Instead [70] proposes the introduction of a *specificity relation* over pairs of precondition and state update axioms. This specificity relation enables for incremental development of domain descriptions: First, the effects of an action can be specified for a large set of situations. Later, if for some subsets of these situations the specification turns out to be incorrect, a new action description that "overrules" the old one in these situations can simply be added to the domain description.

In the following we call the Fluent Calculus of [70] *simple Fluent Calculus with specificity*, abbreviated as $\mathcal{FC}_{L\leq}$. Instead of presenting $\mathcal{FC}_{L\leq}$ formally as a logic programming scheme as it was introduced in [70] we will define its syntax and semantics in our calculus-independent framework. Then we will demonstrate how $\mathcal{FC}_{LN}$ domain descriptions may be seen as generalisations of $\mathcal{FC}_{L\leq}$ if the associated transition systems are considered. Note however, that $\mathcal{FC}_{LN}$ does not solve the above representational problem addressed by the simple Fluent Calculus with specificity.

Definition 2.8.2 *Given a domain description $\mathcal{D}$ for some $\mathcal{FC}$ signature $\Sigma = (SORT, \preceq, FUN, REL)$ and X a variable declaration wrt Σ. $\mathcal{D}$ is a $\mathcal{FC}_{L\leq}$ domain iff for every action a exists a set $\mathrm{P}^a_{\mathcal{D}}$ of pairs ϕ_j, $j = 1, \ldots, n$ consisting of a state update axiom and a precondition axiom of the form*

$$\begin{aligned}&\forall\bar{Y}.\,(\forall(s : Sit).\,(Poss(a(\bar{Y}), s) \Leftrightarrow Holds(St^{-,j}_a, s)))\\&\forall\bar{Y}.\,(\forall(s : Sit).\,(Poss(a(\bar{Y}), s) \Rightarrow\\&\qquad\qquad state(do(a(\bar{Y}), s), s) \circ St^{-,j}_a =_{St} state(s) \circ St^{+,j}_a))\end{aligned}$$

where $(a : Obj^k \rightarrow Act) \in FUN$, $Y \subseteq X$, $\bar{Y} = Y_1, \ldots, Y_k$ is a sequence of the variable names of Y with $(Y_i : Obj) \in Y$ for $i = 1, \ldots, k$. $St^{-,j}_a, St^{+,j}_a \in T_{St,\Sigma}(Y)$ for all $j = 1, \ldots, n$.

Thereby, the valid pair of axioms of $\mathrm{P}^a_{\mathcal{D}}$ is chosen wrt some situation s of a model $\mathcal{M}$. To this end, let $St^-_a(\phi)$ denote the term St^-_a of axiom pair $\phi \in \mathrm{P}^a_{\mathcal{D}}$. Then ϕ is valid in s iff

1. *$\mathcal{M} \models Holds(St^-_a(\phi), s)\sigma$ for some substitution σ, and*
2. *there is no $\phi' \in \mathrm{P}^a_{\mathcal{D}}$ such that $\mathcal{M} \models Holds(St^-_a(\phi'), s)\sigma'$ for some substitution σ' and $St^-_a(\phi)\sigma \circ t =_{St} St^-_a(\phi')\sigma'$ where $t \neq_{St} 1^\circ$.*

Example 2.8.3 ($\mathcal{FC}_{L\leq}$ domain descriptions) *Consider the signature and the state update axioms of Example 2.5.1. Instead of restricting the executability of action mv_brick(x_1, y_1, x_2, y_2) as in Example 2.5.3, we may overrule the pair of precondition and state update axiom by a more specific one which can be applied if the location (x_2, y_2) is not empty:*

$$\begin{aligned}&\forall(x_1, x_2 : XPos), (y_1, y_2 : YPos), (s : Sit).\\&\qquad(\,Poss(mv\_brick(x_1, y_1, x_2, y_2), s) \Leftrightarrow\\&\qquad\;Holds(brick\_at(x_1, y_1) \circ brick\_at(x_2, y_2), s)\,).\end{aligned}$$

We may specify the state associated with the succeeding situation to be unchanged, e.g. in this case

$$\begin{aligned}&\forall(x_1, x_2 : XPos), (y_1, y_2 : YPos), (s : Sit).\\&\qquad(\,Poss(mv\_brick(x_1, y_1, x_2, y_2), s) \Rightarrow\\&\qquad\;state(do(mv\_brick(x_1, y_1, x_2, y_2), s)) =_{St} state(s)\,).\end{aligned}$$

□

Note that by Definition 2.8.2 we may imagine a domain description and a situation where more than one pair of precondition and state update axiom is valid. This causes contradictions if the state equations of the valid state update axioms are not equivalent. To prevent such contradictions it should be ensured by the design of the domain description that the chosen precondition axiom is unique in all considered situations. To design domain descriptions in such a way can be difficult and this problem is also referred to by the *qualification problem*, see [106].

Domain descriptions with a specificity relation may be translated into domain descriptions without a specificity relation requiring negation in action preconditions. Thereby the basic idea is to rewrite each pair ϕ of precondition and state update axiom in such a way that the precondition axiom reflects both the condition of applying ϕ and all the exceptions where another action description "overrules" ϕ. To this end we consider for all pairs of elements ϕ_j, ϕ_i $(j \neq i)$ of $\mathrm{P}^a_{\mathcal{D}}$ the complete set $\{\sigma^i_1, \ldots, \sigma^i_{m_i}\}$ of unifiers for $St^-_a(\phi_i) =_{St} St^-_a(\phi_j) \circ v$. Since AC1 is finitary (by Proposition 2.2.1) this set is always finite. However, it has to be ensured that we only consider those substitutions where $St^-_a(\phi_i)$ is a genuine sub-multiset of $St(\phi_j) \circ v$: If some substitution σ of the complete set of unifiers substitutes v by 1° then σ must not be considered and hence has to be removed. Otherwise, if σ substitutes v by some other variable x then σ has to be removed and instead a set

$$\{(\sigma \setminus \{v \mapsto x\}) \cup \{v \mapsto f_i(x_i) \circ y_i\} \mid (f_i : \vec{T} \to Fl) \in FUN \text{ with fresh } x_i, y_i\}$$

of unifiers has to be added. Since there is only a finite number of fluent symbols in a Fluent Calculus signature the set of unifiers is still finite. Then, using the resulting substitutions $\{\sigma'^i_1, \ldots, \sigma'^i_{m_i}\}$ for all $i \neq j$, we have a new axiom of the following form for each $\phi_j \in \mathrm{P}^a_{\mathcal{D}}$:

$$\begin{aligned}
\forall.\, (\forall(s : Sit).\, (&Poss(a(\bar{Y}), s) \Leftarrow \\
&Holds(g(\phi_j), s) \wedge \\
&\neg Holds(g(\phi_1)\sigma'^1_1, s) \wedge \cdots \wedge \neg Holds(g(\phi_1)\sigma'^1_{m_1}, s) \wedge \\
&\qquad\vdots \\
&\neg Holds(g(\phi_{j-1})\sigma'^{j-1}_1, s) \wedge \cdots \wedge \neg Holds(g(\phi_{j-1})\sigma'^{j-1}_{m_{j-1}}, s) \wedge \\
&\neg Holds(g(\phi_{j+1})\sigma'^{j+1}_1, s) \wedge \cdots \wedge \neg Holds(g(\phi_{j+1})\sigma'^{j+1}_{m_{j+1}}, s) \wedge \\
&\qquad\vdots \\
&\neg Holds(g(\phi_n)\sigma'^n_1, s) \wedge \cdots \wedge \neg Holds(g(\phi_n)\sigma'^n_{m_n}, s)\,)
\end{aligned} \tag{2.17}$$

These axioms can be rewritten in the form as defined for the fragment $\mathcal{FC}_{LN}$ if we ensure that there is only one precondition axiom for each action a in the domain. This can be achieved by substituting the term $a(\bar{Y})$ in the Axiom 2.17 for the pair ϕ_j with $a_j(\bar{Y})$ for the new action $a_j : Obj^k \to Act$ (see the previous subsection for details on the effect of such a transformation). Then, by the completion semantics of logic programs with negation (as used in [70]) we have to substitute in the Axiom 2.17 $\Leftarrow$ by $\Leftrightarrow$ to represent it using the semantics of Section 2.3. The resulting axiom is the precondition axiom for action a_j in the new $\mathcal{FC}_{LN}$ domain. The new state update axiom is simply defined as

$$\begin{aligned}
\forall \bar{Y}.\, (\forall(s : Sit).\, (&Poss(a_j(\bar{Y}), s) \Rightarrow \\
&state(do(a_j(\bar{Y}), s), s) \circ St^{-,j}_a =_{St} state(s) \circ St^{+,j}_a))
\end{aligned}$$

Example 2.8.4 ($\mathcal{FC}_{L\leq}$ domain descriptions continued) *The following two pairs of precondition and state update axioms are result of translating the $\mathcal{FC}_{L\leq}$ domain of Example 2.8.3 into $\mathcal{FC}_{LN}$ according to the above procedure:*

$$\begin{array}{l} \forall(s : \mathit{Sit}), (x_1, x_2 : \mathit{XPos}), (y_1, y_2 : \mathit{YPos}). \\ \qquad (\,\mathit{Poss}(\mathit{mv\_brick}_1(x_1, y_1, x_2, y_2), s) \Leftrightarrow \\ \qquad \mathit{Holds}(\mathit{brick\_at}(x_1, y_1), s) \wedge \\ \qquad \neg\mathit{Holds}(\mathit{brick\_at}(x_1, y_1) \circ \mathit{brick\_at}(x_2, y_2), s)\,) \\ \forall(s : \mathit{Sit}), (x_1, x_2 : \mathit{XPos}), (y_1, y_2 : \mathit{YPos}). \\ \qquad (\,\mathit{Poss}(\mathit{mv\_brick}_2(x_1, y_1, x_2, y_2), s) \Leftrightarrow \\ \qquad \mathit{Holds}(\mathit{brick\_at}(x_1, y_1) \circ \mathit{brick\_at}(x_2, y_2), s)\,) \end{array}$$

$$\begin{array}{l} \forall(s : \mathit{Sit}), (x_1, x_2 : \mathit{XPos}), (y_1, y_2 : \mathit{YPos}). \\ \qquad (\,\mathit{Poss}(\mathit{mv\_brick}_1(x_1, y_1, x_2, y_2), s) \Rightarrow \\ \qquad \mathit{state}(\mathit{do}(\mathit{mv\_brick}(x_1, y_1, x_2, y_2), s)) \circ \mathit{brick\_at}(x_1, y_1) \\ \qquad =_{St} \mathit{state}(s) \circ \mathit{brick\_at}(x_2, y_2)\,) \\ \forall(s : \mathit{Sit}), (x_1, x_2 : \mathit{XPos}), (y_1, y_2 : \mathit{YPos}). \\ \qquad (\,\mathit{Poss}(\mathit{mv\_brick}_2(x_1, y_1, x_2, y_2), s) \Rightarrow \\ \qquad \mathit{state}(\mathit{do}(\mathit{mv\_brick}(x_1, y_1, x_2, y_2), s)) =_{St} \mathit{state}(s)\,) \end{array}$$

□

As a consequence of the translation described above, in terms of the associated transition systems $\mathcal{FC}_{LN}$ domains are at least as expressive as $\mathcal{FC}_{L\leq}$ domains. In fact, the possibility to use negation in precondition axioms of $\mathcal{FC}_{LN}$ enables restricting not only the applicability of a particular state update axiom but also the executability of some action. According to [70] in $\mathcal{FC}_{L\leq}$ an action is executable wrt some situation s iff there is some pair ϕ of precondition and state update axiom such that its positive precondition is fulfilled in s. Instead, in $\mathcal{FC}_{LN}$ an action is executable in some situation s iff there is some precondition axiom ϕ such that its positive *and* negative preconditions are fulfilled in s.

Example 2.8.5 ($\mathcal{FC}_{LN}$ vs. $\mathcal{FC}_{L\leq}$) *Let a set of transition systems be defined by the following $\mathcal{FC}_{LN}$ domain description:*

$$\forall(s : \mathit{Sit}).\,(\mathit{Poss}(a, s) \Leftrightarrow \mathit{Holds}(f, s) \wedge \neg\mathit{Holds}(f \circ f, s)).$$

$$\forall(s : \mathit{Sit}).\,(\mathit{Poss}(a, s) \Rightarrow \mathit{state}(\mathit{do}(a, s)) =_{St} \mathit{state}(s) \circ f).$$

We may depict the only model $\mathcal{M}$ of this domain where the initial situation has a successor situation (as defined by $\leq$) as follows:

$$\begin{tikzcd} \mathit{state}(s0) =_{St} f \arrow[d, "a"] \\ \mathit{state}(\mathit{do}(a, s0)) =_{St} f \circ f \end{tikzcd}$$

This set of transition systems is not definable in $\mathcal{FC}_{L\leq}$. This is due to the fact that every action a that can be executed in s0 and which leads to a state state(do(a, s0)) containing state(s0) as a sub-multiset must be also executable in do(a, s0). □

2.9 Summary

In this chapter we have introduced the concepts of representing action and change in the Fluent Calculus. Our introduction is based on the most important historical roots of the Fluent Calculus: the Situation Calculus and the frame problem. We have also introduced the reasoning problems we want to focus on in this thesis. However, the approach we propose for analysis of Fluent Calculus domains is not limited to these problems.

Our formal presentation of the Fluent Calculus is very similar to the presentations given in recent work by other authors [150, 67] for the Fluent Calculus and [145, 119] for the Situation Calculus. According to these approaches the language used to specify a domain may be different from the language used to specify the reasoning problem. The domain specification consists of domain independent axioms and domain dependent axioms. Different fragments of the Fluent Calculus are defined by the structure of the domain dependent axioms. The fragments we have defined are inspired by previous work on representational issues, adapted to our framework. However, our approach for analysing Fluent Calculus domains is not limited to these fragments.

Our main contribution in this chapter is a general approach of associating transition systems to models of Fluent Calculus domains. So far, this has been done only for specific domains. The most important achievement of the general approach is the generic applicability of numerous results from the field of model checking to investigate decidability questions in the Fluent Calculus (see Chapter 5). However, the approach allows us also to compare the expressiveness of different Fluent Calculus fragments. This idea has proven fruitful as we achieved several new results in this chapter. We showed that under some circumstances non-deterministic domains can be transformed into deterministic ones and that solutions to the planning problem are (but solutions to the extended planning problem are not) invariant under this transformation. Similarly, we showed that under some circumstances domains with disjunctive precondition axioms can be transformed into domains without such axioms and that solutions to the planning problem and the extended planning problem are invariant under this transformation. Finally, we showed that $\mathcal{FC}_{LN}$ is strictly more expressive than $\mathcal{FC}_{L\leq}$.

Part II

Decidability of Reasoning

Chapter 3

Temporal Properties in the Fluent Calculus

In this section we discuss our approach for developing reasoning methods for the Fluent Calculus. At the same time, the section provides an overview of the second part of the thesis.

3.1 Reasoning by Model Checking

As mentioned before, the separation of an intelligent system into heuristic and epistemological components has been justified by introspection. Furthermore, it allows us to develop methods for each of the components independently. However, this separation can cause problems. Knowledge that is stored in the internal representation might have a strong influence on the reasoning process. But this knowledge can not control the reasoning appropriately, since the reasoning component is developed independently. On the other hand, we may "hardwire" the knowledge needed for control into the reasoning mechanism and thereby restrict the applicability of the heuristic component to previously defined domains. Then, instead of aiming to develop a single reasoning mechanism for all domains, we may develop several specialised sub-mechanisms such that each of them performs well only for a particular domain. Although none of these sub-mechanisms is powerful enough to solve every problem, all sub-mechanisms together might perform adequately[1] for a wide range of problems even if the represented knowledge as a whole is not structured in a particular way.

A first sensible step to develop such specialised sub-mechanisms is to investigate the general possibility for automatic reasoning in particularly interesting domains. In other words, we investigate whether the satisfiability/validity of

[1]Here we mean "adequate" in the sense of [12].

queries about some domain description can be *decided* effectively by a computer. Of course, one could argue that even if a certain query were decidable it might be impossible to compute the answer *efficiently*. However, decidability is a first issue and future models of computing, e.g. quantum computing, might change today's view of what is actually feasible.

So far we have defined a number of Fluent Calculus fragments of different syntactic expressiveness as depicted in Figure 2.1. We have also defined a number of problem classes which we wish to solve automatically in these fragments, see Figure 2.2. To investigate decidability of these tasks we aim to apply results of model checking[2] and as mentioned in Section 2.7. There we have also already demonstrated how to view models of Fluent Calculus domains as transition systems. This enables us to interpret formulas of modal/temporal logics, i.e. the common languages in the area of model checking, over models of Fluent Calculus domains. What is missing is the characterisation of formulas of the query logic by formulas of modal/temporal logics. Such a characterisation will allow us to reduce decidability questions for Fluent Calculus domains directly to decidability questions in model checking. To this end we show in the following sections that the query logic of Definition 2.6.3 corresponds to the monadic second order logic over trees. Formulas of monadic second order logic that are invariant under bisimulation can be characterised by the propositional μ-calculus. Smaller fragments of the bisimulation invariant monadic second order logic over trees correspond to other well known temporal logics. The relations we will establish between these modal/temporal logics and the query logic for $\mathcal{FC}$ will not only enable us to apply model checking results, they will also allow us to specify many important system properties in the simpler modal/temporal logics. Since these logics are highly specialised their formulas are much easier to understand than the corresponding formulas of the more general $\mathcal{FC}$ query logic. Hence, after having shown the correspondences formally in the next sections we will represent query logic formulas by their modal/temporal logic counterparts.

After succeeding in describing the considered reasoning tasks as problems of model checking, we need to investigate what results we may apply. To this end some common properties of most Fluent Calculus domains can guide our search. The most important properties follow from the fact that states are represented as commutative monoids (i.e. $\circ$ can be read as the *multiset union*). Since the set of multisets even wrt. a finite alphabet is infinite, the set of states associated with a model of a Fluent Calculus domain is in general infinite as well. Model checking of infinite systems is inherently difficult and the interest in the field has recently grown. Despite of the difficulties some classes of systems have been studied, e.g. systems characterised by *context-free processes*, *Petri nets*, *process algebras*, and *well-structured transition systems*. Among these, Petri nets are particularly promising candidates for establishing relations with Fluent Calculus domains. This follows from the fact that their states are defined by multisets

[2]Although model checking is a well-known area of research we are not aware of a similar attempt to reduce reasoning about action and change *explicitly* to model checking problems. However, in [67] and in particular in [145] model checking results are applied implicitly.

as well. Hence, in Chapter 7 we define Petri nets formally and present a well known reasoning method based on the construction of the *Karp-Miller tree*. Furthermore, we show how the Fluent Calculus fragments $\mathcal{FC}_{PL}$ and $\mathcal{FC}_P$ relate to Petri nets. The relations are established by means of bisimulation. Since all modal/temporal properties we consider here are invariant under bisimulation this provides a very tight correspondence. Finally in Chapter 5, we exploit this correspondence by investigating its consequences for decidability of the reasoning tasks of Figure 2.2. For the investigation of some of the tasks we will additionally consider classical models of computations like *counter machines.*

In this work we show that some reasoning task is actually decidable for some class of Fluent Calculus domains by reducing the task to a problem for which a decision procedure has been already provided. Such a decision procedure together with the reduction procedure is a calculus for the particular reasoning task of the Fluent Calculus. However, by applying the knowledge we gained from our investigation of decidability we may attempt to design a new decision procedure that can be applied directly to Fluent Calculus domains. We may hope that this new procedure is simpler as it does not require an additional reduction process. The development of such a specialised reasoning procedure is the goal of the third part of the work. Thereby we restrict ourselves to the development of a decision procedure for conjunctive planning problems in $\mathcal{FC}_{PL}$.

3.2 Monadic Second-Order Queries

A second order logic where quantification over predicates is restricted to those having arity 1 is usually called *monadic second order logic (MSOL)*, e.g. [126]. A predicate of arity 1 represents a set of individuals. Consequently, quantification over unary predicates may be replaced by quantification over sets of individuals and the use of the relation $\in$ between individuals and sets of individuals.

If all individuals are interpreted as nodes of a tree and the signature of the *MSOL* contains special predicates R to denote ancestor and successor relations and a set P of unary first-order predicates then we denote the corresponding *MSOL* by $MSOL(R, P)$. For example, let $K(\mathcal{M}, L) = (Sit^{\mathcal{M}}, \rightarrow_{\mathcal{M}}, Act^{\mathcal{M}}, \alpha_{\mathcal{M}})$ denote the L-valued transition system associated with some model $\mathcal{M}$ of a Fluent Calculus domain $\mathcal{D}$ with signature Σ. Then $MSOL(\{\leq^{\mathcal{M}}\} \cup \{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}, L)$ denotes the monadic second-order logic over the tree defined by $K(\mathcal{M}, L)$ with the ancestor relation $\leq^{\mathcal{M}}$, the successor relations $\{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}$ and the set L of unary first-order predicates.

Now we show that the query language of Definition 2.6.3 can be viewed as a monadic second-order logic. This helps us to apply some results relating (fragments of) monadic second order logic to well known modal/temporal logics. Formulas of these logics may then be used to characterise formulas of the query logic.

Proposition 3.2.1 (Monadic Second Order Logic) *Given a domain description $\mathcal{D}$ for some Fluent Calculus signature Σ, X some variable declaration X wrt Σ, Y some variable declaration of the form $(y : \mathcal{B})$, and $L \subseteq F_{(x:Sit),\Sigma}(X)$. Furthermore, let $\mathcal{M}$ be a model of $\mathcal{D}$ and $\mathcal{B}^{\mathcal{M}} = \mathcal{P}(Sit^{\mathcal{M}})$. Then, the query logic over Σ wrt X and Y and L is the monadic second order logic $MSOL(\{\leq^{\mathcal{M}}\} \cup \{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}, L)$.*

Proof 3.2.1 As has been shown in Lemma 2.7.1, $K(\mathcal{M}, L)$ defines a L–valued transition system with root $s0^{\mathcal{M}}$. $K(\mathcal{M}, L)$ represents a tree with the nodes $Sit^{\mathcal{M}}$. For all propositions $p \in L$, the value of p only depends on the instantiation of some variable of sort Sit. Furthermore, according to Definition 2.6.3, all variables in formulas of the query logic are either bound in a formula of L or they are interpreted as elements of $Sit^{\mathcal{M}}$ or $\mathcal{P}(Sit^{\mathcal{M}})$, respectively. □

The most expressive propositional modal logic that is considered for model checking in the literature is the *propositional μ-calculus* [80]. The great expressive power is a result of the use of an operator ν representing the greatest fixpoint of some monotonic functional describing the behaviour of a system in a particular state. For example, let p denote some modal property and a a transition. By $\nu x.\,(p \wedge \langle a \rangle x)$ we describe the greatest set of states such that the modal property p is invariant under the transition labelled a.

Definition 3.2.1 *Let P be a set of propositions and A a set of transitions. Then the* propositional μ-calculus *wrt P and A is defined as follows. The set of formulas of the propositional μ-calculus is the smallest set L containing P, X, $\neg\phi$, $\phi \wedge \psi$, $\langle a \rangle \phi$, $\nu x.\phi$ where X is a set of variables, ϕ, ψ are formulas in L, $a \in A$.*

Let $\Theta = (S, \rightarrow, A, \alpha)$ be a P-valued transition system. A valuation *$\mathcal{V}$ assigns to each variable $x \in X$ a subset of S. Then we define a mapping $||\text{-}||^{\Theta}_{\mathcal{V}}$ from formulas of L to sets of states as follows:*

$$\begin{aligned}
||p||^{\Theta}_{\mathcal{V}} &= \{s \mid s \in S \wedge \alpha(p, s) = \top\} \\
||\neg\phi||^{\Theta}_{\mathcal{V}} &= S \setminus ||\phi||^{\Theta}_{\mathcal{V}} \\
||\phi \wedge \psi||^{\Theta}_{\mathcal{V}} &= ||\phi||^{\Theta}_{\mathcal{V}} \cap ||\psi||^{\Theta}_{\mathcal{V}} \\
||\langle a \rangle \phi||^{\Theta}_{\mathcal{V}} &= \{s \mid \exists s' \in S.\, s \xrightarrow{a} s' \wedge s' \in ||\phi||^{\Theta}_{\mathcal{V}}\} \\
||\nu x.\phi||^{\Theta}_{\mathcal{V}} &= \bigcup\{R \subseteq S \mid R \subseteq ||\phi||^{\Theta}_{\mathcal{V}[x/S]}\}
\end{aligned}$$

A closed formula ϕ is valid *in Θ, $\Theta \models \phi$, iff for all states s of Θ, $s \in ||\phi||^{\Theta}$.*

By showing that every formula of the propositional μ-calculus corresponds to some formula of the query logic we demonstrate the expressive power of the query logic. Furthermore, we may express reasoning tasks like the planning problem as formulas of the propositional μ-calculus.

Note also that the propositional μ-calculus respects bisimularity, i.e. if a formula is valid for some transition system it is also valid in all bisimilar systems. Assume we want to prove the validity of some formula ϕ of the query logic for the transition system $K(\mathcal{M}, L)$ associated with some model $\mathcal{M}$ of some Fluent Calculus domain description. To this end we may prove the validity of some formula ϕ' of the propositional μ-calculus which characterises ϕ for any transition system that is bisimilar to $K(\mathcal{M}, L)$.

Corollary 3.2.2 *Given a model $\mathcal{M}$ of some domain description of the Fluent Calculus with signature Σ, X some variable declaration X wrt Σ, Y some variable declaration of the form $(y : \mathcal{B})$. For every closed formula ϕ of the propositional μ-calculus wrt some $L \subseteq F_{(x:Sit),\Sigma}(X)$ and $Act^{\mathcal{M}}$ there exists a formula ϕ' in the query logic over Σ wrt X and Y and L, such that $K(\mathcal{M}, L) \models \phi$ iff $\mathcal{M} \models \phi'$.*

Proof 3.2.2 From Proposition 3.2.1 follows that the above query logic is the monadic second order logic *MSOL*$(\{\leq^{\mathcal{M}}\} \cup \{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}, L)$. In [73] it has been shown that for every class $\mathcal{C}$ of transition systems definable by some closed formula ϕ of the μ-calculus there exists a *MSOL*(R, P) formula ϕ' defining $\mathcal{C}$, where R denotes denotes an ancestor and a set of successor relations and P a set of unary first-order predicates. Consider $K(\mathcal{M}, L) \models \phi$. Then, with $R = \{\leq^{\mathcal{M}}\} \cup \{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}$ and $P = L$ holds $\mathcal{M} \models \phi'$. On the other hand, clearly, if $\mathcal{M} \models \phi'$, then $K(\mathcal{M}, L) \in \mathcal{C}$. □

Note that it can also be shown using [73] that for every formula of the bisimulation invariant fragment of the query logic of Proposition 3.2.1 there exists an equivalent formula in the propositional μ-calculus.

3.3 Monadic Path Queries

The interpretation of sets of individuals in a monadic second-order logic *MSOL*(R, P) may be restricted to those which form branches of the tree. The corresponding logic is called *monadic path logic* [57], denoted *MPL*(R, P). Since the notion of a branch can be defined in *MSOL*(R, P) the monadic path logic *MPL*(R, P) is a fragment of *MSOL*(R, P).

As a direct consequence of Proposition 3.2.1 and the restriction of $\mathcal{B}^{\mathcal{M}}$ to branches follows:

Corollary 3.3.1 (Monadic Path Logic) *Given a domain description $\mathcal{D}$ for the Fluent Calculus with signature Σ, X some variable declaration wrt Σ and Y a variable declaration of the form $(y : \mathcal{B})$ and $L \subseteq F_{(x:Sit),\Sigma}(X)$. Furthermore, let $\mathcal{M}$ be a model of $\mathcal{D}$ and $\mathcal{B}^{\mathcal{M}}$ be the set of all branches in $K(\mathcal{M}, L)$. Then, the query logic over Σ wrt X and Y and L is the monadic path logic MPL$(\{\leq^{\mathcal{M}}\} \cup \{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}, L)$.*

Another popular temporal logic used for model checking is CTL^* [33], also called *full branching temporal time logic.* It is strictly less expressive than the propositional μ-calculus. However, formulas of CTL^* are usually much easier to understand than their μ-calculus counterparts since the definition of the semantics of CTL^* formulas does not require the notion of fixpoint operators.

Here we present CTL^* as in [112] considering finite and infinite runs. However, since in $\mathcal{FC}$ systems are described in an action oriented style, we consider a relativised next operator (a) instead of the unrelativised X.

Definition 3.3.1 *Let A be a set of transitions and P a set of propositions. Then the computational tree logic CTL^* wrt α and A is defined as follows. The set of formulas of CTL^* is the smallest set containing P, $\neg\phi$, $\phi\wedge\psi$, $(a)\phi$, $\phi U\psi$, $E\phi$, where ϕ, ψ are formulas and $a \in A$.*

Let $\Theta = (S, \rightarrow, A, \alpha)$ be a P-valued transition system, t a run and s a state in Θ. Then

1. $\Theta, s \models p$ *iff* $\alpha(p, s) = \top$
2. $\Theta, s \models (\neg\phi)$ *iff* $\Theta, s \not\models \phi$
3. $\Theta, s \models (\phi \wedge \psi)$ *iff* $\Theta, s \models \phi$ *and* $\Theta, s \models \psi$
4. $\Theta, s \models E\phi$ *iff there is a run t' such that $s = t'_0$ and* $\Theta, t' \models \phi$

5. $\Theta, t \models p$ *iff* $\alpha(p, t_0) = \top$
6. $\Theta, t \models (\neg\phi)$ *iff* $\Theta, t \not\models \phi$
7. $\Theta, t \models (\phi \wedge \psi)$ *iff* $\Theta, t \models \phi$ *and* $\Theta, t \models \psi$
8. $\Theta, t \models (a)\phi$ *iff* $\Theta, t_{\geq 1} \models \phi$ *and* $t_0 \xrightarrow{a} t_1$,
9. $\Theta, t \models \phi U\psi$ *iff there is some $i \geq 0$ such that $\Theta, t_{\geq i} \models \psi$ and for each j, if $0 \leq j < i$ then $\Theta, t_{\geq j} \models \phi$*

A formula ϕ is valid *in Θ, $\Theta \models \phi$, iff for all states s of Θ, $\Theta, s \models \phi$. ϕ is* satisfiable *in Θ iff there is a state s of Θ, such that $\Theta, s \models \phi$.*

The following abbreviations will be used: $F\phi \overset{\text{def}}{=} \top U\phi$, $G\phi \overset{\text{def}}{=} (\neg F(\neg\phi))$, $A\phi \overset{\text{def}}{=} (\neg E(\neg\phi))$.

Just as formulas of the propositional μ-calculus correspond to formulas of $MSOL$, CTL^* formulas correspond to formulas of the monadic path logic. Consequently, we may use CTL^* to characterise a weaker fragment of the query logic.

Corollary 3.3.2 (CTL^*) *Given a model $\mathcal{M}$ of some domain description of the Fluent Calculus with signature Σ, X a variable declaration wrt Σ, $L \subseteq F_{(x:Sit),\Sigma}(X)$. For every formula ϕ of the temporal logic CTL^* wrt L and $Act^{\mathcal{M}}$ there exists a formula ϕ' in the query logic of Corollary 3.3.1, such that $K(\mathcal{M}, L), s0^{\mathcal{M}} \models \phi$ iff $\mathcal{M} \models \phi'$.*

Proof 3.3.1 In [57] it has been shown, that for every finitely branching tree t with root ϵ and for every CTL^* formula ϕ exists a formula ϕ' in $MPL(\{\leq^{\mathcal{M}}\} \cup \{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}, L)$, such that $t, \epsilon \models \phi$ iff $t \models \phi'$. This result has been generalised to include infinitely branching trees in [112]. Since $K(\mathcal{M}, L)$ represents a tree with root $s0^{\mathcal{M}}$ and with Corollary 3.3.1, $K(\mathcal{M}, L), s0^{\mathcal{M}} \models \phi$ iff $\mathcal{M} \models \phi'$. □

Again, it can be shown using the expressive completeness result of [112] that for every formula of the bisimulation invariant fragment of the query logic of Corollary 3.3.1 there exists an equivalent formula in CTL^*.

3.4 CTL_U Queries

If quantification over sets of individuals or predicates, respectively, in $MSOL$ is not allowed, the resulting logic is a first-order logic, called *first-order logic over trees*, denoted $FOL(R, P)$, where P and R are defined as for the general $MSOL$. Since $FOL(\{\leq^{\mathcal{M}}\} \cup \{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}, L)$ is a fragment of $MSOL(\{\leq^{\mathcal{M}}\} \cup \{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}, L)$ we can follow from the restriction to first-order predicates and Proposition 3.2.1:

Corollary 3.4.1 (First Order Logic) *Given a domain description $\mathcal{D}$ for the Fluent Calculus with signature Σ, X a variable declaration wrt Σ and $Y = \emptyset$, $L \subseteq F_{(x:Sit),\Sigma}(X)$. Furthermore, let $\mathcal{M}$ be a model of $\mathcal{D}$. Then, the query logic over Σ wrt X and Y is the first order logic over trees denoted by $FOL(\{\leq^{\mathcal{M}}\} \cup \{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}, L)$.*

Since the notion of a run is not definable in first order logic (as the set of runs is not enumerable), it is difficult to compare branching time temporal logics like CTL^* to FOL. However, some interesting formulas in CTL^* do not require to distinguish between runs. In particular, both the planning problem and the extended planning problem can be stated as satisfiability and validity, respectively, of CTL^* formulas that do not require the notion of a run. In the following we define an appropriate fragment of CTL^*, or more precisely of CTL [33]. We call this fragment CTL_U as the use of the operator U is restricted.

Definition 3.4.1 (CTL_U) *Let A be a set of transitions and P a set of propositions. Then the logic CTL_U wrt A and P is defined as follows. The set of* *formulas of CTL_U is the smallest set containing P, $\neg\phi$, $\phi \wedge \psi$, $E(a)\phi$, $E\phi U\psi$, where ϕ, ψ are formulas and $a \in A$.*

The semantics of the operators is defined as in Definition 3.3.1.

To show that CTL_U formulas indeed characterise some FOL formulas we define the following mapping:

Definition 3.4.2 *We define Λ, mapping CTL_U formulas and atomic propositions of $p \in L$ with $L \subseteq F_{(x:Sit),\Sigma}(X)$ to formulas of $FOL(\{\leq^{\mathcal{M}}\} \cup \{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}, L)$ as follows.*

$$\begin{aligned}
\Phi(p,s) &\equiv p\{x \mapsto s\},\\
\Phi(\neg\phi,s) &\equiv \neg\Phi(\phi,s),\\
\Phi(\phi_1 \wedge \phi_2, s) &\equiv \Phi(\phi_1,s) \wedge \Phi(\phi_2,s),\\
\Phi(E(a)\phi,s) &\equiv \exists(s' : Sit).\,(s \xrightarrow{a} s' \wedge \Phi(\phi,s')),\\
\Phi(E\phi U\psi,s) &\equiv \exists(s' : Sit).\,(s \leq s' \wedge \Phi(\psi,s') \wedge\\
&\qquad \forall(s'' : Sit).\,(s \leq s'' \wedge s'' < s' \Rightarrow \Phi(\phi,s''))).
\end{aligned}$$

The following proposition enables for characterisation of some first order formulas of the query logic by CTL_U. Thereby the proposition gives a hint to the expressive power of the (first-order) query logic originally proposed for Fluent Calculus and Situation Calculus domains [109].

Proposition 3.4.2 *Given a model $\mathcal{M}$ of some domain description of some Fluent Calculus signature Σ, X a variable declaration wrt Σ, $L \subseteq F_{(x:Sit),\Sigma}(X)$. For every formula ϕ of the temporal logic CTL_U wrt L and $Act^{\mathcal{M}}$ there exists a formula ϕ' in the logic $FOL(\{\leq^{\mathcal{M}}\} \cup \{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}, L)$, such that $K(\mathcal{M},L), s0^{\mathcal{M}} \models \phi$ iff $\mathcal{M} \models \phi'$.*

Proof 3.4.1 Consider a mapping Λ of CTL^* formulas to $MPL(\{\leq^{\mathcal{M}}\} \cup \{\xrightarrow{a}_{\mathcal{M}} \mid a \in Act^{\mathcal{M}}\}, L)$ as in [57]. Then the following equivalences ensure that E(a) can be expressed without referring to branches in the tree:

$$\begin{aligned}
&K(\mathcal{M},L), s \models \mathrm{E}(a)\phi\\
&\quad\text{iff}\;\; \mathcal{M}, y \mapsto s \models \exists(x : \mathcal{B}).\,(y \in x \wedge \exists(s' : Sit).\,(s' \in x \wedge y \xrightarrow{a} s' \wedge \Lambda(\phi,s')))\\
&\quad\text{iff}\;\; \mathcal{M}, y \mapsto s \models \exists(s' : Sit).\,(y \xrightarrow{a} s' \wedge \Lambda(\phi,s'))\\
&\quad\text{iff}\;\; \mathcal{M}, y \mapsto s \models \Phi(\mathrm{E}(a)\phi, y)
\end{aligned}$$

The simplification is valid since from $y \xrightarrow{a} s'$ and follows that there is a branch $x \in \mathcal{B}$ such that $y \in x$ and $s' \in x$ (as $\mathcal{B}$ contains all branches of the tree).

Similarly E$\phi U\psi$ can be expressed without referring to branches in the tree:

$$\begin{aligned}
&K(\mathcal{M},L), s \models \mathrm{E}\phi U\psi\\
&\quad\text{iff}\;\; \mathcal{M}, y \to s \models \exists(x : \mathcal{B}).\,(y \in x \wedge \exists(s' : Sit).\,(s' \in x \wedge y \leq s' \wedge \Lambda(\psi,s') \wedge\\
&\qquad\qquad \forall(s'' : Sit).\,(s \leq s'' \wedge s'' < s' \Rightarrow \Lambda(\phi,s''))))\\
&\quad\text{iff}\;\; \mathcal{M}, y \to s \models \exists(s' : Sit).\,(y \leq s' \wedge \Lambda(\psi,s') \wedge\\
&\qquad\qquad \forall(s'' : Sit).\,(s \leq s'' \wedge s'' < s' \Rightarrow \Lambda(\phi,s'')))\\
&\quad\text{iff}\;\; \mathcal{M}, y \to s \models \Phi(\mathrm{E}\phi U\psi, y)
\end{aligned}$$

Again, the simplification is valid since from $y \xrightarrow{a_1} s_1$, $s_1 \xrightarrow{a_2} s_2, \ldots, s_{n-1} \xrightarrow{a_n} s'$ follows that there is a branch $x \in \mathcal{B}$ such that $y \in x, s_1 \in x, \ldots, s' \in x$.

Let ϕ be a formula of CTL_U, then for the formula $\phi' \equiv \exists(r : Sit).\,(\forall(s : Sit).\, r \leq s \wedge \Phi(\phi, r))$ holds clearly $K(\mathcal{M}, L), s0_{\mathcal{M}} \models \phi$ iff $\mathcal{M} \models \phi'$. □

Using the logic CTL_U we may represent the planning problem as defined by Query 2.9 by satisfiability of the simple formula

$$\lambda_0 \wedge \mathrm{EF}\lambda_e. \tag{3.1}$$

The extended planning problem of Query 2.11 may be represented by validity of

$$\lambda_0 \Rightarrow \mathrm{EF}\lambda_e \tag{3.2}$$

Example 3.4.1 (Airport continued) *Consider our airport domain of Example 2.5.4. Assume that in the initial state no plane has landed, no passenger has arrived yet, one runway and four bays are available (the number of planes queueing is unknown). We would like to know whether there is an action sequence that causes precisely 22 passenger units to arrive. This planning problem can be encoded as a CTL_U formula as follows:*

$$\begin{aligned}&Holds(runway \circ bay^4) \wedge\\ &\overline{Holds(runway^2 \circ bay^5 \circ passenger \circ plane\_l\_b \circ plane\_l\_s)} \wedge\\ &EF(Holds(passenger^{22}) \wedge \overline{Holds(passenger^{23})}).\end{aligned}$$

If we only like to know whether an action sequences exists which causes at least 22 passenger units to arrive then the question can be represented as a conjunctive planning problem, i.e. a planning problem without negative propositions in the goal:

$$\begin{aligned}&Holds(runway \circ bay^4) \wedge\\ &\overline{Holds(runway^2 \circ bay^5 \circ passenger \circ plane\_l\_b \circ plane\_l\_s)} \wedge\\ &EF\,Holds(passenger^{22}).\end{aligned}$$

Now let us assume that in the above initial state the exact number of available bays is unknown, but there is at least one. We would like to know whether there is an action sequence for each possible initial state that causes precisely 22 passenger units to arrive. This is an extended planning problem which can be represented in CTL_U as follows:

$$\begin{aligned}&Holds(runway \circ bay) \wedge\\ &\overline{Holds(runway^2 \circ passenger \circ plane\_l\_b \circ plane\_l\_s)} \Rightarrow\\ &EF(Holds(passenger^{22}) \wedge \overline{Holds(passenger^{23})}).\end{aligned}$$

□

Note that a decision procedure for satisfiability of $\phi \wedge \mathrm{EF}\psi$ may also be used to decide validity of $\phi \Rightarrow \mathrm{AG}\neg\psi$ and a decision procedure for validity of $\phi \Rightarrow \mathrm{EF}\psi$ to decide satisfiability of $\phi \wedge \mathrm{AG}\neg\psi$.

The characterisation of models of $\mathcal{FC}$ domain descriptions by transition systems together with the characterisation of queries by expressions of modal/temporal logics we are now ready to investigate suitable classes of transition systems, i.e. classes that suggest strong relations to particular $\mathcal{FC}$ fragments and for which many results concerning model checking have been achieved.

Chapter 4

The Fluent Calculus and Petri nets

The formal and graphical language of Petri nets has been developed in [116]. Petri nets are used to model concurrent systems. They allow the specification of infinite state systems and are true generalisations of finite state automata. After many years of research about Petri nets their theory is well developed. In this chapter we will establish tight relationships between $\mathcal{FC}_{PL}$, $\mathcal{FC}_P$ and classes of Petri nets. As we will show in Chapter 5, these relationships are very fruitful since they allow us to answer many decidability problems in the Fluent Calculus theory with the help of their Petri net counterparts.

4.1 Petri nets

In the following, basic concepts of Petri net theory are defined. The underlying idea is that a concurrent system consists of processes. For execution a process requires certain system resources. During execution the required resources are blocked by the executed process, i.e. they can not be used by other processes. After execution some of the blocked resources may not be available any more and new resources may have been created.

Definition 4.1.1 *A tuple $\mathcal{P} = (P, T, E, W, m_0)$ is called a* Petri net *if*

1. *P and T are non-empty finite disjoint sets of vertices, elements of P are called* places *and elements of T are called* transitions. *$E \subseteq (P \times T) \cup (T \times P)$ is a set of edges.*

2. *$W : E \to \mathbb{N}^+$, which is called a* weight function.

3. *A mapping $m : P \to \mathbb{N}$ is called a* marking. *m_0 is a marking, called the* initial *marking. The set $\mathbb{N}^P$ of vectors is understood as the set of all markings.*

The set of transitions in Petri nets is used to model processes of the concurrent system and places represent different types of system resources. The weight function together with the set of edges expresses how many resources of what type are blocked, used and generated by some process. The actual amount of available resources of each type is modelled by a marking. Consequently, a marking denotes the state of a concurrent system where only the available resources are considered.

Petri nets are usually represented graphically by depicting places and transitions as vertices and edges as directed connections between them. Weights of edges are represented as labels if different from 1. The number associated with a place by a marking is depicted by an appropriate number of so called *tokens* on this place.

Example 4.1.1 (Airport continued) *Consider the airport model of Example 2.2.2 and Example 2.5.4. The number of small planes and big planes queueing is represented by the places* plane_q_s *and* plane_q_b*. The transitions* queue_s *and* queue_b *model the arrival of the planes. The bay capacity is represented by the tokens on place* bay*. Landing is modelled by the transitions* land_b *and* land_s *for big and small planes, respectively. The tokens on the places* plane_l_b *and* plane_l_s *model the number of planes at the airport. Passenger units are modelled by the tokens on the place* passenger*. The takeoff is described by the transitions* take_off_b *(big plane) and* take_off_s *(small plane). The runway is modelled by the place* runway*. In Figure 4.1 we consider the state of an airport system where three planes are queueing (two big planes and one small plane), four bays are available, no plane has got permission to land and one runway is available for takeoff.* □

The actual meaning of condition, effect and changing depends on the chosen semantics for Petri nets. Here, we consider only the most popular semantics in Petri net theory – interleaving. As it turns out, interleaving corresponds to the standard semantics of action execution in the Fluent Calculus.

Interleaving

In the following we will also denote a marking as a vector $m \in \mathbb{N}^P$. We will also use a partial ordering on vectors of integers: $m \geq m'$ for two markings m and m' iff $m(p) \geq m'(p)$ for all places $p \in P$.

According to the interleaving semantics, a marking may change only if there is a single transition causing this change.

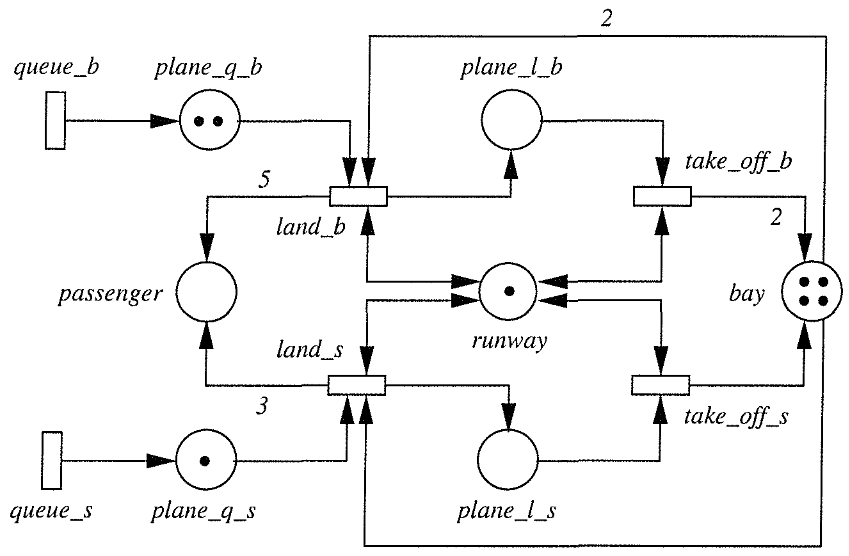


Figure 4.1: The airport example as Petri net.

Definition 4.1.2 (Interleaving semantics) *Let* (P, T, E, W, m_0) *be a Petri net.* Condition, effect *and* change *are defined for each transition* $t \in T$ *and place* $p \in P$ *by*

- $t^-(p) = W((p, t))$,
- $t^+(p) = W((t, p))$,
- $\delta t(p) = t^+(p) - t^-(p)$, *respectively.*

A transition $t \in T$ *is* enabled *at a marking* m *iff* $t^- \leq m$. *If an enabled transition* t *is* fired, *for the new marking* m' *holds* $m' = m + \delta t$.

In the Example 4.1.1, the condition $land\_b^-$ of firing transition *land_b* is given by $land\_b^-(bay) = 2$, $land\_b^-(plane\_q\_b) = 1$, $land\_b^-(runway) = 1$, and $land\_b^-(p) = 0$ for $p \in \{plane\_q\_s, plane\_l\_b, plane\_l\_b, passenger\}$.

The effect $land\_b^+$ is defined by $land\_b^+(runway) = 1$, $land\_b^+(passenger) = 5$, $land\_b^+(plane\_l\_b) = 1$, and $land\_b^+(p) = 0$ for all other places p.

Consequently, for the change $\delta land\_b$ follows that $\delta land\_b(passenger) = 5$, $\delta land\_b(bay) = -2$, $\delta land\_b(plane\_q\_b) = -1$, $\delta land\_b(plane\_l\_b) = 1$, and $\delta land\_b(p) = 0$ for all other places p.

The above definitions can be generalised to describe the subsequent execution of several transitions. Like t^- describes the minimal marking for the transition t to be enabled τ^- describes the minimal marking for a sequence τ of transitions to be enabled. For τ to be enabled in some marking the first transition must be enabled and the remaining transition sequence must be enabled after executing the first transition. Consequently, the minimal marking τ^- is given by the minimal marking enabling both (characterised mathematically by componentwise max) the first transition and the remaining sequence adjusted by the effect of the first transition (which may produce or consume tokens required for the remaining sequence).

Definition 4.1.3 *Let (P, T, E, W, m_0) be a Petri net and $t \in T$. The* transition *of the marking m to m' by firing the enabled transition t is denoted as $m[t\rangle m'$.*

For finite transition sequences $\tau = t_1 \cdots t_n$ with $t_i \in T$ for $1 \leq i \leq n$ we define δ, $\_^+$, $\_^-$ inductively:

1. *if $n = 0$, τ is* enabled *in any marking m, $\delta\tau = \tau^- = \tau^+ = \vec{0}$,*
2. *if $n > 0$, $\tau^- = \max(t_1^-, (t_2 \cdots t_n)^- - \delta t_1)$, $\delta\tau = \delta t_1 + \delta(t_2 \cdots t_n)$, $\tau^+ = \delta\tau + \tau^-$. τ is* enabled *in the marking m iff t_1 is enabled in m and $t_2 \cdots t_n$ is enabled in $m + \delta t_1$*

We write $m[\tau\rangle m'$ iff $m' = m + \delta\tau$ and τ is enabled in m. $[\tau\rangle$ denotes the relation on $\mathbb{N}^P \times \mathbb{N}^P$ such that $(m, m') \in [\tau\rangle$ iff $m[\tau\rangle m'$.

Characterisations of Petri nets

All possible behaviours of a Petri nets system can be described by the notion of the reachability tree of the system if an initial marking is given. The set of reachable markings of a Petri net, which is part of the information contained in the reachability tree, is often used to classify the system.

Definition 4.1.4 *Let $\mathcal{P} = (P, T, E, W, m_0)$ be a Petri net. We define the* reachability tree, *$RT(\mathcal{P})$, inductively as follows:*

1. *Let m_0 be the marking of the root node.*
2. *For every node n of $RT(\mathcal{P})$ labelled by some marking m and for every transition t which is enabled in m, add a node n' marked m' such that $m[t\rangle m'$ and add an arc from n to n' labelled t.*

The set $\Re(m_0, \mathcal{P})$ of all markings of $RT(\mathcal{P})$ is called the reachability set *of $\mathcal{P}$.*

If the behaviour of a Petri net has to be investigated, many propositions about markings can be reduced to the question whether the marking *covers* another one as defined in the following:

Definition 4.1.5 *Let (P, T, E, W, m_0) be a Petri net. Then the mapping $\alpha : \mathbb{N}^P \times \mathbb{N}^P \to \{\top, \bot\}$ is defined for $m, n \in \mathbb{N}^P$ as follows: $\alpha(m, n) = \top$ iff $m \leq n$. We say also n* covers *m.*

Let $K(\mathcal{P}, \mathbb{N}^P) = (\Re(m_0, \mathcal{P}), \to, T, \alpha)$, where $\to$ denotes the set of relations $\xrightarrow{t}$ with $(m, m') \in \xrightarrow{t}$ iff $m[t\rangle m'$ for $m, m' \in \Re(m_0, \mathcal{P})$.

The significance of covering of some marking m in Petri nets results from the fact that whenever a transition sequence can be fired in m it can be fired in all markings m' which cover m, for example *land_b* can be fired in all markings covering m with $m(bay) = 2$, $m(plane\_q\_b) = 1$, $m(runway) = 1$, and $m(p) = 0$ for all other places p.

Clearly, a Petri net defines a $\mathbb{N}^P$–valued transition system according to Definition 2.7.1:

Proposition 4.1.1 *Let $\mathcal{P} = (P, T, E, W, m_0)$ be a Petri net. Then $K(\mathcal{P}, \mathbb{N}^P)$ is a $\mathbb{N}^P$-valued transition system.*

To distinguish an element m of $\mathbb{N}^P$ with its algebraic properties from the proposition m in a $\mathbb{N}^P$–valued transition system we write instead *covered*(m) if we refer to m being a proposition (where $\alpha(covered(m), n) \overset{\text{def}}{=} \alpha(m, n)$).

In the following subsections we show correspondences between different Fluent Calculus fragments and appropriate instances of Petri nets. In particular, we will use the following definition of 1–safe Petri nets as it can be found in, e.g. [10].

Definition 4.1.6 *A Petri net (P, T, E, W, m_0) is called* 1–safe *iff*

1. $\forall m \in \Re(m_0, \mathcal{P}). m \leq \vec{1}$
2. $\forall e \in E. W(e) = 1$

We describe a 1-safe Petri net also as a tuple (P, T, E, m_0).

4.2 Coverability Analysis of Petri nets

The *coverability problem* is a classical problem in Petri net theory and is also sometimes referred to as the *control-state reachability problem.* The question is: given a marking m is there a marking m' in $\Re(m_0, \mathcal{P})$ which covers m, i.e., $m' \geq m$. We may represent this problem as a particular conjunctive planning problem wrt. the $\mathbb{N}^P$-valued transition system associated with some Petri net:

$$covered(m_0) \wedge \neg(\bigvee_{p \in P} covered(m_0 + \epsilon^p)) \wedge \mathrm{EF}\, covered(m) \tag{4.1}$$

Thereby ϵ^p denotes the mapping $P \to \mathbb{N}$ where $\epsilon^p(q) = 0$ for all $q \in P$ with $q \neq p$ and $\epsilon^p(p) = 1$. Consequently, the sub-formula $m_0 \wedge \neg(\bigvee_{p \in P} \mathit{covered}(m_0 + \epsilon^p))$ of Formula 4.1 before the temporal operator is valid for some marking m' of the transition system iff $m_0 = m'$. The sub-formula EF$\mathit{covered}(m)$ states that there is a branch where m is covered, eventually.

This and many other interesting properties of Petri nets can be investigated using the so-called *Karp-Miller tree* resulting from the following algorithm[1], first defined in [76]. The Karp-Miller tree is a finite abstraction of the set of reachable markings $\Re(m_0, \mathcal{P})$ in the sense that every element of the possibly infinite set of reachable markings of the Petri net $\mathcal{P}$ is represented by some node in the finite Karp-Miller tree of $\mathcal{P}$. On the other hand, a node of the Karp-Miller tree may also represent markings that can not be reached. However, we can decide whether it is possible to cover some arbitrary marking m' in $\mathcal{P}$ simply by checking whether a node in the Karp-Miller tree covers m'. Relevant information is not lost since a node in the Karp-Miller tree represents only a marking m that cannot be reached if a marking m' can be actually reached which covers m. Also, from the Karp-Miller tree we cannot find the set of smallest covering markings, but we can find some covering marking if one exists.

The Karp–Miller tree uses abstract markings, called *pseudo-markings* to represent certain sets of markings. Pseudo-markings are functions from P to $\mathbb{N} \cup \{\omega\}$. We also define $\forall n \in \mathbb{N} : \omega > n$ and $\omega + n = \omega - n = \omega + \omega = \omega$. Using this we also extend the notation $m[t_1 \cdots t_k\rangle m'$ for such markings in the obvious way.

Algorithm 4.2.1 (*Karp-Miller Tree*)

```
Input: a Petri net $\mathcal{P} = (P, T, E, W, m_0)$, a name $r$ for the root node
Output: a tree $RT$ of nodes labelled by pseudo-markings
Initialisation: set $U := \{node(r, m_0)\}$ of unprocessed nodes
while $U \neq \emptyset$

    select some $(k, m) \in U$;
    $U := U \setminus \{(k, m)\}$;
    if there is no ancestor node $(k_1, m_1)$ of $(k, m)$ with $m = m_1$ then
        $m_2 = m$;
        for all ancestors $(k_1, m_1)$ of $(k, m)$ such that $m_1 < m$ do
            for all places $p \in P$ such that $m_1(p) < m(p)$ do $m_2(p) = \omega$;
        $m := m_2$;
        for every transition $t$ such that $m[t\rangle m'$ do
            create node $(k', m')$;
            create arc labelled $t$ from $(k, m)$ to $(k', m')$;
            $U := U \cup (k', m')$;
```

[1]The algorithm presented here differs slightly from the original.

The main idea of the algorithm is to simulate a Petri net $\mathcal{P}$ until one reaches a marking which is greater or equal than a preceding one. If the marking is strictly greater, then one will generalise the marking by inserting ω's for all places where the number of tokens has actually increased. Simulation will then proceed using this new pseudo-marking. For example, if one can reach a marking $m_2 = \langle 1, 3, 2, 2\rangle$ from $m_1 = \langle 1, 0, 2, 1\rangle$ by firing a certain sequence of transitions $t_1 \cdots t_n$, the algorithm will proceed from the pseudo-marking $\langle 1, \omega, 2, \omega\rangle$. The introduction of ω's is justified by the monotonicity of Petri nets. I.e. whenever a transition sequence τ with $\delta\tau \geq \vec{0}$ is enabled in some marking m, then τ is enabled in the marking m' with $m[\tau\rangle m'$. If, additionally, $\delta\tau(p) > 0$ for some place p, then by repeatedly firing τ we may increase the number of tokens at place p arbitrarily. In the above example, the repeated firing of the sequence $t_1 \cdots t_n$ can be used to generate arbitrarily large number of tokens in the places 2 and 4.

4.3 $\mathcal{FC}_{PL}$ and Petri nets

Now we are ready to define a mapping from models $\mathcal{M}$ of domain descriptions $\mathcal{D}$ to Petri nets. Basically, fluents are mapped to places and actions are mapped to transitions. However, the weight function has to reflect both conditions and (positive and negative) effects of the execution of actions. Note that we cannot map domain descriptions to Petri nets, since they do not associate a particular state with the initial situation $s0$.

Definition 4.3.1 **($\mathcal{FC}_{PL} \to$ Petri nets)** *Let $\mathcal{D}$ be a domain description in $\mathcal{FC}_{PL}$ and let $\mathcal{M}$ be a model of $\mathcal{D}$. Then, by $\mathcal{P}(\mathcal{D}, \mathcal{M})$ we define a mapping of $\mathcal{M}$ and $\mathcal{D}$ to a Petri net (P, T, E, W, m_0), such that*

1. *$P = Fl^{\mathcal{M}}$, $T = Act^{\mathcal{M}}$,*
2. *let $Fl(g)$ denote the set of all fluents occurring in $g \in St^{\mathcal{M}}$. For each $\mathtt{SUA}^a_{\mathcal{D}} \in \mathtt{SUA}_{\mathcal{D}}$ we define with the help of $F_a^- = Fl(St_a^-)$, $F_a^+ = Fl(St_a^+)$, $F_a^= = Fl(St_a^=)$ the edges of the Petri net:*

$$E = \bigcup_{a \in Act^{\mathcal{M}}} (F_a^- \cup F_a^=) \times \{a\} \cup \bigcup_{a \in Act^{\mathcal{M}}} \{a\} \times (F_a^= \cup F_a^+)$$

3. *for each $(f, a) \in E$, $W((f, a)) = |St_a^-, f| + |St_a^=, f|$, and for each $(f, a) \in E$, $W((f, a)) = |St_a^+, f| + |St_a^=, f|$.*

For each $g \in St_{\mathcal{M}}$ a marking $m_{\mathcal{M}}(g) : St_{\mathcal{M}} \to \mathbb{N}^P$ is defined as follows:

$$m_{\mathcal{M}}(g) = \{f \to n \mid f \in Fl_{\mathcal{M}} \wedge n = |g, f|\}$$

Accordingly, for each $s \in Sit_{\mathcal{M}}$ a marking $m_{\mathcal{M}}(s) : Sit_{\mathcal{M}} \to \mathbb{N}^P$ is defined as $m_{\mathcal{M}}(s) = m_{\mathcal{M}}(state_{\mathcal{M}}(s))$. In particular, m_0 is given by $m_{\mathcal{M}}(s0_{\mathcal{M}})$.

The following theorem establishes correctness of the above mapping, thereby enabling the reduction of many problems concerning models of $\mathcal{FC}_{PL}$-descriptions to problems of Petri net theory! Note that the transition system associated with a model $\mathcal{M}$ of some $\mathcal{FC}_{PL}$ domain $\mathcal{D}$ always forms a tree. But the transition system associated with the Petri net corresponding to $\mathcal{D}$ and $\mathcal{M}$ usually does not. Hence, the transition systems are in general not isomorphic but bisimilar.

Theorem 4.3.1 *Let $\mathcal{M}$ be a model of some domain description $\mathcal{D}$ in $\mathcal{FC}_{PL}$ with signature Σ. Let $\mathcal{P} = (P, T, E, W, m_0) = \mathcal{P}(\mathcal{D}, \mathcal{M})$ be the corresponding Petri net. Then $K(\mathcal{M}, L_\Sigma)$ and $K(\mathcal{P}, \mathbb{N}^P)$ are bisimilar.*

Proof 4.3.1 To show bisimularity we need to prove 1) the existence of appropriate mappings between the propositions of the two transition systems, 2) that for each state of the first transition system exists a bisimilar state of the second system, and 3) that for each state of the second transition system exists a bisimilar state of the first system.

1) Mappings. Let's consider the mappings $\Psi_{\mathcal{M}} : L_\Sigma \to \mathbb{N}^P$ and $\Psi_{\mathcal{P}} : \mathbb{N}^P \to L_\Sigma$ between propositions of $K(\mathcal{P}, \mathbb{N}^P)$ and $K(\mathcal{M}, L_\Sigma)$:

$$\begin{aligned}\Psi_{\mathcal{M}}(\mathit{Holds}(g)) &\overset{\text{def}}{=} \mathit{covered}(m_{\mathcal{M}}(g)),\\ \Psi_{\mathcal{P}}(\mathit{covered}(m)) &\overset{\text{def}}{=} \mathit{Holds}(f_1^{m(f_1)} \circ f_2^{m(f_2)} \circ \cdots \circ f_k^{m(f_k)})\end{aligned}$$

where $\{f_1, f_2, \ldots, f_k\} = P$, $\mathit{Holds}(g) \in L_\Sigma$ and $m \in \mathbb{N}^P$. Since we consider Herbrand-$\mathrm{E}_{\mathcal{FC}}$-models only, the situation $s0^{\mathcal{M}}$ exists in every $\mathcal{M}$ and is predecessor (wrt $\leq^{\mathcal{M}}$) of all other $s \in \mathit{Sit}^{\mathcal{M}}$, hence $K(\mathcal{M}, L_\Sigma)$ is a rooted transition system. Since $K(\mathcal{P}(\mathcal{D}, \mathcal{M}), \mathbb{N}^P)$ is rooted as well (in $m_0(s0^{\mathcal{M}})$), it suffices to show the existence of a bisimulation Φ with $(s0^{\mathcal{M}}, m_0(s0^{\mathcal{M}})) \in \Phi$ and the consistency of the mappings $\Psi_{\mathcal{P}}$ and $\Psi_{\mathcal{M}}$.

First, we show that for every situation $s \in \mathit{Sit}^{\mathcal{M}}$, $\alpha_{\mathcal{M}}(\mathit{Holds}(g), s)$ with $g \in \mathit{St}^{\mathcal{M}}$ is true iff $\alpha(\Psi_{\mathcal{M}}(\mathit{Holds}(g)), m_{\mathcal{M}}(s))$ is true. From the equational theory for the sort $\mathit{St}^{\mathcal{M}}$, which must be fulfilled in every model of $\mathcal{D}$, follows that $\mathcal{M} \models \mathit{Holds}(g, s)$ iff for all $g' \in \mathit{St}^{\mathcal{M}}$, such that $|g', f| = |g, f|$ for all $f \in \mathit{Fl}^{\mathcal{M}}$, $\mathcal{M} \models \mathit{Holds}(g', s)$. Furthermore, from the definition of $\mathit{Holds}(g, s)$ follows that $\mathcal{M} \models \mathit{Holds}(g, s)$ iff $|g, f| \leq |\mathit{state}_{\mathcal{M}}(s), f|$ for all $f \in \mathit{Fl}^{\mathcal{M}}$. From the definition of the corresponding Petri net follows $(m_{\mathcal{M}}(s))(f) = |\mathit{state}^{\mathcal{M}}(s), f|$ for all $f \in \mathit{Fl}^{\mathcal{M}}$ and for markings m' with $\mathit{covered}(m') = \Psi_{\mathcal{M}}(\mathit{Holds}(g)) = \mathit{covered}(m_{\mathcal{M}}(g))$ holds $m'(f) = |g, f|$ for all $f \in \mathit{Fl}^{\mathcal{M}}$. Hence, $\mathcal{M} \models \mathit{Holds}(g, s)$ iff $m \leq m_{\mathcal{M}}(s)$ with $\mathit{covered}(m) = \Psi_{\mathcal{M}}(\mathit{Holds}(g))$, i.e. (by definition of $\alpha_{\mathcal{M}}$ and α $\alpha_{\mathcal{M}}(\mathit{Holds}(g), s)$ is true iff $\alpha(\Psi_{\mathcal{M}}(\mathit{Holds}(g)), m_{\mathcal{M}}(s))$ is true.

Similarly, for every $m \in \Re(m_0, \mathcal{P})$ and every $m' : P \to \mathbb{N}$, $\alpha(\mathit{covered}(m'), m)$ is true iff $\alpha(\Psi_{\mathcal{P}}(\mathit{covered}(m')), s)$ is true where $s \in \mathit{Sit}^{\mathcal{M}}$ and $m = m_{\mathcal{M}}(s)$. This follows from the definition of $\Psi_{\mathcal{P}}$ and the fact that every $s \in \mathit{Sit}^{\mathcal{M}}$ with

$m = m_{\mathcal{M}}(s)$ fulfils $|state^{\mathcal{M}}(s), p| = m(p)$ for all $p \in P$: $\Psi_{\mathcal{P}}(covered(m')) = Holds(p_1^{m'(p_1)} \circ p_2^{m'(p_2)} \circ \cdots \circ p_k^{m'(p_k)}) = Holds(g)$, $\alpha_{\mathcal{M}}(Holds(g), s)$ is true for $s \in Sit^{\mathcal{M}}$ iff $|g, p| \leq |state^{\mathcal{M}}(s), p|$ for all $p \in Fl^{\mathcal{M}}$.

2) Bisimilarity I. Now let $(s, m) \in \Phi \subseteq Sit^{\mathcal{M}} \times \Re(m_0, \mathcal{P})$ such that $m_{\mathcal{M}}(s) = m$. We prove that, for every action $a \in Act^{\mathcal{M}}$ and $s \xrightarrow{a}_{\mathcal{M}} s'$, there exists a marking m' such that $m_{\mathcal{M}}(s') = m'$ and $m \xrightarrow{a} m'$. From the state update axioms follows for $s' = do^{\mathcal{M}}(a, s)$ that $\mathcal{M} \models state(do(a,s)) \circ St_a^- =_{St} state(s) \circ St_a^+$ (since $\mathcal{M} \models Holds(St_a^- \circ St_a^=, s)$ if $s \xrightarrow{a}_{\mathcal{M}} s'$).

From the above arguments follows $\mathcal{M} \models Holds(St_a^- \circ St_a^=, s)$ iff $|St_a^-, p| + |St_a^=, p| \leq m(p)$ for all $p \in P$. From the definition of P we conclude that there is a transition $a \in T$ such that $W(p, a) = |St_a^-, p| + |St_a^=, p|$. Hence, a is enabled in $m_{\mathcal{M}}(s)$ iff $\mathcal{M} \models Holds(St_a^- \circ St_a^=, s)$. Furthermore, according to the state update axiom for $a \in Act^{\mathcal{M}}$ for all $f \in Fl^{\mathcal{M}}$: $|state^{\mathcal{M}}(do^{\mathcal{M}}(a,s)), f| = |state^{\mathcal{M}}(s), f| - |St_a^-, f| + |St_a^+, f|$. If the transition a is fired in P, $m \xrightarrow{a} m'$, then for m' and all $f \in Fl^{\mathcal{M}}$ holds

$$\begin{aligned} m'_{\mathcal{M}}(f) &= m_{\mathcal{M}}(f) - |St_a^-, f| - |St_a^=, f|) + |St_a^+, f| + |St_a^=, f| \\ &= m_{\mathcal{M}}(f) - |St_a^-, f| + |St_a^+, f| = |state_{\mathcal{M}}(s'), f| \end{aligned}$$

3) Bisimilarity II. Analogously, for every transition $t \in T$ and $m \xrightarrow{t} m'$, there exists $s' \in Sit^{\mathcal{M}}$ and $a \in Act^{\mathcal{M}}$ such that $m_{\mathcal{M}}(s') = m'$ and $s' = do^{\mathcal{M}}(a, s)$. This follows from the one–to–one mapping of $\mathcal{T}_{\Sigma, Act}(\emptyset)$ to T and from the above bidirectional correspondences. Since, for $(s0^{\mathcal{M}}, m_0)$ $m_{\mathcal{M}}(s0^{\mathcal{M}}) = m_0$ holds (see definition), it follows by induction that $s0^{\mathcal{M}} \sim m_0$. □

Example 4.3.1 (**$\mathcal{FC}_{PL} \to$ Petri net**) *Let $\mathcal{M}_a$ be a model of $\mathcal{D}_a$ of Example 2.5.4 such that $state^{\mathcal{M}}(s0^{\mathcal{M}}) =_{St} bay^4 \circ plane\_q\_b^2 \circ plane\_q\_s \circ runway$. Then $\mathcal{P}(\mathcal{D}_a, \mathcal{M}_a)$ is isomorphic to the Petri net $\mathcal{P}$ of Example 4.1.1 where fluent names correspond with place names, and action names correspond with transition names. $K(\mathcal{M}_a, L_{\Sigma_a})$ and $K(\mathcal{P}, \mathbb{N}^P)$ are bisimilar.* □

The above theorem will be applied in Section 5.3 to reduce satisfiability of the planning problem, i.e. the satisfiability of formulas like $\lambda_0 \wedge \mathrm{EF}\lambda_e$, to the (decidable) reachability problem in Petri nets.

However, the question arises whether we can establish a similar correspondence in the opposite direction, i.e. we want to show that for every Petri net, there exists a corresponding model of a domain description in $\mathcal{FC}_{PL}$. To this end, we use the following mapping. Note that in this mapping we do not hard-wire the initial marking m_0 of a Petri net; this will enable us to examine richer classes of problems later on.

Definition 4.3.2 (**Petri nets $\to \mathcal{FC}_{PL}$**) *Let $\mathcal{P} = (P, T, E, W, m_0)$ be a Petri net. Then by $\mathcal{D}(\mathcal{P})$ a mapping from a Petri net $\mathcal{P}$ to a domain description $\mathcal{D}$ is defined as follows:*

1. *The signature of $\mathcal{D}$ is given by a $\mathcal{FC}_{PL}$ signature where the elements of Act and Fl are given by FUN: $\{(p :\rightarrow Fl) \mid p \in P\}$, $\{(t :\rightarrow Act) \mid t \in T\}\}$*

2. *Let for each transition t*

$$\begin{array}{rcl} St_t^{=} & = & p_1^{\max(0,W(t,p_1)-W(p_1,t))} \circ \cdots \circ p_k^{\max(0,W(t,p_k)-W(p_k,t))} \\ St_t^{+} & = & p_1^{W(t,p_1)-\max(0,W(t,p_1)-W(p_1,t))} \circ \cdots \circ \\ & & p_k^{W(t,p_k)-\max(0,W(t,p_k)-W(p_k,t))} \\ St_t^{-} & = & p_1^{W(p_1,t)-\max(0,W(t,p_1)-W(p_1,t))} \circ \cdots \circ \\ & & p_k^{W(p_k,t)-\max(0,W(t,p_k)-W(p_k,t))} \end{array}$$

for $\{p_1, \ldots, p_k\} = P$. For every $t \in T$, $\mathcal{D}(\mathcal{P})$ contains the following pair of precondition and state update axioms:

$$\forall(s : Sit).\,(Poss(t,s) \Leftrightarrow Holds(St_t^{-} \circ St_t^{=}, s))$$
$$\forall(s : Sit).\,(Poss(t,s) \Rightarrow state(do(t,s)) \circ St_t^{-} =_{St} state(s) \circ St_t^{+})$$

Furthermore, for every domain description $\mathcal{D}$, we assume the domain independent axioms described in Section 2.2.

The following theorem establishes correctness of the above embedding of Petri nets into $\mathcal{FC}_{PL}$ and it will be the main tool to prove the undecidability of the $\mathcal{FC}_{PL}$ entailment problem in Section 5.2. (The reason why the theorem does not hold for all models of $\mathcal{D}$ is that the Fluent Calculus encoding does not contain the initial marking of the Petri net. However, as evident from the proof, the particular model $\mathcal{M}$ can be isolated easily.)

Theorem 4.3.2 *For every Petri net $\mathcal{P} = (P, T, E, W, m_0)$, there exists a domain description $\mathcal{D}$ wrt some signature Σ in $\mathcal{FC}_{PL}$ and a model $\mathcal{M}$ of $\mathcal{D}$, such that the transition system $K(\mathcal{M}, L_\Sigma)$ and $K(\mathcal{P}, \mathbb{N}^P)$ are bisimilar.*

Proof 4.3.2 We consider the Herbrand-$\mathrm{E}_{\mathcal{FC}}$-models defined by the marking m_0: $\mathcal{M} \models state(s0) =_{St} p_1^{m_0(p_1)} \circ \cdots \circ p_k^{m_0(p_k)}$ where $P = \{p_1, \ldots, p_k\}$.

1) Mappings. Furthermore, the mappings $\Psi_{\mathcal{M}}$ and $\Psi_{\mathcal{P}}$ describe the mappings between labels of $K(\mathcal{M}, L_\Sigma)$ and $K(\mathcal{P}, \mathbb{N}^P)$.

2) Bisimilarity I. Now let $(s, m) \in \Phi \subseteq Sit^{\mathcal{M}} \times \Re(m_0, \mathcal{P})$ such that $s \in Sit^{\mathcal{M}}(m)$ where $Sit^{\mathcal{M}}(m)$ denotes the set of all situations s such that $m_{\mathcal{M}}(s) = m$. We prove that for every transition $t \in T$ and $m \xrightarrow{t} m'$, there exists a situation s' such that $m_{\mathcal{M}}(s') = m'$ and $s \xrightarrow{t}_{\mathcal{M}} s'$ for all $s \in Sit^{\mathcal{M}}(m)$. From the definition of $\mathcal{P}$ follows for the marking m', if $m \xrightarrow{t} m'$,

$$m'(p) = m(p) + t^{+}(p) - t^{-}(p) = m(p) + W(t,p) - W(p,t)$$

for all $p \in P$, since $t^{-}(p) = W(p,t) \leq m(p)$ for all $p \in P$.

From the definition of $Holds(g, s)$ follows that $\mathcal{M} \models Holds(g, s)$ with $g \in St^{\mathcal{M}}$ iff $|g, p| \leq m_{\mathcal{M}}(s)(p)$ for all $p \in P$. From the definition of $\mathcal{D}$ we conclude that there is a state update axiom $\mathtt{SUA}^t_{\mathcal{D}}$ with $t \in T_{\Sigma,Act}(\emptyset)$:

$$\forall(s : Sit).\,(Holds(St_t^- \circ St_t^=, s) \Rightarrow state(do(t, s)) \circ St_t^- =_{St} state(s) \circ St_t^+)$$

where, in every model of $\mathcal{D}$,

$$\begin{aligned} St_t^- \circ St_t^= \quad =_{St} \quad & p_1^{W(t,p_1)-\max(0,W(t,p_1)-W(p_1,t))} \circ \cdots \circ \\ & p_k^{W(t,p_k)-\max(0,W(t,g_k)-W(g_k,t))} \\ & \circ\, p_1^{\max(0,W(t,p_1)-W(p_1,t))} \circ \cdots \circ p_k^{\max(0,W(t,g_k)-W(g_k,t))} \\ =_{St} \quad & p_1^{W(p_1,t)} \circ \cdots \circ p_k^{W(p_k,t)}. \end{aligned}$$

Hence, $\mathcal{M} \models Holds(p_1^{W(p_1,t)} \circ \cdots \circ p_k^{W(p_k,t)}, s)$ iff t is enabled in m.

If t is fired, according to the state update axiom for t, for all $p \in Fl^{\mathcal{M}}$:

$$\begin{aligned} & |state^{\mathcal{M}}(do^{\mathcal{M}}(t, s)), p| \\ = \quad & |state^{\mathcal{M}}(s), p| - |St_t^-, p| + |St_t^+, p| \\ = \quad & |state^{\mathcal{M}}(s), p| - (W(p, t) - \max(0, W(t, p) - W(p, t))) \\ & +W(t, p) - \max(0, W(t, p) - W(p, t))) \\ = \quad & |state^{\mathcal{M}}(s), p| + W(t, p) - W(p, t) \end{aligned}$$

3) Bisimilarity II. Analogously, for every action $a \in Act^{\mathcal{M}}$ and $s \xrightarrow{a}_{\mathcal{M}} s'$, there exists $m' \in \Re(m_0, \mathcal{P})$ and $t \in T$ such that $m_{\mathcal{M}}(s') = m'$ and $m \xrightarrow{t} m'$. This follows from the one–to–one mapping of T to $T_{\Sigma,Act}(\emptyset)$ and from the above bidirectional correspondences.

For $(s0^{\mathcal{M}}, m_0)$ holds $m_{\mathcal{M}}(s0^{\mathcal{M}}) = m_0$. By induction follows $s0^{\mathcal{M}} \sim m_0$. □

It is well known that the class $\mathcal{C}_{FA}$ of transition systems defined by finite automata without acceptance condition is strictly contained in the class of transition systems defined by Petri nets [116]. As a consequence of the above theorem, $\mathcal{C}_{FA}$ is also strictly contained in the class of transition systems defined by $\mathcal{FC}_{PL}$ domains. In [145] it has been shown that the set of transition systems definable by a restricted Situation Calculus fragment corresponds to $\mathcal{C}_{FA}$. Hence, in terms of the associated transition systems $\mathcal{FC}_{PL}$ domains are strictly more expressive than Situation Calculus domains of [145].

4.4 $\mathcal{FC}_P$ and 1–safe Petri nets

As we show in this section a similar correspondence as between models of $\mathcal{FC}_{PL}$ domains and Petri nets can be established between models of $\mathcal{FC}_P$ domains and 1–safe Petri nets.

Definition 4.4.1 (**$\mathcal{FC}_P \to$ 1–safe Petri nets**) *Let $\mathcal{D}$ be a domain description in $\mathcal{FC}_P$ and let $\mathcal{M}$ be a model of $\mathcal{D}$. Then, by $\mathcal{P}(\mathcal{D}, \mathcal{M})$ we define a mapping of $\mathcal{M}$ and $\mathcal{D}$ to a Petri net (P, T, E, W, m_0):*

1. $P = \{\overline{f} \mid f \in \mathit{Fluent}^{\mathcal{M}}\} \cup \mathit{Fl}^{\mathcal{M}}$,
2. $T = \{a \in \mathit{Act}^{\mathcal{M}}\}$,
3. *let $\mathit{Fl}(g)$ denote the set of all fluents occurring in $g \in \mathit{St}^{\mathcal{M}}$ and let $a \in \mathit{Act}^{\mathcal{M}}$. Furthermore, let $F_a^- = \mathit{Fl}(\mathit{St}_a^-)$, $F_a^+ = \mathit{Fl}(\mathit{St}_a^+)$, $F_a^= = \mathit{Fl}(\mathit{St}_a^=)$, $F_a^n = \mathit{Fl}(\mathit{St}_a^n)$ and $\overline{G} = \{\overline{f} \mid f \in G\}$ for some set $G \subseteq \mathit{Fl}^{\mathcal{M}}$. Then the edges of the Petri net are defined as:*

$$\begin{aligned} E \;=\; & \bigcup_{a \in \mathit{Act}^{\mathcal{M}}} (F_a^- \cup F_a^= \cup \overline{F_a^n}) \times \{a\} \cup \\ & \bigcup_{a \in \mathit{Act}^{\mathcal{M}}} \{a\} \times (\overline{((F_a^- \cup F_a^n) \setminus F_a^+)} \cup F_a^+ \cup F_a^=) \end{aligned}$$

4. *for each $e \in E$, $W(e) = 1$.*

For each $g \in \mathit{St}^{\mathcal{M}}$ a marking $m_{\mathcal{M}}(g) : \mathit{St}^{\mathcal{M}} \to \mathbb{N}^P$ is defined as follows:

$$m_{\mathcal{M}}(g) = \{f \to |g, f| \mid f \in \mathit{Fl}^{\mathcal{M}}\} \cup \{\overline{f} \to 1 - |g, f| \mid f \in \mathit{Fl}^{\mathcal{M}}\}$$

Accordingly, for each $s \in \mathit{Sit}^{\mathcal{M}}$ a marking $m_{\mathcal{M}}(s) : \mathit{Sit}^{\mathcal{M}} \to \mathbb{N}^P$ is defined as $m_{\mathcal{M}}(s) = m_{\mathcal{M}}(\mathit{state}^{\mathcal{M}}(s))$. In particular, m_0 is given by $m_{\mathcal{M}}(s0^{\mathcal{M}})$.

Instead of considering the set $\mathbb{N}^P$ of basic propositions as in Definition 4.1.1 we will use a restricted set of propositions which we define as $\mathbb{N}^P_{\mathit{Fl}}$. Each element of $\mathbb{N}^P_{\mathit{Fl}}$ is a function of the form:

$$\{f \mapsto n \mid f \in \mathit{Fl}^{\mathcal{M}} \wedge n \in \mathbb{N}\} \cup \{\overline{f} \to 0 \mid f \in \mathit{Fl}^{\mathcal{M}}\}.$$

In the following we will show that in the Petri net resulting from the above mapping the place $\overline{f}$ is marked iff f is not marked. As a consequence, all propositions *covered*(m) of $\mathbb{N}^P$ mapping $\overline{f}$ to a number different from zero can be expressed by propositions of $\mathbb{N}^P_{\mathit{Fl}}$ using negation, e.g., *covered*(m) with $m(\overline{f}) = 1$ can be expressed by *covered*$(m) \wedge \neg$*covered*(m') where $m(p) = m'(p)$ for all $p \in \mathit{Fl}^{\mathcal{M}}$ and $p \neq f$ and $m'(f) = 1$.

The following theorem establishes correctness of the above mapping and ensures that the result is indeed a 1–safe Petri net.

Theorem 4.4.1 *Let $\mathcal{M}$ be a model of some domain description $\mathcal{D}$ in $\mathcal{FC}_P$ with signature Σ. Let $\mathcal{P} = (P, T, E, W, m_0) = \mathcal{P}(\mathcal{D}, \mathcal{M})$ be the corresponding Petri net. Then*

1. *$K(\mathcal{M}, L_\Sigma)$ and $K(\mathcal{P}, \mathbb{N}^P)$ are bisimilar,*

2. *$\mathcal{P}$ is 1–safe.*

Proof 4.4.1
1) Mappings. We consider the mappings $\Psi_\mathcal{M} : L_\Sigma \to \mathbb{N}^P_{Fl}$ and $\Psi_\mathcal{P} : \mathbb{N}^P_{Fl} \to L_\Sigma$ between propositions of $K(\mathcal{M}, L_\Sigma)$ and $K(\mathcal{P}, \mathbb{N}^P)$:

$$\begin{aligned}\Psi_\mathcal{M}(\mathit{Holds}(g)) &\overset{\text{def}}{=} \mathit{covered}(\{f \mapsto |g,f| \mid f \in \mathit{Fl}^\mathcal{M}\} \\ &\qquad\qquad \cup \{\overline{f} \mapsto 0 \mid f \in \mathit{Fl}^\mathcal{M}\}), \\ \Psi_\mathcal{P}(\mathit{covered}(m)) &\overset{\text{def}}{=} \mathit{Holds}(f_1^{m(f_1)} \circ f_2^{m(f_2)} \circ \cdots \circ f_k^{m(f_k)})\end{aligned}$$

Since $K(\mathcal{M}, L_\Sigma)$ and $K(\mathcal{P}(\mathcal{D}, \mathcal{M}), \mathbb{N}^P)$ are both rooted, it suffices to show the existence of a bisimulation Φ with $(s0^\mathcal{M}, m_0(s0^\mathcal{M})) \in \Phi$ and the consistency of the mappings $\Psi_\mathcal{P}$ and $\Psi_\mathcal{M}$.

First, we show that for every situation $s \in \mathit{Sit}^\mathcal{M}$, $\alpha_\mathcal{M}(\mathit{Holds}(g), s)$ with $g \in \mathit{St}^\mathcal{M}$ is true iff $\alpha(\Psi_\mathcal{M}(\mathit{Holds}(g)), m_\mathcal{M}(s))$ is true. From the equational theory for the sort $\mathit{St}^\mathcal{M}$ follows that $\mathcal{M} \models \mathit{Holds}(g, s)$ iff for all $g' \in \mathit{St}^\mathcal{M}$, such that $|g', f| = |g, f|$ for all $f \in \mathit{Fl}^\mathcal{M}$, $\mathcal{M} \models \mathit{Holds}(g', s)$. Furthermore, from the definition of $\mathit{Holds}(g, s)$ follows that $\mathcal{M} \models \mathit{Holds}(g, s)$ iff $|g, f| \leq |\mathit{state}^\mathcal{M}(s), f|$ for all $f \in \mathit{Fl}^\mathcal{M}$. From the definition of the corresponding Petri net follows $(m_\mathcal{M}(s))(f) = |\mathit{state}^\mathcal{M}(s), f|$ for all $f \in \mathit{Fl}^\mathcal{M}$ and for markings m' with $\mathit{covered}(m') = \Psi_\mathcal{M}(\mathit{Holds}(g))$ holds $m'(f) = |g, f|$ for all $f \in \mathit{Fl}^\mathcal{M}$. Since $m'(\overline{f}) = 0$ for all $f \in \mathit{Fl}^\mathcal{M}$, these places do not influence the value of α. $\mathcal{M} \models \mathit{Holds}(g, s)$ iff for m' with $\mathit{covered}(m') = \Psi_\mathcal{M}(\mathit{Holds}(g))$ holds $m' \leq m_\mathcal{M}(s)$, i.e. (per definition of $\alpha_\mathcal{M}$ and α) $\alpha_\mathcal{M}(\mathit{Holds}(g), s)$ is true iff $\alpha(\Psi_\mathcal{M}(\mathit{Holds}(g)), m_\mathcal{M}(s))$ is true.

Similarly, for every $m \in \Re(m_0, \mathcal{P})$ and every $m' \in \mathbb{N}^P_{Fl}$, $\alpha(\mathit{covered}(m'), m)$ is true iff $\alpha(\Psi_\mathcal{P}(\mathit{covered}(m')), s)$ is true where $s \in \mathit{Sit}^\mathcal{M}$ and $m = m_\mathcal{M}(s)$. This follows from the definition of $\Psi_\mathcal{P}$ and the fact that every $s \in \mathit{Sit}^\mathcal{M}$ with $m = m_\mathcal{M}(s)$ fulfils $|\mathit{state}^\mathcal{M}(s), p| = m(p)$ for all $p \in \mathit{Fl}^\mathcal{M}$: $\Psi_\mathcal{P}(\mathit{covered}(m')) = \mathit{Holds}(p_1^{m'(p_1)} \circ p_2^{m'(p_2)} \circ \cdots \circ p_k^{m'(p_k)}) = \mathit{Holds}(g)$, $\alpha_\mathcal{M}(\mathit{Holds}(g), s)$ is true for $s \in \mathit{Sit}^\mathcal{M}$ iff $|g, p| \leq |\mathit{state}^\mathcal{M}(s), p|$ for all $p \in \mathit{Fl}^\mathcal{M}$.

2) Bisimilarity I. Now let $(s, m) \in \Phi \subseteq \mathit{Sit}^\mathcal{M} \times \Re(m_0, \mathcal{P})$ such that $m_\mathcal{M}(s) = m$. We prove that, for every action $a \in \mathit{Act}^\mathcal{M}$ and $s \xrightarrow{a}_\mathcal{M} s'$, there exists a marking m' such that $m_\mathcal{M}(s') = m'$ and $m \xrightarrow{a} m'$. From the state update axioms follows for $s' = do^\mathcal{M}(a, s)$ that $\mathcal{M} \models \mathit{state}(do(a, s)) \circ \mathit{St}_a^- =_{St} \mathit{state}(s) \circ \mathit{St}_a^+$ (since $\mathcal{M} \models \mathit{Holds}(\mathit{St}_a^- \circ \mathit{St}_a^=, s)$ if $s \xrightarrow{a}_\mathcal{M} s'$).

From the above arguments follows $\mathcal{M} \models \underline{\mathit{Holds}(\mathit{St}_a^- \circ \mathit{St}_a^=, s) \text{ iff } |\mathit{St}_a^-, p|} +$ $|\mathit{St}_a^=, p| \leq m(p)$ for all $p \in \mathit{Fl}^\mathcal{M}$. Furthermore, $\mathcal{M} \models \underline{\mathit{Holds}(\mathit{St}_a^- \circ \mathit{St}_a^n, s)}$ iff $m(p) < |\mathit{St}_a^n, p|$ for all $p \in \mathit{Fl}^\mathcal{M}$ and $m(p) < |\mathit{St}_a^n, p|$ iff $|\mathit{St}_a^n, p| \leq m(\overline{p})$.

From the definition of $\mathcal{P}$ we conclude that there is a transition $a \in T$ such that $W(p, a) = |\mathit{St}_a^-, p| + |\mathit{St}_a^=, p|$ and $W(\overline{p}, a) = |\mathit{St}_a^n, p|$ for all $p \in \mathit{Fl}_{calM}$.

Hence, a is enabled in $m_{\mathcal{M}}(s)$ iff $\mathcal{M} \models \mathit{Holds}(St_a^- \circ St_a^=, s) \wedge \overline{\mathit{Holds}(St_a^n, s)}$. Furthermore, according to the state update axiom for $a \in \mathit{Act}$ for all $f \in \mathit{Fl}^{\mathcal{M}}$: $|\mathit{state}^{\mathcal{M}}(\mathit{do}^{\mathcal{M}}(a,s)), f| = |\mathit{state}^{\mathcal{M}}(s), f| - |St_a^-, f| + |St_a^+, f|$. If the transition a is fired in $\mathcal{P}$, $m \xrightarrow{a} m'$, then for m' and all $f \in \mathit{Fl}^{\mathcal{M}}$ holds

$$\begin{aligned} m'(f) &= m(f) - |St_a^-, f| - |St_a^=, f|) + |St_a^+, f| + |St_a^=, f| \\ &= m(f) - |St_a^-, f| + |St_a^+, f| = |\mathit{state}^{\mathcal{M}}(s'), f| \end{aligned}$$

and

$$\begin{aligned} m'(\overline{f}) &= m(\overline{f}) - |St_a^n, f| + |St_a^n, f| + |St_a^-, f| - |St_a^+, f| \\ &= m(\overline{f}) + |St_a^-, f| - |St_a^+, f| = 1 - |\mathit{state}^{\mathcal{M}}(s'), f| = 1 - m'(f) \end{aligned}$$

3) Bisimilarity II. Analogously, for every transition $t \in T$ and $m \xrightarrow{t} m'$, there exists $s' \in \mathit{Sit}^{\mathcal{M}}$ and $a \in \mathit{Act}^{\mathcal{M}}$ such that $m_{\mathcal{M}}(s') = m'$ and $s' = \mathit{do}^{\mathcal{M}}(a,s)$. This follows from the one–to–one mapping of $T_{\Sigma,\mathit{Act}}(\emptyset)$ to T and from the above bidirectional correspondences. Since, for $(s0^{\mathcal{M}}, m_0)$ $(m_{\mathcal{M}}(s0^{\mathcal{M}}))(f) = m_0(f)$ holds and $(m_{\mathcal{M}}(s0^{\mathcal{M}}))(\overline{f}) = m_0(\overline{f})$ for all fluents $f \in \mathit{Fl}^{\mathcal{M}}$ (see definition), it follows by induction that $s0^{\mathcal{M}} \sim m_0$.

Finally, by axiom (NM) in $\mathcal{FC}_P$ it follows $m_{\mathcal{M}}(s0^{\mathcal{M}})(f) \leq 1$ for all models $\mathcal{M}$ in the corresponding Petri net $\mathcal{P}$. By definition of $m_{\mathcal{M}}$ and the above equation $m'(\overline{f}) = 1 - m'(f)$ which holds after firing any transition in $\mathcal{P}$ for all fluents $f \in \mathit{Fl}^{\mathcal{M}}$, the Petri net $\mathcal{P}$ is 1–safe. □

This theorem will be applied in Section 5 to reduce satisfiability of formulas of the propositional μ-calculus for $\mathcal{FC}_P$ domains to satisfiability for 1–safe Petri nets.

Example 4.4.1 (**$\mathcal{FC}_P \rightarrow$ 1–safe Petri net**) *Consider the domain of Example 2.5.6 and the model $\mathcal{M}_o$ where $\mathit{state}(s0) =_{St} \mathit{runway} \circ \mathit{bay} \circ \mathit{plane\_q\_b} \circ \mathit{plane\_q\_s}$. The corresponding 1–safe Petri net is depicted in Figure 4.2. Note in particular the places $\overline{\mathit{plane\_q\_b}}$ and $\overline{\mathit{plane\_q\_s}}$ which represent the non-existence of fluents plane_q_b and plane_q_s.* □

Again, we want to establish a similar correspondence in the opposite direction, i.e. we want to show that for every 1–safe Petri net, there exists a corresponding model of a domain description in $\mathcal{FC}_P$. To this end, we use the mapping given in Definition 4.3.2. It remains to be shown that axiom (NM) is fulfilled for all 1–safe Petri nets.

Theorem 4.4.2 *For every 1-safe Petri net $\mathcal{P} = (P, T, E, m_0)$, there exists a domain description $\mathcal{D}$ in $\mathcal{FC}_P$ and a model $\mathcal{M}$ of $\mathcal{D}$, such that $K(\mathcal{M}, L_\Sigma)$ and $K(\mathcal{P}, \mathbb{N}^P)$ are bisimilar.*

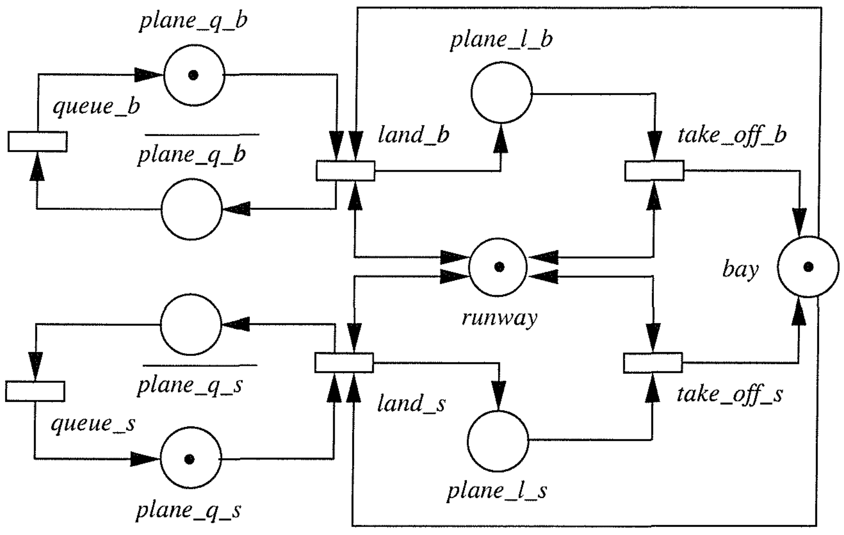


Figure 4.2: The 1–safe Petri net corresponding to the domain of Example 2.5.6.

Proof 4.4.2 Consider the Herbrand–$E_{\mathcal{FC}}$–models defined by the marking m_0: $\mathcal{M} \models state(s0) =_{St} p_1^{m_0(p_1)} \circ \cdots \circ p_k^{m_0(p_k)}$ where $P = \{p_1, \ldots, p_k\}$. Furthermore, the mappings $\Psi_{\mathcal{M}}$ and $\Psi_{\mathcal{P}}$ of Proof 4.3.1 describe the mappings between labels of $K(\mathcal{M}, L_\Sigma)$ and $K(\mathcal{P}, \mathbb{N}^P)$. Following the Proof 4.3.1 for a marking m' with $m \xrightarrow{t} m'$, $m'(p) = m(p) + W(t,p) - W(p,t)$ and

$$|state^{\mathcal{M}}(do^{\mathcal{M}}(t,s)), p| = |state^{\mathcal{M}}(s), p| + W(t,p) - W(p,t)$$

if $m'(p) = |state^{\mathcal{M}}(s), p|$ for all $p \in P$ and all $s \in Sit^{\mathcal{M}}$. Consequently, $|state^{\mathcal{M}}(do^{\mathcal{M}}(t,s)), p| \leq 1$ iff $m'(p) \leq 1$ and axiom (NM) is valid if $\mathcal{P}$ is 1–safe. □

As a consequence of this theorem, $\mathcal{FC}_P$ is equally expressive as a further restricted fragment of $\mathcal{FC}_P$ where precondition axioms must not contain expressions of the form $\overline{Holds(g,s)}$. This follows from the above proof where the mapping from 1–safe Petri nets to $\mathcal{FC}_P$ is defined using the mapping from Petri nets to $\mathcal{FC}_{PL}$.

In this chapter we have shown that propositional Fluent Calculus domains, in particular $\mathcal{FC}_{PL}$ and $\mathcal{FC}_P$ domains, are strongly related to Petri nets. It is well known (e.g., [141]) that the propositional μ–calculus cannot distinguish between strongly bisimilar transition systems, i.e. all formulas that are valid for one transition system are also valid in all bisimilar systems. As we proved bisimulation equivalence between models of these Fluent Calculus domains and Petri nets, we can reduce satisfiability problems (of the propositional μ-calculus) for $\mathcal{FC}_P$ and $\mathcal{FC}_{PL}$ to/from satisfiability problems in Petri nets.

Chapter 5

Decidability of Reasoning in the Fluent Calculus

In this chapter we investigate the possibilities for automatic reasoning in the Fluent Calculus using the concepts introduced in the previous chapters. In the following four sections we apply the correspondences between Fluent Calculus domains and Petri nets. We start by proving decidability of satisfiability of the most powerful class of query logic formulas for the weakest Fluent Calculus fragment considered here – $\mathcal{FC}_P$. Then we show that satisfiability of the weakest query logic – CTL_U – is already undecidable for $\mathcal{FC}_{PL}$ – the fragment of $\mathcal{FC}$ that is more expressive than $\mathcal{FC}_P$ but weaker than all other fragments considered here. Having established boundaries for decidability of the entailment problem in this way, we prove in Section 5.3 and Section 5.4 decidability of the planning problem and the extended planning problem, respectively, for $\mathcal{FC}_{PL}$ domains. In Section 5.5 we show that these problems are not decidable in $\mathcal{FC}_L$ domains. To this end we show that the halting problem for any deterministic two-counter machine can be reduced to a planning problem for a $\mathcal{FC}_L$ domain. Similarly, in Section 5.6 we extend the result on undecidability of planning in $\mathcal{FC}_{PLN}$ domains of [67] to a propositional version of $\mathcal{FC}_{L\leq}$.

5.1 Entailment in $\mathcal{FC}_P$

Theorem 5.1.1 *Let $\mathcal{D}$ be a domain description $\mathcal{D}$ in $\mathcal{FC}_P$ with signature Σ and ϕ a formula of the propositional μ–calculus wrt L_Σ and $T_{Act,\Sigma}(\emptyset)$. Let ϕ' denote the formula of the query logic corresponding to ϕ according to Corollary 3.2.2. Then it is decidable whether ϕ' is satisfiable by some model $\mathcal{M}$ of $\mathcal{D}$.*

Proof 5.1.1 To every 1–safe Petri net $\mathcal{P}$ is associated a bisimilar finite automaton $\mathcal{A}_\mathcal{P}$ where $\mathcal{A}_\mathcal{P}$ consists simply of the reachability graph of $\mathcal{P}$, i.e. the graph resulting from identifying nodes labelled by equivalent markings in the reachability tree of $\mathcal{P}$. This is due to the fact that there exist only $2^{|P|}$ possible markings in a 1–safe Petri net with P the set of places. Consequently, a finite automaton is also associated with every model $\mathcal{M}$ of a $\mathcal{FC}_P$ domain description $\mathcal{D}$. Furthermore, since every model is up to bisimularity determined by the initial state, every domain description has only finitely many non–bisimilar models. As a consequence of [126] it follows that for any given finite automaton the satisfiability of any formula ϕ of the propositional μ–calculus is decidable (for a discussion of algorithms cf. [30]). By Corollary 3.2.2 satisfiability of ϕ' can be reduced to satisfiability of ϕ. □

Clearly, the decision procedure for satisfiability of some formula ϕ may be also used to decide validity of $\neg\phi$.

Instead of using the correspondence between $\mathcal{FC}_P$ domain descriptions and 1–safe Petri nets of Theorem 4.4.1, we may prove the above also by establishing a direct link between $\mathcal{FC}_P$ and finite automata. However, for 1–safe Petri nets efficient and more specialised model checking methods have been developed, e.g. [36]. By applying Theorem 5.1.1 we may apply these methods to $\mathcal{FC}_P$ domains with little additional effort.

5.2 Entailment in $\mathcal{FC}_{PL}$

In this section we show that entailment for $\mathcal{FC}_{PL}$, the second weakest fragment investigated here, is already undecidable, even if we only consider the first order queries (formulas characterised by CTL_U). As the Fluent Calculus fragments $\mathcal{FC}_{PLN}$, $\mathcal{FC}_L$ and $\mathcal{FC}_{LN}$, respectively, are strictly more expressive than $\mathcal{FC}_{PL}$, it follows that entailment of CTL_U is undecidable in these fragments as well.

Theorem 5.2.1 *Let $\mathcal{D}$ be a $\mathcal{FC}_{PL}$ domain description with signature Σ and ϕ a CTL_U-formula wrt L_Σ and $T_{Act,\Sigma}(\emptyset)$. The question whether ϕ is satisfiable by $\mathcal{D}$, is undecidable.*

Proof 5.2.1 In [34] with a correction in [35] the undecidability of the model checking problem of the following formula has been shown for Petri nets $\mathcal{P} = (P, T, E, W, m_0)$:

$$\pi = \mathrm{AG}(\mathrm{E}(t_{AB})\top \Rightarrow \mathrm{E}(t_{AB})\mathrm{EF}\,\mathrm{Dead})$$

where $t_{AB} \in T$, Dead $= \bigwedge_{t \in T} \neg\mathrm{E}(t)\top$. The formula can be read as: Whenever the transition t_{AB} is enabled then eventually after firing t_{AB} a marking is reached where no tranisition is enabled.

The model checking problem for a Petri net $\mathcal{P} = (P, T, E, W, m_0)$ is defined as the problem to decide whether the associated transition system $K(\mathcal{P}, \mathbb{N}^P)$ with the initial marking m_0 satisfies a formula ϕ. Hence, to prove the claim, it is sufficient to show that for every Petri net there is a domain description $\mathcal{D}$ and a formula π' such that a model $\mathcal{M}$ of $\mathcal{D}$ satisfies π' iff $K(\mathcal{P}, \mathbb{N}^P)$ satisfies π in m_0.

Note that the labels $\mathbb{N}^P$ of the transition system $K(\mathcal{P}, \mathbb{N}^P)$ can only represent lower bounds on the number of tokens of some marking. However, we may express upper bounds by using negation. Consequently, an initial marking m can be completely characterised as the set of all markings n such that for all $p \in P$, $m(p) \leq n(p) \wedge \neg(\overline{m}(p) \leq n(p))$ where $\overline{m}(p) = m(p) + 1$. I.e., if $\mathcal{P} = (P, T, E, W, m_0)$ and $\mathcal{P}' = (P, T, E, W, m_0')$ are two Petri nets and both associated transition systems satisfy for all $p \in P$, $m(p) \leq n(p) \wedge \neg(\overline{m}(p) \leq n(p))$ in m_0 and m_0', respectively, it follows $m_0 = m_0' = m$ and $\mathcal{P}$ and $\mathcal{P}'$ are equivalent. Let $\hat{g}$ denote the term $p_1^{|g,p_1|+1} \circ \cdots \circ p_n^{|g,p_n|+1}$ for $p_i \in P$, $1 \leq i \leq n$.

Assume, that $\mathcal{D}(\mathcal{P})$ is the domain description which corresponds to the Petri net constructed in Section 4.5 of [35]. Then a model $\mathcal{M}$ satisfies

$$\begin{aligned}\pi' = \ & \mathit{Holds}(g) \wedge \overline{\mathit{Holds}(\hat{g})} \wedge \\ & \mathrm{AG}(\mathrm{E}(t_{AB})\top \Rightarrow \mathrm{E}(t_{AB})\mathrm{EF} \textstyle\bigwedge_{t \in T} \neg \mathit{Holds}(St_t^= \circ St_t^-))\end{aligned}$$

where $\mathit{Holds}(g) = \Psi_{\mathcal{P}}(m)$ using the mapping of Proof 4.3.1 and $g \in T_{\Sigma,St}(\emptyset)$, in situation $s_{\mathcal{M}}$ iff there is a model $\mathcal{M}'$ of $\mathcal{D}(\mathcal{P})$ such that $s0^{\mathcal{M}'}$ satisfies π' (note, that any two L-labelled transition systems for $\mathcal{D}(\mathcal{P})$ rooted in $s_{\mathcal{M}}$ and $s0^{\mathcal{M}'}$, respectively, where $|\mathit{state}^{\mathcal{M}'}(s0^{\mathcal{M}'}), f| = |\mathit{state}^{\mathcal{M}}(s_{\mathcal{M}}), f|$ for all $f \in T_{\Sigma,Fl}(\emptyset)$ are isomorphic). The transition system $K(\mathcal{M}', L_\Sigma)$ is bisimilar to $K(\mathcal{P}, \mathbb{N}^P)$, hence it satisfies π' iff $K(\mathcal{P}, \mathbb{N}^P)$ satisfies π. □

Corollary 5.2.2 *Let $\mathcal{D}$ be a $\mathcal{FC}_{PL}$ domain description with signature Σ and ϕ a CTL_U-formula wrt L_Σ and $T_{Act,\Sigma}(\emptyset)$. The question whether ϕ is entailed by $\mathcal{D}$, is undecidable.*

Proof 5.2.2 This follows easily from the fact, that for the above Petri net

$$\begin{aligned}\pi'' = \ & (\mathit{Holds}(g) \wedge \overline{\mathit{Holds}(\hat{g})}) \Rightarrow \\ & \mathrm{AG}(\mathrm{E}(t_{AB})\top \Rightarrow \mathrm{E}(t_{AB})\mathrm{EF} \textstyle\bigwedge_{t \in T} \neg \mathit{Holds}(St_t^= \circ St_t^-))\end{aligned}$$

is entailed by $\mathcal{D}$ iff π' is satisfied by $\mathcal{D}$ (due to the isomorphism mentioned in the previous proof). □

Note that the property described by the formula π'' in the above proof can be characterised as follows: "Whenever a certain action a is executable, after the execution of a, it is possible to reach a terminal state". Such propositions could, e.g., be of interest if the resource management of an operating system has to be verified. Furthermore, we can imagine a train entering a fail–safe mode. We might want to know, whether the train, after entering this mode, cannot change certain parameters anymore, e.g. increasing of speed is impossible. The undecidability of the entailment problem restricts the possibility of automated verification of such properties.

Note that the proof in [34] reduces undecidability of the above problem to undecidability of the question whether two Petri nets have the same reachability sets. This result has been shown by Rabin and a proof can be found in [56]. As a consequence of this result, equivalence between domain descriptions wrt some initial states is in general undecidable as well.

5.3 Planning Problems in $\mathcal{FC}_{PL}$

Despite of the above undecidable entailment problems, important classes of CTL_U-formulas can be decided. To simplify the presentation of the following results we define the set Λ_{Holds} as a subset of the boolean combinations of labels L_Σ (see Definition 2.6.2).

Definition 5.3.1 *Let Σ be a $\mathcal{FC}$ signature.*

$$\Lambda_{Holds} = \{Holds(g^p) \wedge \overline{Holds(g^n)} \mid g^p, g^n \in T_{St,\Sigma}(\emptyset)\}$$

Theorem 5.3.1 *Let $\mathcal{D}$ be an arbitrary domain description in $\mathcal{FC}_{PL}$. Then, the satisfiability of any formula of the form $\pi = \lambda_0 \wedge EF\lambda_e$ where $\lambda_0, \lambda_e \in \Lambda_{Holds}$ is decidable.*

Proof 5.3.1 In this proof we construct a domain $\mathcal{D}'$ and a finite set of formulas Π', such that $\mathcal{D}'$ satisfies some formula $\pi' \in \Pi'$ iff $\mathcal{D}$ satisfies π. Then, we show that the question whether $\mathcal{D}'$ satisfies π' is decidable by reduction to the Petri net reachability problem which is known to be decidable ([103]).

Consider g_0^p to be the term g^p, g_0^n the term g^n of the formula λ_0 and g_e^p to be the term g^p, g_e^n the term g^n of the formula λ_e respectively. Again, by $\hat{g}$ we denote for every state term g the term $f_1^{|g,f_1|+1} \circ \cdots \circ f_n^{|g,f_n|+1}$ for all f_i of Fl.

We define G_0^n to be the set of fluents not occurring in g_0^n and G_e^n the fluents not occurring in g_e^n. These sets contain those fluents which can appear arbitrarily often in the initial state described by λ_0 and in the final state described by λ_e. Now, we introduce a new signature Σ' containing the signature Σ of $\mathcal{D}$ and additionally, two sets, $A_0 = \{a_0^f \mid f \in G_0^n\}$ and $A_e = \{a_e^f \mid f \in G_e^n\}$, of new constants of type Act. We augment the domain description $\mathcal{D}$ by the following pairs of precondition axiom and state update axiom and call the resulting domain description[1] $\mathcal{D}'$:

$$\forall(s : \mathit{Sit}).\,(\mathit{Poss}(a,s) \Leftrightarrow \top)$$
$$\forall(s : \mathit{Sit}).\,(\mathit{Poss}(a,s) \Rightarrow \mathit{state}(\mathit{do}(a_0^f, s)) =_{St} \mathit{state}(s) \circ f)$$

for each $f \in G_0^n$, and

$$\forall(s : \mathit{Sit}).\,(\mathit{Poss}(a,s) \Leftrightarrow \mathit{Holds}(f,s))$$
$$\forall(s : \mathit{Sit}).\,(\mathit{Poss}(a,s) \Rightarrow f \circ \mathit{state}(\mathit{do}(a_e^f, s)) =_{St} \mathit{state}(s))$$

for each $f \in G_e^n$.

We call the first set of state update axioms (those for G_0^n) $\mathtt{SUA}_0$ and the second set of state update axioms (those for G_e^n) $\mathtt{SUA}_e$. Now, let Π' be the set of formulas

$$\mathit{Holds}(g_0') \wedge \overline{\mathit{Holds}(\hat{g_0'})} \wedge \mathrm{EF}(\mathit{Holds}(g_e') \wedge \overline{\mathit{Holds}(\hat{g_e'})})$$

for all $g_0', g_e' \in T_{\Sigma,St}(\emptyset)$ with $|g_0^n, f| > |g_0', f| \geq |g_0^p, f|$ for all $f \in T_{\Sigma,Fl}(\emptyset)$ where $f \notin G_0^n$ and $|g_0', f| = |g_0^p, f|$ for all $f \in G_0^n$, and $|g_e^n, f| > |g_e', f| \geq |g_e^p, f|$ for all $f \in T_{\Sigma,Fl}(\emptyset)$ where $f \notin G_e^n$ and $|g_e', f| = |g_e^p, f|$ for all $f \in G_e^n$.

Since for every $n \geq 0$ and fluent f, $\hat{f}^n$ contains fluent f at least once, it follows that the set of all g such that $\bigwedge_{f \in T_{\Sigma,Fl}(\emptyset)} \forall(z : St).\, \hat{f}^n \circ z \neq_{St} f^{|g,f|}$ is finite (by definition of $\mathit{Holds}(\_,\_)$), hence Π' is finite.

Now, we show that whenever $\mathcal{D}$ satisfies π there is a $\pi' \in \Pi'$ such that $\mathcal{D}'$ satisfies π'. Assume, $\mathcal{D}$ satisfies π, i.e. there is a model $\mathcal{M}$ and a sequence of situations $s_0, s_1, \ldots, s_n$ in $K(\mathcal{M}, L_\Sigma)$ such that λ_0 is true in s_0 and λ_e is true in s_n. Since we did not remove any state update axioms it suffices to show, that there is a $\pi' \in \Pi'$, with the corresponding λ_0, λ_e denoted as $\lambda_0^{\pi'}$ and $\lambda_e^{\pi'}$, and a model $\mathcal{M}'$ of $\mathcal{D}'$ such that

1. there is a situation s_0' with $s_{-m}' \xrightarrow{a_{-m}} s_{-m}' \xrightarrow{a_{-m+1}} \cdots \xrightarrow{a_{-2}} s_{-1}' \xrightarrow{a_{-1}} s_0'$ such that $\lambda_0^{\pi'}$ holds in s_{-m}' and $|\mathit{state}^{\mathcal{M}}(s_0), f| = |\mathit{state}^{\mathcal{M}'}(s_0'), f|$ for all $f \in T_{\Sigma,Fl}(\emptyset)$, and

2. there is a situation s_{n+k}' with $s_n' \xrightarrow{a_n} s_{n+1}' \xrightarrow{a_{n+1}} \cdots \xrightarrow{a_{n+k-2}} s_{n+k-1}' \xrightarrow{a_{n+k-1}} s_{n+k}'$ such that $\lambda_e^{\pi'}$ holds in s_{n+k}' and $|\mathit{state}^{\mathcal{M}}(s_n), f| = |\mathit{state}^{\mathcal{M}'}(s_n'), f|$ for all $f \in T_{\Sigma,Fl}(\emptyset)$.

Assume, there is a situation s_0 fulfilling λ_0 and for a situation $s_{-m}' \in \mathit{Sit}^{\mathcal{M}'}$ for some model $\mathcal{M}'$ of $\mathcal{D}'$ the number n_f of each fluent f occurring in $\mathit{state}^{\mathcal{M}'}(s_{-m}')$ is given by $|g_0^p, f| \leq n_f < |g_0^n, f|$ if $f \in G_0$, and $|g_0^p, f| \leq n_f < |g_0^p, f| + 1$ otherwise (clearly, such a model exists if a model $\mathcal{M}$ exists). Then a situation

[1]Note that in all models and for all situations $\mathit{Holds}(1^\circ, s)$ is assigned $\top$.

$s'_0 \in Sit^{\mathcal{M}'}$ such that $|state^{\mathcal{M}}(s_0), f| = |state^{\mathcal{M}'}(s'_0), f|$ for all fluents f can be reached by a finite number of transitions defined by axioms of $\mathtt{SUA}_0$ (describing the actions $a_{-1}, \dots, a_{-m}$). Similarly assume, there is a situation s_n fulfilling λ_e and for a situation $s'_{n+k} \in Sit^{\mathcal{M}'}$ for some model $\mathcal{M}'$ of $\mathcal{D}'$ the number n_f of each fluent f in $state^{\mathcal{M}'}(s'_{n+k})$ is given by $|g^p_e, f| \leq n_f < |g^n_e, f|$ if $f \in G_e$, and $|g^p_e, f| \leq n_f < |g^p_e, f| + 1$ otherwise (again, the $\mathcal{M}'$ exists if there is a model $\mathcal{M}$). Then a situation $s'_n \in Sit^{\mathcal{M}'}$ such that $|state^{\mathcal{M}}(s_n), f| = |state^{\mathcal{M}'}(s'_n), f|$ for all fluents f leads by a finite number of transitions defined by axioms of $\mathtt{SUA}_e$ (describing actions $a_n, \dots, a_{n+k-1}$) to the situation s'_{n+k}. According to construction of Π', there is a $\pi' \in \Pi'$ such that $\lambda_0^{\pi'}$ and $\lambda_e^{\pi'}$ determine the number of fluents correspondingly.

As a second step, we prove that whenever there is a $\pi' \in$ H$'$ such that $\mathcal{D}'$ satisfies π', $\mathcal{D}$ satisfies π. Suppose, that there is a model $\mathcal{M}'$ and a situation $s'_{n+k} = do^{\mathcal{M}'}(a_{n+k-1}, do^{\mathcal{M}'}(a_{n+k-1}, \dots, do^{\mathcal{M}'}(a_{-m}, s'_{-m}) \dots))$ in $\mathcal{M}'$ such that $\lambda_0^{\pi'}$ is true in s'_{-m}, $\lambda_e^{\pi'}$ is true in s'_{n+k} and to determine $state^{\mathcal{M}'}(s'_{n+k})$, m denotes the number of transitions which are described by axioms of $\mathtt{SUA}_0$, k denotes the number of transitions which are described by axioms of $\mathtt{SUA}_e$ and n denotes the number of transitions which are described by all remaining state update axioms. Then, the following propositions hold:

1. There is a situation $s''_0 \in Sit^{\mathcal{M}'}$ such that $s''_{-m} \xrightarrow{a'_{-m}} s''_{-m} \xrightarrow{a'_{-m+1}} \dots \xrightarrow{a'_{-2}} s''_{-1} \xrightarrow{a'_{-1}} s''_0$ where $a'_{-1}, \dots a'_{-m}$ represent exactly those actions in s'_{n+k} with descriptions in $\mathtt{SUA}_0$ and $\mathcal{M}' \models state(s''_{-m}) =_{St} state(s'_{-m})$. This is due to the fact, that an application of an axiom of $\mathtt{SUA}_0$ strictly increases the number of some fluent, only. Hence, applications of axioms which were possible in situations containing smaller amounts of this fluent, are still possible after increasing the number. Furthermore, clearly, λ_0 is true in all such s''_0.

2. There is a situation $s''_n \in Sit^{\mathcal{M}'}$ such that $s''_n \xrightarrow{a'_n} s''_{n+1} \xrightarrow{a'_{n+1}} \dots \xrightarrow{a'_{n+k-2}} s''_{n+k-1} \xrightarrow{a'_{n+k-1}} s''_{n+k}$ where $a'_n, \dots, a'_{n+k}$ represent exactly those actions in s'_{n+k} with descriptions in $\mathtt{SUA}_e$ and s''_n with $s''_0 \xrightarrow{a'_0} s''_1 \xrightarrow{a'_1} \dots \xrightarrow{a'_{n-2}} s''_{n-1} \xrightarrow{a'_{n-1}} s''_n$ where $a'_0, \dots, a'_{n-1}$ represent exactly those actions in s'_{n+k} with descriptions in $\mathtt{SUA}$ in the same order as they appear in s'_{n+k}. This is due to the fact, that an application of an axiom of $\mathtt{SUA}_e$ strictly decreases the number of some fluent, only. Hence, applications of axioms which were possible in situations containing greater amounts of this fluent, are still possible before decreasing the number. Note, that $\mathcal{M}' \models state(s''_{n+k}) =_{St} state(s'_{n+k})$ and clearly, λ_e is true in all such s''_n.

Since, $s''_0 \xrightarrow{a'_0} s''_1 \xrightarrow{a'_1} \dots \xrightarrow{a'_{n-2}} s''_{n-1} \xrightarrow{a'_{n-1}} s''_n$ contains only actions of $\mathcal{D}$ and any pair s''_n and s''_0 fulfils λ_0 and λ_e, respectively, there is a model $\mathcal{M}$ of $\mathcal{D}$ and

$s_0 \in Sit^{\mathcal{M}}$ and $s_n \in Sit^{\mathcal{M}}$ such that $|state^{\mathcal{M}}(s_0), f| = |state^{\mathcal{M}'}(s_0''), f|$ and $|state^{\mathcal{M}}(s_n), f| = |state^{\mathcal{M}'}(s_n''), f|$ for all fluents f and $s_0 \xrightarrow{a_0'} s_1 \xrightarrow{a_1'} \cdots \xrightarrow{a_{n-2}'} s_{n-1} \xrightarrow{a_{n-1}'} s_n$.

Finally, for every model $\mathcal{M}'$ of $\mathcal{D}'$ which satisfies some $\pi' \in \Pi'$ there is a model $\mathcal{M}''$ where π' is satisfied in $s0^{\mathcal{M}''}$. Hence, there exists a Petri net where $\mathcal{P}(\mathcal{D}', \mathcal{M}'') = (P, T, E, W, m_0)$ such that using the mappings of Proof 4.3.1,

$$\Psi_{\mathcal{M}''}(Holds(g_0'))(f) \leq m_0(f) < \Psi_{\mathcal{M}''}(Holds(f^{|g_0', f|+1}))(f)$$

for all fluents f (i.e. $m_0 = m_{\mathcal{M}''}(g_0')$). The formula π' is true in $\mathcal{M}''$ iff there is a sequence of actions $a_0, \ldots, a_{n-1}$ with $n \in \mathbb{N}$ such that $\lambda_e^{\pi'}$ holds in $s_n \in Sit^{\mathcal{M}''}$ with $s0^{\mathcal{M}''} \xrightarrow{a_0'} s_1 \xrightarrow{a_1'} \cdots \xrightarrow{a_{n-2}'} s_{n-1} \xrightarrow{a_{n-1}'} s_n$. Due to bisimularity, such an action sequence exists iff there is a corresponding transition sequence in $\mathcal{P}$ such that $m_0 \xrightarrow{t_0} m_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} m_e$ and $\Psi_{\mathcal{M}''}(Holds(g_e'))(f) \leq m_e(f) < \Psi_{\mathcal{M}''}(Holds(f^{|g_e', f|+1}))(f)$ (i.e. $m_e = m_{\mathcal{M}''}(g_e')$). The latter problem is called the *reachability problem* for Petri nets and is known to be decidable [103]. □

Corollary 5.3.2 *Let $\mathcal{D}$ be an arbitrary domain description in $\mathcal{FC}_{PL}$. Then, the satisfiability of any formula of the form $\pi = \lambda_0 \wedge EF\lambda_e$ where $\lambda_0, \lambda_e \in \Lambda_{L_\Sigma}$ is decidable.*

Proof 5.3.2 From Equation 2.7 it follows that for propositional $\mathcal{FC}$ domains any formula of Λ_{L_Σ}, i.e. the language of boolean combinations of elements of L_Σ (see Definition 2.6.2), can be represented as a finite disjunction of formulas of Λ_{Holds}. Consequently, any planning problem for $\mathcal{FC}_{PL}$ domains can be represented by a finite set of instances of the problem $\pi = \lambda_0 \wedge \mathrm{EF}\lambda_e$ where $\lambda_0, \lambda_e \in \Lambda_{Holds}$. These problems are decidable by Theorem 5.3.1. □

Note that the algorithm of [103] to decide Petri net reachability problems also allows the computation of a transition sequence which leads from the initial marking to the final marking. It can thus be applied to compute an actual action sequence which solves the planning problem.

Since the conjunctive planning problem is an instance of the planning problem we may apply the Petri net reachability algorithm also to decide the conjunctive planning problem. However, the complexity of this algorithm is very high and precise complexity boundaries for solving the reachability problem are not yet established. Instead, to solve the conjunctive planning problem we can also use the much simpler Procedure 4.2.1 of Karp-Miller. In Chapter 7 we propose an appropriately adapted method to this end and we show the completeness of the method for conjunctive planning problems for $\mathcal{FC}_{PL}$ domains.

Another question is whether it is possible to decide the extended planning problem in $\mathcal{FC}_{PL}$. Deciding the extended planning problem is particularly interesting when the initial situation is not completely known. In this case from the solution to the planning problem it might follow that there is a successful plan for some initial state. From the solution to an extended planning problem we may conclude whether there is a successful plan for every initial state. The decidability of the extended planning problem for $\mathcal{FC}_{PL}$ is examined in the next section.

5.4 Extended Planning Problems in $\mathcal{FC}_{PL}$

Theorem 5.4.1 *Let $\mathcal{D}$ be an arbitrary domain description in $\mathcal{FC}_{PL}$. Then the validity of any formula of the form $\pi = \lambda_0 \Rightarrow EF\lambda_e$ where $\lambda_0, \lambda_e \in \Lambda_{Holds}$ is decidable.*

Proof 5.4.1 Again, consider g_0^p to be the term g^p, g_0^n the term g^n of the formula λ_0 and g_e^p to be the term g^p, g_e^n the term g^n of the formula λ_e respectively. By $\hat{g}$ we denote a state term associated with state term g as defined in Proof 5.3.1.

We prove the claim by constructing a finite set $\Theta = \{\mathcal{P}'_1, \ldots, \mathcal{P}'_n\}$ of Petri nets $\mathcal{P}'_i = (P'_i, T'_i, E'_i, W'_i, m'_i)$, $1 \leq i \leq n$, such that π is true in all models of $\mathcal{D}$ iff the union of the reachability sets $\Re(m'_i, \mathcal{P}')$ for all $1 \leq i \leq n$ contains the set S of all markings m with $|g_0^p, f| \leq m(f) < |g_0^n, f|$ for all fluents f occurring in g_0^n and $|g_0^p, f| \leq m(f)$, otherwise. The latter problem is decidable due to the following results.

According to [56], for every finite set Θ of Petri nets with equivalent places P there exists a Petri net $\mathcal{P}_{\cup}^{\Theta} = (P^{\Theta}, T^{\Theta}, E^{\Theta}, W^{\Theta}, m^{\Theta})$ such that $\Re(m^{\Theta}, \mathcal{P}_{\cup}^{\Theta})|_P = \bigcup_{\mathcal{P}=(P,T,E,W,m)\in\Theta} \Re(m, \mathcal{P})$ where $\Re(m^{\Theta}, \mathcal{P}_{\cup}^{\Theta})|_{P^{\Theta}\setminus P}$ is finite. Consequently, the above problem can be reduced to the question whether for $\mathcal{P}_{\cup}^{\Theta}$, $\Re(m^{\Theta}, \mathcal{P}_{\cup}^{\Theta})|_P$ contains S (*containment problem*).

Note that S can be represented as a finite union of linear sets (i.e., S can be represented by a semi-linear set – for definitions of linear and semi-linear, see [29]) of markings given by the above conditions. Furthermore, for every linear set L it is straightforward to construct a Petri net $\mathcal{P}_L = (P_L, T_L, E_L, W_L, m_L)$ such that $\Re(m_L, \mathcal{P}_L) = L$. As in [85] it is possible to effectively construct for every such Petri net $\mathcal{P}_L$ a Petri net $\mathcal{P}_L^{\Theta} = (P_L^{\Theta}, T_L^{\Theta}, E_L^{\Theta}, W_L^{\Theta}, m_L^{\Theta})$ such that $\Re(m_L^{\Theta}, \mathcal{P}_L^{\Theta})|_{P_L} = \Re(m_L, \mathcal{P}_L) \cap \Re(m^{\Theta}, \mathcal{P}_{\cup}^{\Theta})|_{P_L}$, where $\Re(m_L^{\Theta}, \mathcal{P}_L^{\Theta})|_{P_L^{\Theta}\setminus P_L}$ is finite. Since $\Re(m_L, \mathcal{P}_L)$ is linear for all $\mathcal{P}_L$ and due to the construction of $\mathcal{P}_L^{\Theta}$ in [85], $\Re(m_L^{\Theta}, \mathcal{P}_L^{\Theta})$ must be semi-linear if $\Re(m_L, \mathcal{P}_L)|_{P_L} = \Re(m_L^{\Theta}, \mathcal{P}_L^{\Theta})|_{P_L}$. Hence, the containment problem is decidable by using the following propositions which were shown in [85, 61]:

1. It is decidable whether the reachability set of an arbitrary Petri net is semi-linear, and

2. if the reachability set is semi-linear, it can be effectively computed.

Finally, the equivalence of semi-linear sets can be decided (see, e.g. [50]). Hence, it is possible to decide whether $\Re(m^{\Theta}, \mathcal{P}_{U}^{\Theta})|_{P_L}$ contains $\Re(m_L, \mathcal{P}_L)$ for every linear set L of S by verifying that $\Re(m_L, \mathcal{P}_L) = \Re(m_L^{\Theta}, \mathcal{P}_L^{\Theta})|_{P_L}$.

Now, we present the construction of the finite set Θ of Petri nets, such that the first claim holds. To this end note that there is a symmetry fulfilled by every domain description $\mathcal{D}$ of $\mathcal{FC}_{PL}$, namely for every domain description $\mathcal{D}$ there is a *reverse* domain description $\hat{\mathcal{D}}$ such that for every model $\mathcal{M}$ of $\mathcal{D}$, situation $s \in Sit^{\mathcal{M}}$, $a \in Act^{\mathcal{M}}$ and $s \xrightarrow{a} s'$, there is a $\hat{\mathcal{M}}$ of $\hat{\mathcal{D}}$, situation $\hat{s} \in Sit^{\hat{\mathcal{M}}}$, $\hat{a} \in Act^{\hat{\mathcal{M}}}$ and $\hat{s} \xrightarrow{\hat{a}} \hat{s}'$ such that $|state^{\mathcal{M}}(s), f| = |state^{\hat{\mathcal{M}}}(\hat{s}'), f|$ and $|state^{\mathcal{M}}(s'), f| = |state^{\hat{\mathcal{M}}}(\hat{s}), f|$ for all fluents f. $\hat{\mathcal{D}}$ is given by the signature of $\mathcal{D}$ where each action symbol a is substituted by some action symbol $\hat{a}$. Every pair of precondition and state update axiom

$$\begin{aligned}&\forall(s : Sit).\,(Poss(a, s) \Leftrightarrow Holds(St_a^{-} \circ St_a^{=}, s))\\&\forall(s : Sit).\,(Poss(a, s) \Rightarrow state(do(a, s)) \circ St_a^{-} =_{St} state(s) \circ St_a^{+})\end{aligned}$$

in $\mathcal{D}$ is substituted by

$$\begin{aligned}&\forall(s : Sit).\,(Poss(a, s) \Leftrightarrow Holds(St_a^{+} \circ St_a^{=}, s))\\&\forall(s : Sit).\,(Poss(a, s) \Rightarrow state(do(\hat{a}, s)) \circ St_a^{+} =_{St} state(s) \circ St_a^{-})\end{aligned}$$

From this and some observations given in the proof of Theorem 5.3.1 in this work, we conclude, the formula π is valid in $\mathcal{D}$ iff for every number n_f of fluents given by $|g_0^n, f| > n_f \geq |g_0^p, f|$ if f occurs in g_0^n and $n_f \geq |g_0^p, f|$ otherwise, exists a model $\hat{\mathcal{M}}$ of $\hat{\mathcal{D}}$ and situations $\hat{s}_e$, $s0^{\hat{\mathcal{M}}}$ such that

1. $s0^{\hat{\mathcal{M}}} \xrightarrow{\hat{a}_0} \hat{s}_1 \xrightarrow{\hat{a}_1} \cdots \xrightarrow{\hat{a}_{m-1}} \hat{s}_{m-1} \xrightarrow{\hat{a}_m} \hat{s}_e$ and $\hat{s}_e \in Sit^{\hat{\mathcal{M}}}$ for actions $\hat{a}_0, \ldots, \hat{a}_m \in Act^{\hat{\mathcal{M}}}$, and
2. $|state^{\hat{\mathcal{M}}}(\hat{s_e}), f| = n_f$ for all fluents f, and
3. $|g_e^n, f| > |state^{\hat{\mathcal{M}}}(s0^{\hat{\mathcal{M}}}), f| \geq |g_e^p, f|$ if f occurs in g_e^n, $|state^{\hat{\mathcal{M}}}(s0^{\hat{\mathcal{M}}}), f| \geq |g_e^p, f|$ otherwise.

The set of initial states described by the third condition may be not finite. To be able to reduce the above problem to a finite number of questions for Petri nets (where the initial marking corresponds to the initial state $state^{\hat{\mathcal{M}}}(s0^{\hat{\mathcal{M}}})$ of a particular model $\mathcal{M}$) we apply the idea of the Proof 5.3.1 and construct a new domain description $\hat{\mathcal{D}}'$ by adding a set of new actions and appropriate pairs of precondition and state update axioms of the form:

$$\begin{aligned}&\forall(s : Sit).\,(Poss(a, s) \Leftrightarrow \top)\\&\forall(s : Sit).\,(Poss(a, s) \Rightarrow state(do(\hat{a}_e^f, s)) =_{St} state(s) \circ f)\end{aligned}$$

for all $f \in G_e^n$ and $G_e^n = \{f \mid f \in Fl \wedge |g_e^n, f| = 0\}$. And finally, we investigate the (obviously finite) set Θ of Petri nets $\mathcal{P}(\hat{\mathcal{D}}', \hat{\mathcal{M}}')$, where $g = state^{\hat{\mathcal{M}}'}(s0^{\hat{\mathcal{M}}'})$ fulfils $|g_e^p, f| \leq |g, f| < |g_e^p, f| + 1$ for all fluents f. The state term g determines the initial marking of $\mathcal{P}(\hat{\mathcal{D}}', \hat{\mathcal{M}}')$. Then, π is valid iff the union of the reachability sets of Petri nets of Θ contains all markings corresponding to the second condition, i.e. all markings m such that $m(f) = n_f$ for all fluents f and all possible n_f. □

Corollary 5.4.2 *Let $\mathcal{D}$ be an arbitrary domain description in $\mathcal{FC}_{PL}$. Then, the validity of any formula of the form $\pi = \lambda_0 \Rightarrow EF\lambda_e$ where $\lambda_0, \lambda_e \in \Lambda_{L_\Sigma}$ is decidable.*

Proof 5.4.2 As for the planning problem, from Equation 2.7 it follows that any formula of Λ_{L_Σ} for $\mathcal{FC}_{PL}$ can be represented as a finite disjunction of formulas of Λ_{Holds}. Consequently, any extended planning problem for $\mathcal{FC}_{PL}$ domains can be represented by a finite set of instances of the problem $\pi = \lambda_0 \Rightarrow \mathrm{EF}\lambda_e$ where $\lambda_0, \lambda_e \in \Lambda_{Holds}$. These problems are decidable by Theorem 5.4.1. □

Example 5.4.1 (Airport continued) *For the airport domain of Example 2.5.4 all the queries of Example 3.4.1 are decidable.* □

5.5 Planning Problems in $\mathcal{FC}_L$

Unfortunately, as we show next for non-propositional Fluent Calculus domains the planning problem is undecidable.

Theorem 5.5.1 *Let $\mathcal{D}$ be a $\mathcal{FC}_L$ domain description wrt signature Σ, X some variable declaration and π be a planning problem, i.e. a formula of the form $\lambda_0 \wedge EF\lambda_e$ with $\lambda_0, \lambda_e \in \Lambda_{L_\Sigma(X)}$. The satisfiability of π is undecidable.*

We will prove the theorem by reducing the halting problem of deterministic two-counter machines to the above satisfiability problem. For deterministic two-counter machines the halting problem is undecidable [110]. Informally, a two-counter machine consists of two counters and a program which describes how the counters are decremented or incremented.

Definition 5.5.1 *A* two-counter machine *is a system $M = (2, Q, \delta, q_0, q_f)$, where Q is a nonempty finite set of internal states, $q_0 \in Q$ is an initial state, and $q_f \in Q$ is a final state. M uses $\{Z, P\}$ as a tape alphabet. δ is a move relation which is a subset of $Q \times \{1,2\} \times \{Z, P, -, 0, +\} \times Q$ (where "$-$", "0", and "$+$" denote left-shift, no-shift, and right-shift of a head, respectively). Tapes are one-way (rightward) infinite. The leftmost squares of the tapes contain the symbol "Z", and all other squares contain "P". Each element of δ is of the form $[q, i, s, q']$ where $q, q' \in Q$, $i \in \{1,2\}$, $s \in \{Z, P, -, 0, +\}$.*

A configuration *of M is a tuple $(q, n_1, n_2) \in Q \times \mathbb{N}^2$. The transition relation $\rightarrow_M$ over configurations of M is defined as follows:*

$$(q, n_1, n_2) \rightarrow_M (q', n_1', n_2) \quad \textit{and} \quad (q, n_1, n_2) \rightarrow_M (q', n_1, n_2')$$

iff one of the conditions 1-5 is satisfied.

1. $[q, i, Z, q'] \in \delta$ *and* $n_i = n_i' = 0$.
2. $[q, i, P, q'] \in \delta$ *and* $n_i = n_i' > 0$.
3. $[q, i, -, q'] \in \delta$ *and* $n_i - 1 = n_i'$.
4. $[q, i, 0, q'] \in \delta$ *and* $n_i = n_i'$.
5. $[q, i, +, q'] \in \delta$ *and* $n_i + 1 = n_i'$.

Let $c = (q, n_1, n_2)$ be a configuration and let $\Re(c, M)$ denote the set $\{c' \mid c \overset{*}{\rightarrow}_M c'\}$. c is called accepted *iff there exists $(q, n_1', n_2') \in \Re(c, M)$ such that $q = q_f$.*

M is called deterministic *if for all distinct elements $[q_1, i_1, x_1, q_1'], [q_2, i_2, x_2, q_2'] \in \delta$ holds*

$$q_1 = q_2 \Rightarrow (i_1 = i_2 \wedge x_1 \neq x_2 \wedge x_1, x_2 \in \{Z, P\})$$

A two-counter machine characterises a L_M-valued transition system. To this end we consider a set L_M of atomic propositions to consist of some propositions that allow us to define the acceptance condition[2].

$$L_M = Q \cup \{(q, n, \leq, i) \mid q \in Q \wedge n \in \mathbb{N} \wedge i \in \{1, 2\}\}$$

Then we define the mapping $\alpha : L_M \times \Re(c, M) \rightarrow \{\top, \bot\}$ accordingly such that $\alpha((q), (q', n_1', n_2')) = \top$ iff $q = q'$, $\alpha((q, n, \leq, i), (q', n_1', n_2')) = \top$ iff $q = q'$ and $n \leq n_i'$, for $i \in \{1, 2\}$.

Clearly, $K(M, L_M) = (\Re(c, M), \rightarrow_M, \delta, \alpha)$ is a L_M-valued transition system.

The halting problem for a two-counter machine M is the problem to decide whether a configuration $c = (q, n_1, n_2)$ is accepted by M. It can be expressed as the satisfiability of the formula

$$(q, n_1, \leq, 1) \wedge \neg(q, n_1 + 1, \leq, 1) \wedge (q, n_2, \leq, 2) \wedge \neg(q, n_2 + 1, \leq, 2) \wedge \mathrm{EF}(q_f) \tag{5.1}$$

To show that there exists a domain description such that satisfiability of π is undecidable we show that a deterministic two-counter machine M can be

[2] Our choice of L_M is based on its simplicity and its sufficient expressive power wrt a subset of L_Σ.

encoded into a domain description $\mathcal{D}(M)$ of $\mathcal{FC}_L$. To this end we consider the $\mathcal{FC}_L$ signature Σ_c which contains the domain specific sort N and the functions:
$\{f_q :\to Fl \mid q \in Q\}$; $f_1 : N \to Fl$; $f_2 : N \to Fl$;
$o_Z :\to N$; $o_P : N \to N$; $\{a_r : N \to Act \mid r \in \delta \wedge r \neq [q,i,Z,q']\}$; $\{a_r :\to Act \mid r \in \delta \wedge r = [q,i,Z,q']\}$.

1. For every $r = [q,i,Z,q'] \in \delta$ of the above form we add the axioms:
   $\forall(s : Sit).\,(Poss(a_r, s) \Leftrightarrow Holds(f_i(o_Z) \circ f_q, s))$
   $\forall(s : Sit).\,(Poss(a_r, s) \Rightarrow state(do(a_r, s)) \circ f_q =_{St} state(s) \circ f_{q'})$.
2. For every $r = [q,i,P,q'] \in \delta$ of the above form we add the axioms:
   $\forall(s : Sit),(x : N).\,(Poss(a_r(x), s) \Leftrightarrow Holds(f_i(o_P(x)) \circ f_q, s))$
   $\forall(s : Sit),(x : N).\,(Poss(a_r(x), s) \Rightarrow$
   $state(do(a_r(x), s)) \circ f_q =_{St} state(s) \circ f_{q'})$.
3. For every $r = [q,i,-,q'] \in \delta$ of the above form we add the axioms:
   $\forall(s : Sit),(x : N).\,(Poss(a_r(x), s) \Leftrightarrow Holds(f_i(o_P(x)) \circ f_q, s))$
   $\forall(s : Sit),(x : N).\,(Poss(a_r(x), s) \Rightarrow$
   $state(do(a_r(x), s)) \circ f_q \circ f_i(o_P(x)) =_{St} state(s) \circ f_{q'} \circ f_i(x))$.
4. For every $r = [q,i,0,q'] \in \delta$ of the above form we add the axioms:
   $\forall(s : Sit),(x : N).\,(Poss(a_r(x), s) \Leftrightarrow Holds(f_q, s))$
   $\forall(s : Sit),(x : N).\,(Poss(a_r(x), s) \Rightarrow$
   $state(do(a_r(x), s)) \circ f_q =_{St} state(s) \circ f_{q'})$.
5. For every $r = [q,i,+,q'] \in \delta$ of the above form we add the axioms:
   $\forall(s : Sit),(x : N).\,(Poss(a_r(x), s) \Leftrightarrow Holds(f_i(x) \circ f_q, s))$
   $\forall(s : Sit),(x : N).\,(Poss(a_r(x), s) \Rightarrow$
   $state(do(a_r(x), s)) \circ f_q \circ f_i(x) =_{St} state(s) \circ f_{q'} \circ f_i(o_P(x)))$.

Proof 5.5.1 To prove soundness of the above embedding we show that for any deterministic two-counter machine M with initial configuration $c_0 = (q^0, n_1^0, n_2^0)$ and the model $\mathcal{M}$ of $\mathcal{D}(M)$ with

$$state^{\mathcal{M}}(s0^{\mathcal{M}}) = f_{q^0} \circ f_1(\underbrace{o_P(\ldots o_P(o_Z)\ldots)}_{n_1^0}) \circ f_2(\underbrace{o_P(\ldots o_P(o_Z)\ldots)}_{n_2^0})$$

$K(\mathcal{M}, L'_{\Sigma_c}(X))$ and $K(M, L_M)$ are bisimilar.
Thereby we define the subset $L'_{\Sigma_c}(X)$ of $L_{\Sigma_c}(X)$ as follows

$$\begin{aligned} L'_{\Sigma_c} = \; & \{Holds(f_q) \mid q \in Q\} \\ & \{Holds(f_i(t) \circ f_q) \mid q \in Q \wedge i \in \{1,2\} \wedge t \in T_{N,\Sigma_c}(1^\circ)\} \\ & \{\exists(x : N).\,Holds(f_i(\underbrace{o_P(\ldots(o_P}_{n}(x))\ldots)) \circ f_q) \\ & \qquad \mid q \in Q \wedge i \in \{1,2\} \wedge n \in \mathbb{N}\} \\ & \{\exists(x,y : N).\,Holds(f_i(x) \circ f_i(y)) \mid i \in \{1,2\}\} \end{aligned}$$

The mappings $\Psi_{\mathcal{M}} : L'_{\Sigma_c} \to L_M$ and $\Psi_M : L_M \to L'_{\Sigma_c}$ between propositions of $K(\mathcal{M}, L'_{\Sigma_c})$ and $K(M, L_M)$ for $i \in \{1,2\}$ and $q \in Q$:

$$\begin{aligned}
\Psi_{\mathcal{M}}(\mathit{Holds}(f_q)) &= (q),\\
\Psi_{\mathcal{M}}(\exists(x:N).\,\mathit{Holds}(f_i(\underbrace{o_P(\ldots(o_P}_{n}(x))\ldots)) \circ f_q)) &= (q,n,\leq,i)\\
\Psi_{\mathcal{M}}(\exists(x,y:N).\,\mathit{Holds}(f_i(x) \circ f_i(y))) &= \bot
\end{aligned}$$

$$\begin{aligned}
\Psi_M(q) &= \mathit{Holds}(f_q),\\
\Psi_M(q,n,\leq,i) &= \exists(x:N).\,\mathit{Holds}(f_i(\underbrace{o_P(\ldots(o_P}_{n}(x))\ldots)) \circ f_q)
\end{aligned}$$

Clearly $\Psi_{\mathcal{M}}$ and Ψ_M are bijections and $\Psi_{\mathcal{M}} = \Psi_M^{-1}$ except for the mapping of $\exists(x,y:N).\,\mathit{Holds}(f_i(x) \circ f_i(y))$ for $i \in \{1,2\}$. As we will show in the following, in a model where $\mathit{state}^{\mathcal{M}}(s0^{\mathcal{M}})$ does not contain several copies of $f_i(\_)$ there is no state associated with a subsequent situation of $s0^{\mathcal{M}}$ which contains several copies of $f_i(\_)$. Consequently, by restricting to only those models any proposition of the form $\exists(x,y:N).\,\mathit{Holds}(f_i(x) \circ f_i(y))$ is assigned $\bot$ in every state. The appropriate restriction of the set of models is expressed in the query corresponding to the halting problem.

Furthermore, let $c = (q, n_1, n_2)$ be a configuration and

$$\mathit{state}^{\mathcal{M}}(c) = f_q \circ f_1\underbrace{(o_P(\ldots o_P(o_Z)\ldots)}_{n_1} \circ f_2\underbrace{(o_P(\ldots o_P(o_Z)\ldots)}_{n_2}.$$

Then, from the definition of *Holds* follows for any $p \in L_M$, $\alpha(p,c) = \top$ iff $\alpha(\Psi_M(p), \mathit{state}^{\mathcal{M}}(c)) = \top$. And, vice versa, for any $p \in L'_{\Sigma_c}$, $\alpha(p, \mathit{state}^{\mathcal{M}}(c)) = \top$ iff $\alpha(\Psi_{\mathcal{M}}(p), c) = \top$.

Since $K(\mathcal{M}, L'_{\Sigma_c})$ and $K(M, L_M)$ are rooted in $m_0(s0^{\mathcal{M}})$, it suffices to show the existence of a bisimulation Φ with $(s0^{\mathcal{M}}, (q^0, n_1^0, n_2^0)) \in \Phi$.

Now let $(s,c) \in \Phi \subseteq \mathit{Sit}^{\mathcal{M}} \times \Re(c, M)$ such that $s \in \mathit{Sit}^{\mathcal{M}}(c)$ where $\mathit{Sit}^{\mathcal{M}}(c)$ denotes the set of all situations s where $\mathit{state}^{\mathcal{M}}(s) = \mathit{state}^{\mathcal{M}}(c)$. We prove that for every transition $t \in \delta$ and $c \xrightarrow{t}_M c'$, there exists a situation s' such that $\mathit{state}^{\mathcal{M}}(s') = \mathit{state}^{\mathcal{M}}(c')$ and $s \xrightarrow{a_t}_{\mathcal{M}} s'$ for all $s \in \mathit{Sit}^{\mathcal{M}}(c)$. To this end we investigate all cases of the transition relation in Definition 5.5.1 with $c = (q, n_1, n_2)$:

1. Let $r = [q, i, Z, q']$ and $c \xrightarrow{r}_M c'$. Then $n_i = 0$ and $n'_i = 0$ and $\mathit{state}^{\mathcal{M}}(c)$ contains $f_i(o_Z)$ and f_q, i.e. $\mathcal{M} \models \mathit{Holds}(f_i(o_Z) \circ f_q, s)$. Furthermore, a_r is applicable and $\mathit{state}^{\mathcal{M}}(do^{\mathcal{M}}(a_r, s))$ contains $f_{q'}$ and $f_1(t_1)$ and $f_2(t_2)$ unchanged (for terms t_1, t_2, but not f_q. Consequently, $s' = do^{\mathcal{M}}(a_r, s)$ and $\mathit{state}^{\mathcal{M}}(s') =_{St} \mathit{state}^{\mathcal{M}}(c')$.

2. Let $r = [q, i, P, q']$ and $c \xrightarrow{r}_M c'$. Then $n_i > 0$ and $n'_i = n_i$ and $state^{\mathcal{M}}(c)$ contains $f_i(o_P(t))$ (for some term t), i.e. $\mathcal{M} \models Holds(f_i(o_P(t)) \circ f_q, s)$. Furthermore, $a_r(t)$ is applicable and $state^{\mathcal{M}}(do^{\mathcal{M}}(a_r(t)), s)$ contains $f_{q'}$ and $f_1(t_1)$ and $f_2(t_2)$ unchanged (for some terms t_1 and t_2), but not f_q. Consequently, $s' = do^{\mathcal{M}}(a_r(t), s)$ and $state^{\mathcal{M}}(s') =_{St} state^{\mathcal{M}}(c')$.

3. Let $r = [q, i, -, q']$ and $c \xrightarrow{r}_M c'$. Then $n_i > 0$ and $n'_i = n_i - 1$ and $state^{\mathcal{M}}(c)$ contains $f_i(o_P(t))$ (for some term t), i.e. $\mathcal{M} \models Holds(f_i(o_P(t)) \circ f_q, s)$. Furthermore, $a_r(t)$ is applicable and $state^{\mathcal{M}}(do^{\mathcal{M}}(a_r(t), s))$ contains $f_{q'}$ and $f_1(t)$ ($f_2(t)$) and $f_2(t_2)$ ($f_1(t_1)$, respectively) unchanged (for some terms t_1 and t_2), but not f_q. Consequently, $s' = do^{\mathcal{M}}(a_r(t), s)$ and $state^{\mathcal{M}}(s') =_{St} state^{\mathcal{M}}(c')$.

4. Let $r = [q, i, 0, q']$ and $c \xrightarrow{r}_M c'$. Then $n'_i = n_i$ and $state^{\mathcal{M}}(c)$ contains f_q, i.e. $\mathcal{M} \models Holds(f_q, s)$. Furthermore, $a_r(t_i)$ is applicable and $state^{\mathcal{M}}(do^{\mathcal{M}}(a_r(t_i), s))$ contains $f_{q'}$ and $f_1(t_1)$ and $f_2(t_2)$ unchanged (for some terms t_1 and t_2), but not f_q. Consequently, $s' = do^{\mathcal{M}}(a_r(t_i), s)$ and $state^{\mathcal{M}}(s') =_{St} state^{\mathcal{M}}(c')$.

5. Let $r = [q, i, -, q']$ and $c \xrightarrow{r}_M c'$. Then $n'_i = n_i + 1$ and $state^{\mathcal{M}}(c)$ contains $f_i(t)$ (for some term t), i.e. $\mathcal{M} \models Holds(f_i(t) \circ f_q, s)$. Furthermore, $a_r(t)$ is applicable and $state^{\mathcal{M}}(do^{\mathcal{M}}(a_r(t), s))$ contains $f_{q'}$ and $f_1(o_P(t))$ ($f_2(o_P(t))$) and $f_2(t_2)$ ($f_1(t_1)$, respectively) unchanged (for some terms t_1 and t_2), but not f_q. Consequently, $s' = do^{\mathcal{M}}(a_r(t), s)$ and $state^{\mathcal{M}}(s') =_{St} state^{\mathcal{M}}(c')$.

Analogously, let $c_{\mathcal{M}}(s)$ for $s \in Sit^{\mathcal{M}}$ denote the configuration (q, n^1, n^2) such that

$$state^{\mathcal{M}}(s) =_{St} f_q \circ f_1(\underbrace{o_P(\ldots o_P(o_Z)\ldots)}_{n^1}) \circ f_2(\underbrace{o_P(\ldots o_P(o_Z)\ldots)}_{n^2}).$$

Then for every action $a \in Act^{\mathcal{M}}$ and $s \xrightarrow{a}_{\mathcal{M}} s'$, there exists $c' \in \Re(c_0, M)$ and $t \in \delta$ such that $c_{\mathcal{M}}(s') = c'$ and $c \xrightarrow{t}_M c'$. This follows from the one–to–one mapping of δ to $T_{Act,\Sigma}(\emptyset)$ and from the fact that Ψ_M and $\Psi_{\mathcal{M}}$ are bijections. Furthermore, from the definition of the state update axioms it follows that there exists no situation where two distinct actions may be both executable, hence there is always a unique transition $t \in \delta$ associated with the execution of some action a: by definition, the state associated with the initial situation contains exactly one fluent f_q with $q \in Q$. In every state update axiom exactly one fluent f_q is removed and exactly one fluent $f_{q'}$ with $q, q' \in Q$ is added. By induction it holds that every state associated with a reachable situations contains precisely one fluent f_q with $q \in Q$. Consequently, only actions where the precondition axioms contain the same fluent f_q ($q \in Q$) may be executable simultaneously. By definition of deterministic two-counter machines and their embedding in

$\mathcal{FC}_L$ two actions a_1, a_2 may only share f_q in the precondition axiom if the precondition axiom of a_1 contains additionally $f_i(o_Z)$ while the precondition axiom of a_2 contains additionally $f_i(o_P(x))$ with $i \in \{1,2\}$. Since there is no ground term t of sort $St^{\mathcal{M}}$ such that $f_i(o_Z)$ and $f_i(o_P(x))$ are unifiable with t, a_1 must correspond to the transition $[q,i,Z,q'] \in \delta$ for some $q' \in Q$, and a_2 must correspond to the transition $[q,i,P,q'] \in \delta$ for some $q' \in Q$.

For $(s0^{\mathcal{M}}, c_0)$ holds $c_{\mathcal{M}}(s0^{\mathcal{M}}) = c_0$. By induction and the above relations it follows that $s0^{\mathcal{M}} \sim c_0$.

Finally, the halting problem of two-counter machines can be encoded as a planning problem for $\mathcal{D}(M)$ using the mapping Ψ_M:

$$\begin{aligned}
&\mathit{Holds}(f_1(\underbrace{o_P(\ldots(o_P}_{n_1}(x))\ldots)) \circ f_q) \wedge \\
&\neg\mathit{Holds}(f_1(\underbrace{o_P(\ldots(o_P}_{n_1+1}(x))\ldots)) \circ f_q) \wedge \neg\exists(x,y:N).\,\mathit{Holds}(f_1(x) \circ f_1(y)) \wedge \\
&\mathit{Holds}(f_2(\underbrace{o_P(\ldots(o_P}_{n_2}(x))\ldots)) \circ f_q) \wedge \\
&\neg\mathit{Holds}(f_2(\underbrace{o_P(\ldots(o_P}_{n_2+1}(x))\ldots)) \circ f_q) \wedge \neg\exists(x,y:N).\,\mathit{Holds}(f_2(x) \circ f_2(y)) \wedge \\
&\textstyle\bigwedge_{r \in Q \wedge r \neq q} \neg\mathit{Holds}(f_r) \wedge \mathrm{EF}\mathit{Holds}(f_{q_f})
\end{aligned}$$

Thereby the negative sub-formulas[3] ensure that the initial state does not contain $f_1(\_)$, $f_2(\_)$ multiple times, that the fluents $f_1(\_)$, $f_2(\_)$ are uniquely determined, and that f_q is the only fluent representing an element of Q. □

Corollary 5.5.2 *Let $\mathcal{D}$ be a $\mathcal{FC}_L$ domain description wrt signature Σ, X some variable declaration and let π be a formula describing a conjunctive planning problem, i.e. π is of the form $\lambda_0 \wedge EF\lambda_e$ with $\lambda_0 \in \Lambda_{L_\Sigma(X)}$ and $\lambda_e = l_1 \wedge \cdots \wedge l_n$ with $l_1, \ldots, l_n \in L_\Sigma(X)$. The satisfiability of π is undecidable.*

Proof 5.5.2 This follows simply from the fact that the planning problem of the above proof is also a conjunctive planning problem with $\lambda_e = \mathit{Holds}(f_{q_f})$. □

Corollary 5.5.3 *Let $\mathcal{D}$ be a $\mathcal{FC}_L$ domain description wrt signature Σ, X some variable declaration and let π be a formula describing an extended planning problem, i.e. π is of the form $\lambda_0 \Rightarrow EF\lambda_e$ with $\lambda_0, \lambda_e \in \Lambda_{L_\Sigma(X)}$. The validity of π is undecidable.*

[3]Note that instead of allowing atomic propositions based on $\mathit{Holds}(\_)$ only, we may additionally consider state equations in L'_{Σ_c}. Then we may simply write e.g. $\mathit{state}(s) =_{St} f_1(o_P(o_Z)) \circ f_2(o_Z) \circ f_q$ to define the initial state. It would be interesting to investigate whether such expressions actually increase the expressive power of L_Σ.

Proof 5.5.3 Note that in the planning problem representing the halting problem of Proof 5.5.1 the initial state is completely specified. Hence, we may state the halting problem for two-counter machines also as the validity of

$$(q, n_1, \leq, 1) \wedge \neg(q, n_1 + 1, \leq, 1) \wedge (q, n_2, \leq, 2) \wedge \neg(q, n_2 + 1, \leq, 2) \Rightarrow \mathrm{EF}(q_f),$$

Using the mapping Ψ_M we can transform this formula into a formula π. Deciding the validity of π is an extended planning problem. Hence, from the proof of undecidability of the planning problem for $\mathcal{FC}_L$ domains follows also the undecidability of the extended planning problem for $\mathcal{FC}_L$ domains. □

Furthermore, in the Proof 5.5.1 the state term associated with $s0^{\mathcal{M}}$ may not contain any fluent more than once. Since none of the state update axioms may increase the number of copies of some fluent to become greater than one, it follows that the axiom (NM) of Definition 2.5.3 is fulfilled in all models of interest of domain $\mathcal{D}(M)$. Consequently, the planning problem as well as the extended planning problem are undecidable even if $\mathcal{FC}_L$ is additionally restricted by axiom (NM).

5.6 Planning Problems in $\mathcal{FC}_{PLN}$

It has been shown that the use of negation in precondition axioms is already sufficient to describe computationally complete systems:

Proposition 5.6.1 *Let $\mathcal{D}$ be a $\mathcal{FC}_{PLN}$ domain description wrt signature Σ, X some variable declaration and π be a planning problem, i.e. a formula of the form $\lambda_0 \wedge EF\lambda_e$ with $\lambda_0, \lambda_e \in \Lambda_{L_\Sigma(X)}$. The satisfiability of π is undecidable.*

A proof of this theorem is given in [67]. Similarly to the proof of Theorem 5.5.1 it relies on the reduction of the halting problem of two-counter machines to $\mathcal{FC}_{PLN}$ domains. Since only constant fluents are allowed in $\mathcal{FC}_{PLN}$ the counter value $n_i \in \mathbb{N}$ with $i \in \{1, 2\}$ is represented by a term $\underbrace{f_i \circ \cdots \circ f_i}_{n_i}$.

Incrementing and decrementing of the counter is represented as adding and removing of a single fluent f_i, respectively. In contrast to $\mathcal{FC}_{PL}$ domains the test of the counter for zero can be expressed using negation: $\neg Holds(f_i, s)$. Consequently, this feature can be understood as being responsible for the expressive power gained by $\mathcal{FC}_{PLN}$ wrt $\mathcal{FC}_{PL}$.

Note that for the same reason as for $\mathcal{FC}_L$ domains the extended planning problem and the conjunctive planning problem are both undecidable for $\mathcal{FC}_{PLN}$ domains as well.

The simple Fluent Calculus with specificity

As has been argued in Section 2.8, $\mathcal{FC}_{LN}$ is more expressive than the simple Fluent Calculus with specificity relation $\mathcal{FC}_{L\leq}$ of [70]. Furthermore, for $\mathcal{FC}_{PLN}$, the propositional version of $\mathcal{FC}_{LN}$, the planning problem has been shown to be undecidable. On the other hand, it is clear from Section 5.5 that since any two-counter machine can be represented as a $\mathcal{FC}_{L}$ domain, it can also be represented as a domain of the simple Fluent Calculus with specificity and, hence, the planning problem is undecidable in $\mathcal{FC}_{L\leq}$ as well. However, for $\mathcal{FC}_{PL}$, the propositional version of $\mathcal{FC}_{L}$, we have shown in Section 5.3 that the planning problem is in fact decidable. The question arises whether the planning problem is also decidable in the propositional version $\mathcal{FC}_{PL\leq}$ of $\mathcal{FC}_{L\leq}$ (we define $\mathcal{FC}_{PL\leq}$ domains simply as $\mathcal{FC}_{L\leq}$ domains where all fluents and actions are constants). In the following we will show that this is not the case. To this end we will represent $\mathcal{FC}_{PLN}$ domains as $\mathcal{FC}_{PL\leq}$ domains. We will show bisimularity between the transition systems associated with models of the $\mathcal{FC}_{PLN}$ domains and restricted versions of the transition systems associated with models of the $\mathcal{FC}_{L\leq}$ domains. Such a restricted version results from removing all situations s from a transition system where a particular fluent (f_c) does not occur in $state(s)$. Then we show that every formula describing a planning problem for a $\mathcal{FC}_{PLN}$ domain $\mathcal{D}$ can be extended to characterise the above restriction. Since it turns out that this extended formula describes a planning problem for the $\mathcal{FC}_{PL\leq}$ domain that corresponds to $\mathcal{D}$, it follows that every planning problem for a $\mathcal{FC}_{PLN}$ domain can be represented as a planning problem for a $\mathcal{FC}_{PL\leq}$ domain.

We define a $\mathcal{FC}_{PL\leq}$ domain to be a $\mathcal{FC}_{L\leq}$ domain (see Definition 2.8.2) where all fluents and actions are constants. Now we describe how $\mathcal{FC}_{PLN}$ domains are translated into $\mathcal{FC}_{PL\leq}$ domains.

Definition 5.6.1 ($\mathcal{FC}_{PLN} \to \mathcal{FC}_{PL\leq}$) *Let $\mathcal{D}$ be a domain description in $\mathcal{FC}_{PLN}$ wrt some signature Σ. We define $\Sigma_{\leq}$ as the signature Σ together with the additional definition $f_c :\to Fl$ where f_c does not appear in Σ. Furthermore, we define $\mathcal{D}_{\leq}$ to denote a $\mathcal{FC}_{PL\leq}$ domain wrt $\Sigma_{\leq}$, such that for every action a and every pair of precondition and state update axioms of $\mathcal{D}$ of the form*

$$\begin{array}{l}\forall(s : Sit).\,(Poss(a,s) \Leftrightarrow Holds(St_a^- \circ St_a^=, s) \wedge \overline{Holds(St_a^n, s)})\\ \forall(s : Sit).\,(Poss(a,s) \Rightarrow state(do(a,s),s) \circ St_a^- =_{St} state(s) \circ St_a^+)\end{array} \qquad (5.2)$$

where for those fluents $f_1,\ldots,f_k$ of Σ with $|St_a^n, f_i| > 0$ ($i = 1,\ldots,k$) $n_i = |St_a^n, f_i| - |St_a^- \circ St_a^=, f_i| > 0$ also holds, $\mathcal{D}_{\leq}$ contains the following $2^k + 1$ pairs *of axioms:*

$$\begin{array}{l}
\forall(s:Sit).\,(Poss(a,s) \Leftrightarrow Holds(f_c \circ St_a^- \circ St_a^=, s)) \\
\forall(s:Sit).\,(Poss(a,s) \Rightarrow state(do(a,s),s) \circ St_a^- =_{St} state(s) \circ St_a^+) \\
\forall(s:Sit).\,(Poss(a,s) \Leftrightarrow Holds(f_c \circ St_a^- \circ St_a^= \circ f_1^{n_1}, s)) \\
\forall(s:Sit).\,(Poss(a,s) \Rightarrow state(do(a,s),s) \circ f_c =_{St} state(s)) \\
\vdots \\
\forall(s:Sit).\,(Poss(a,s) \Leftrightarrow Holds(f_c \circ St_a^- \circ St_a^= \circ f_1^{n_1} \circ \cdots \circ f_k^{n_k}, s)) \\
\forall(s:Sit).\,(Poss(a,s) \Rightarrow state(do(a,s),s) \circ f_c =_{St} state(s))
\end{array} \tag{5.3}$$

The idea behind this mapping is based on viewing a statement like $\overline{Holds(St_a^n, s)}$ as a set of exceptions for the executability of action a. For every fluent f_i occuring in St_a^n with $n_i > 0$, $\overline{Holds(St_a^n, s)}$ describes one exception, i.e. the upper bound $|St_a^n, f_i|$ of copies of fluent f_i that must not be reached. Note that if $n_i \leq 0$ for some fluent f_i then $Poss(a,s)$ is not valid for every situation s and the action a is never executable. In $\mathcal{FC}_{PL\leq}$ the case where no exception applies is represented by the precondition axiom which contains only the $Holds(St_a^- \circ St_a^=, s)$ statement of the corresponding precondition axiom of the $\mathcal{FC}_{PLN}$ domain. For every exception this axiom is overruled by another more specific precondition axiom. Since several exceptions may apply simultaneously we have to ensure that there is always one unique most specific precondition axiom for every combination of exceptions. However, in $\mathcal{FC}_{L\leq}$ (and $\mathcal{FC}_{PL\leq}$) domains it is not possible to prevent the executability of an action (see Section 2.8) explicitly. To label the situations that are not reachable in models of the $\mathcal{FC}_{PLN}$ domain we introduce an additional fluent f_c which is removed from a state whenever an exceptions has applied. Then, for every situation in a model for the $\mathcal{FC}_{PLN}$ domain there exists a situation "labeled" by f_c in the corresponding model of the $\mathcal{FC}_{PL\leq}$ domain.

Proposition 5.6.2 *Let $\mathcal{D}$ be a $\mathcal{FC}_{PLN}$ domain description and $\mathcal{M}$ be a model of $\mathcal{D}$. Let $\mathcal{M}_\leq$ denote the model of the $\mathcal{FC}_{PL\leq}$ domain $\mathcal{D}_\leq$ where for all $f \in Fl^{\mathcal{M}}$ $|state^{\mathcal{M}_\leq}(s0^{\mathcal{M}_\leq}), f| = |state^{\mathcal{M}}(s0^{\mathcal{M}}), f|$ and $|state^{\mathcal{M}_\leq}(s0^{\mathcal{M}_\leq}), f_c| = 1$. Let $K_c(\mathcal{M}_\leq, L_{\Sigma_\leq})$ denote the transition system which is the result of removing all those situations s from $K(\mathcal{M}_\leq, L_{\Sigma_\leq})$ where $state^{\mathcal{M}_\leq}(s)$ does not contain the fluent f_c. Then $K_c(\mathcal{M}_\leq, L_{\Sigma_\leq})$ and $K(\mathcal{M}, L_\Sigma)$ are bisimilar.*

Proof 5.6.2 Note first that none of the actions of $\mathcal{D}$ is executable in some situation s if $state^{\mathcal{M}_\leq}(s)$ does not contain f_c. Consequently, those situations are leaves of $K(\mathcal{M}_\leq, L_{\Sigma_\leq})$ and, hence, $K_c(\mathcal{M}_\leq, L_{\Sigma_\leq})$ also forms a tree.

Furthermore, there may be several pairs of axioms describing preconditions and effects of some action a. According to the specificity relation the first pair is chosen in some situation s iff $Holds(f_c \circ St_a^- \circ St_a^=, s)$ is satisfied but not $Holds(f_c \circ$

$St_a^- \circ St_a^= \circ f_{i_1}^{n_{i_1}} \circ \cdots \circ f_{i_m}^{n_{i_m}}, s)$ for any $\{i_1, \ldots, i_m\} \subseteq \{1, \ldots, k\}$, i.e. iff with $n_i = |St^- a \circ St_a^=, f_i| + |St_a^n, f_i|$, $i = 1, \ldots, k$

$$Holds(f_c \circ St_a^- \circ St_a^=) \wedge \overline{Holds(f_1^{n_1} \circ \cdots \circ f_k^{n_k}, s)} \tag{5.4}$$

is satisfied. Another pair is chosen iff $Holds(f_c \circ St_a^- \circ St_a^=, s)$ but Proposition 5.4 is not satisfied. In this case the chosen pair is unique since for any set of precondition axioms

$$\begin{array}{l} \forall (s : Sit).\, (Poss(a, s) \Leftrightarrow Holds(f_c \circ St_a^- \circ St_a^= \circ F_1, s)), \\ \vdots \\ \forall (s : Sit).\, (Poss(a, s) \Leftrightarrow Holds(f_c \circ St_a^- \circ St_a^= \circ F_n, s)) \end{array} \tag{5.5}$$

that are satisfied in s there exists another pair of axioms with precondition

$$\forall (s : Sit).\, (Poss(a, s) \Leftrightarrow Holds(f_c \circ St_a^- \circ St_a^= \circ F_1 \circ \cdots \circ F_n, s))$$

which is satisfied and which overrules each precondition of the Set 5.5.

We consider the following mappings $\Psi_{\Sigma_\leq} : L_{\Sigma_\leq} \to L_\Sigma$ and $\Psi_\Sigma : L_\Sigma \to L_{\Sigma_\leq}$ between propositions of $K_c(\mathcal{M}_\leq, L_{\Sigma_\leq})$ and $K(\mathcal{M}, L_\Sigma)$ for any $g \in St_\mathcal{M}$, $m > 1$:

$$\begin{array}{rcl} \Psi_{\Sigma_\leq}(Holds(g)) & = & Holds(g), \\ \Psi_{\Sigma_\leq}(Holds(f_c \circ g)) & = & Holds(g), \\ \Psi_{\Sigma_\leq}(Holds(f_c^m \circ g)) & = & \bot \\ \Psi_\Sigma(Holds(g)) & = & Holds(g) \end{array}$$

Note that $Holds(f_c \circ g, s)$ is satisfied in a situation s of $K_c(\mathcal{M}_\leq, L_\Sigma)$ iff $Holds(g, s)$ is satisfied. This is due to the fact that all states associated with situations of $K_c(\mathcal{M}_\leq, L_{\Sigma_\leq})$ contain precisely one copy of f_c. As another consequence, $Holds(f_c^m \circ g, s)$ is not satisfied in $K_c(\mathcal{M}_\leq, L_{\Sigma_\leq})$ for any $m > 1$. Since $\Psi_{\Sigma_\leq}$ respects these relations, $\alpha(\Psi_{\Sigma_\leq}(p), s) = \top$ iff $\alpha(p, s) = \top$ for all $p \in L_{\Sigma_\leq}$ and, clearly, $\alpha(\Psi_\Sigma(p), state^{\mathcal{M}_\leq}(s)) = \top$ iff $\alpha(p, state^\mathcal{M}(s)) = \top$ for all $p \in L_\Sigma$.

Since $K_c(\mathcal{M}_\leq, L_{\Sigma_\leq})$ and $K(\mathcal{M}, L_\Sigma)$ are rooted in $s0^{\mathcal{M}_\leq}$ and $s0^\mathcal{M}$, respectively, it suffices to show the existence of a bisimulation Φ with $(s0^{\mathcal{M}_\leq}, s0^\mathcal{M}) \in \Phi$.

We denote by $Sit^\mathcal{M}(s_\leq)$ for $s_\leq \in Sit^{\mathcal{M}_\leq}$ the set of all situations $s \in Sit^\mathcal{M}$ such that for all $f \in Fl^\mathcal{M}$ $|state^\mathcal{M}(s), f| = |state^{\mathcal{M}_\leq}(s_\leq), f|$.

Now let $(s, s_\leq) \in \Phi \subseteq Sit^\mathcal{M} \times Sit^{\mathcal{M}_\leq}$ such that $s \in Sit^\mathcal{M}(s_\leq)$. We prove that for every action $a \in Act^\mathcal{M}$ and $s_\leq \xrightarrow{a}_{\mathcal{M}_\leq} s'_\leq$, there exists a situation s' such that $s \xrightarrow{a}_\mathcal{M} s'$ and for all $f \in Fl^\mathcal{M}$ $|state^\mathcal{M}(s'), f| = |state^\mathcal{M}(s'_\leq), f|$. Since the Formula 5.4 is satisfied in some situation s and some action a iff the precondition of Axioms 5.2 is satisfied, we have the following cases:

1. Formula 5.4 is satisfied and, hence, the first pair of Axioms 5.3 is valid. Consequently, $s_{\leq} \xrightarrow{a}_{\mathcal{M}_{\leq}} s'_{\leq}$ for some situation $s_{\leq}$. Due to Axioms 5.2, $s \xrightarrow{a}_{\mathcal{M}} s'$ for some s'. Furthermore for all $f \in Fl^{\mathcal{M}}$ holds

$$\begin{aligned}
|state^{\mathcal{M}}(s'), f| &= |state^{\mathcal{M}}(s), f| - |St_a^-, f| + |St_a^+, f| \\
&= |state^{\mathcal{M}_{\leq}}(s_{\leq}), f| - |St_a^-, f| + |St_a^+, f| \\
&= |state^{\mathcal{M}_{\leq}}(s'_{\leq}), f|.
\end{aligned}$$

2. Formula 5.4 is not satisfied, but (at least) one of the preconditions of the Axioms 5.3 is satisfied. Then a is executable in $s_{\leq}$ of $K(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$, but $|state^{\mathcal{M}_{\leq}}(s'_{\leq}), f_c| = |state^{\mathcal{M}_{\leq}}(s_{\leq}), f_c| - 1 = 0$. Hence, by definition of $K_c(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$, a is not executable in $s_{\leq}$ of $K_c(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$.

3. Formula 5.4 is not satisfied and none of the preconditions of the Axioms 5.3 is satisfied. Then a is not executable in $K_c(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$ nor in $K(\mathcal{M}, L_{\Sigma})$.

On the other hand, we denote by $Sit^{\mathcal{M}}(s)$ for $s \in Sit^{\mathcal{M}}$ the set of all situations $s_{\leq} \in Sit^{\mathcal{M}_{\leq}}$ such that for all $f \in Fl^{\mathcal{M}}$ $|state^{\mathcal{M}_{\leq}}(s_{\leq}), f| = |state^{\mathcal{M}}(s), f|$.

Then, for every action $a \in Act^{\mathcal{M}_{\leq}}$ and $s \xrightarrow{a}_{\mathcal{M}} s'$, there exists a situation $s'_{\leq}$ such that $s_{\leq} \xrightarrow{a}_{\mathcal{M}} s'_{\leq}$ and for all $f \in Fl^{\mathcal{M}}$ $|state^{\mathcal{M}_{\leq}}(s'_{\leq}), f| = |state^{\mathcal{M}}(s'), f|$. This is a consequence of

1. the fact that every state associated with some situation s of $K_c(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$ contains precisely one copy of f_c,

2. the above mentioned correspondence between Formula 5.4 and the precondition of Axioms 5.2, and,

3. the relation between successor states of Case 1,

By induction and the above relations follows $s0^{\mathcal{M}} \sim s0^{\mathcal{M}_{\leq}}$ for $K(\mathcal{M}, L_{\Sigma})$ and $K_c(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$. □

Now, to show that every planning problem $\phi = \lambda_0 \wedge \text{EF}\lambda_e$ of domain $\mathcal{D}$ can also be represented as a planning problem ϕ' of $\mathcal{D}_{\leq}$, we have to find a CTL_U formula that reflects both, the planning problem of domain $\mathcal{D}$ and the restriction of the considered situations of $K(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$ to those that appear also in $K_c(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$. Since for every model $\mathcal{M}$ of $\mathcal{D}$ exists a model $\mathcal{M}_{\leq}$ such that $K(\mathcal{M}, L_{\Sigma}) \sim K_c(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$ and with Ψ_{Σ} (we apply Ψ_{Σ} on a formula λ by substituting every atomic proposition p occurring in λ by $\Psi_{\Sigma}(p)$):

$$K(\mathcal{M}, L_{\Sigma}) \models \lambda_0 \wedge \text{EF}\lambda_e \quad \text{iff} \quad K_c(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}}) \models \Psi_{\Sigma}(\lambda_0) \wedge \text{EF}\Psi_{\Sigma}(\lambda_e)$$

However, the formula ϕ' has to take the following into account:

1. The formula ϕ' should only be satisfied by those models $\mathcal{M}$ of $\mathcal{D}_{\leq}(\mathcal{D})$ where $|state^{\mathcal{M}_{\leq}}(s0^{\mathcal{M}_{\leq}}), f_c| = 1$. This is to ensure that $K_c(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$ is defined and every branch of $K_c(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$ is also a branch in $K(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$.
2. The formula ϕ' should only be satisfied by those paths π of $K(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$ where each state associated with some situation on π contains f_c. Clearly, every such path of $K(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$ is also a path of $K_c(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$. Only the state associated with the last situation on a path may not contain f_c.

A formula which fulfils these requirements is given as follows.

$$\phi' = \Psi_{\Sigma}(\lambda_0) \wedge \mathit{Holds}(f_c) \wedge \overline{\mathit{Holds}(f_c^2)} \wedge \mathrm{EF}(\Psi_{\Sigma}(\lambda_e) \wedge \mathit{Holds}(f_c))$$

The additional statement $\mathit{Holds}(f_c) \wedge \overline{\mathit{Holds}(f_c^2)}$ ensures that the initial situation is "labeled" by precisely one copy of f_c, i.e. it represents indeed a situation of some model $\mathcal{M}$ of the $\mathcal{FC}_{PLN}$ domain. The statement $\mathit{Holds}(f_c)$ after the temporal operator EF ensures that the final situation represents also a situation of $\mathcal{M}$. Since f_c may only disappear at a leaf of $K(\mathcal{M}_{\leq}, L_{\Sigma_{\leq}})$, all situations on the path from the initial situation to the final situation represent situations of $\mathcal{M}$. Hence, the formula ϕ' is satisfied in $\mathcal{D}_{\leq}$ iff the planning problem ϕ is satisfied in $\mathcal{D}$.

Clearly, ϕ' is itself a planning problem for domain $\mathcal{D}_{\leq}$. Consequently, any planning problem of $\mathcal{FC}_{PLN}$ and, hence, also the halting problem for two-counter machines can be represented as a planning problem in $\mathcal{FC}_{PL_{\leq}}$.

5.7 Summary of Results

The Table 5.1 summarises the results we proved in this chapter. Every position in the table determines a decision problem. The column determines a Fluent Calculus fragment and the line determines the class of queries considered. The character "+" represents a positive result, i.e. decidability of the decision problem. The character "−" represents general undecidability of the problem class. The results we have explicitly proven in this chapter are marked by $\ominus$ and $\oplus$, respectively. The frame surrounding some lines and columns shows how the decidability/undecidability of other decision problems depends on the proven result. E.g., as a consequence of the undecidability of the planning problem in $\mathcal{FC}_L$ the extended planning problem in $\mathcal{FC}_{LN}$ is also undecidable. However, due to the two dimensions of the table, some dependencies could not be represented, e.g. undecidability of the extended planning problem in $\mathcal{FC}_{LN}$ is also a consequence of undecidability of planning in $\mathcal{FC}_{PL_{\leq}}$.

As mentioned before the undecidability of the planning problem for $\mathcal{FC}_{PLN}$ domains (marked by "$\ominus$*") has been shown in [67], which was developed independently of our work. The decision problems marked by "*" in the table are

	$\mathcal{FC}_{LN}$	$\mathcal{FC}_{L}$	$\mathcal{FC}_{PLN}$	$\mathcal{FC}_{PL\leq}$	$\mathcal{FC}_{PL}$	$\mathcal{FC}_{P}$
Entailment (μ)	$-^*$	$-$	$-^*$	$-$	$-$	$\oplus$
Entailment (CTL_U)	$-^*$	$-$	$-^*$	$-$	$\ominus$	$+$
Extended Planning	$-^*$	$-$	$-^*$	$-$	$\oplus$	$+$
Planning	$-^*$	$\ominus$	$\ominus^*$	$\ominus$	$\oplus$	$+^{**}$
Conjunctive Planning	$-^*$	$-$	$-^*$	$-$	$+$	$+$

Fragment	Syntactic Properties
$\mathcal{FC}_{LN}$	linear, positive and negative preconditions
$\mathcal{FC}_{L}$	linear, positive preconditions
$\mathcal{FC}_{PLN}$	propositional, linear, positive and negative preconditions
$\mathcal{FC}_{PL\leq}$	propositional, linear, positive preconditions, specificity
$\mathcal{FC}_{PL}$	propositional, linear, positive preconditions
$\mathcal{FC}_{P}$	propositional, positive (and negative) preconditions

Table 5.1: The table on top contains a summary of our results and some of their consequences. The table below recalls the syntactic properties of the fragments considered here.

also consequences of this result. Also in [67], it has been shown that monadic second-order queries are decidable for a so-called *restricted Fluent Calculus*. This fragment does not allow the expression of the kind of state equations as used in the Definition 2.4.1 of state update axioms. In state formulas of [67] their expression requires two free variables, but only one is allowed in the restricted Fluent Calculus. Hence, the decidability results concerning the restricted Fluent Calculus can not be compared with our results. Finally, it has been shown in [67], that first-order queries are decidable for propositional Fluent Calculus fragments equipped with the Axiom (NM). For the fragment $\mathcal{FC}_P$ this result is also a consequence of the decidability of the strictly more expressive μ-calculus queries.

The planning problem for $\mathcal{FC}_P$ (marked by $+^{**}$) has also been addressed in [69]. There it has been shown that the planning problem where the initial state is completely specified is decidable. To provide efficient decision procedures a representation using binary decision diagrams has been proposed.

Part III

Automatic Reasoning

Chapter 6

Fluent Calculus Domains as Logic Programs

6.1 Automatic Conjunctive Planning

From an automatic reasoning point of view the most interesting results of Table 5.1 are those concerning the decidability of (extended/conjunctive) planning problems for the fragment $\mathcal{FC}_{PL}$. This is in particular the case if we keep in mind that $\mathcal{FC}_{PL}$ allows the description of a much richer class of systems than $\mathcal{FC}_P$ or the situation calculus fragment of [145], which enable the representation of finite state machines, only. However, as the proofs of decidability of reachability in [103] and semi-linearity in [85, 61] for Petri nets, respectively, show, the decision algorithms are complicated and computationally very expensive (in fact, at the time of writing the complexity boundaries are not even sufficiently explored). In contrast to this, the existence of a comparatively simple decision procedure for the conjunctive planning problem is guaranteed by its correspondence (see Section 6.3) to the coverability problem for Petri nets, which can be solved by Algorithm 4.2.1 (see Section 4.2). Hence, in this last part of this work we focus on developing an automatic reasoning procedure to decide the conjunctive planning problem for $\mathcal{FC}_{PL}$ domains.

To enable general automatic reasoning about Fluent Calculus domain descriptions, it has been proposed in [68] to represent Fluent Calculus domains as definite E-programs (i.e. definite logic programs with an equational theory, see Definition 1.3.1 in Section 1.3). Definite E-programs do not allow the use of negation in the definition of clauses. Consequently, only domain descriptions of $\mathcal{FC}_L$ and its propositional fragment $\mathcal{FC}_{PL}$ can be represented in this way. However, as we demonstrate in this chapter, even for $\mathcal{FC}_{PL}$ the application of a standard calculus for definite E-programs like SLDE-resolution does not lead to satisfying results, in particular it does not provide a decision procedure for the CPP. Because of this, the method we will develop in Chapter 7 is based on program transformation techniques for definite logic programs. Hence, we may apply it also to some $\mathcal{FC}_L$ domains. Further generalisation would require significant extensions of the underlying program transformation techniques.

6.2 $\mathcal{FC}_L$ domains as definite E-programs

According to Section 2.3 we consider only Herbrand-$\mathtt{E}_{\mathcal{FC}}$-interpretations (see Section 2.3) for Fluent Calculus domain descriptions. Hence, in contrast to Herbrand-E-interpretations, the function *state* can be interpreted freely, rather than by the mapping to the term set $\{state(s) \mid s \in T_{Sit,\Sigma}(\emptyset)\}$ according to Definition 1.1.10. Since for definite E-programs only Herbrand-E-interpretations are considered the function *state* has to be represented in a different way. It is well known that any function may be equivalently represented as a relation between the domain and the co-domain. As Herbrand-E-interpretations allow the free interpretation of predicates we may represent the function *state* in definite E-programs by some fresh predicate $AssSt : Sit \times St$.

We may now rewrite the domain independent axioms $\mathtt{O}_{Sit}$ (see Section 2.2) in a logic programming style, where we denote the representation of each Fluent Calculus predicate by a subscript 1 (*Less* represents the predicate <).

$$\begin{array}{l} Less_1(s_1, s_2) \leftarrow s_2 = do(x, s_1), Poss_1(x, s_1). \\ Less_1(s_1, s_2) \leftarrow s_2 = do(x, s), Poss_1(x, s), Less_1(s_1, s). \end{array} \tag{6.1}$$

Due to Proposition 1.3.1 shown in [72], every definite E-program has a smallest Herbrand-E-model. Note that in the smallest Herbrand-E-model the second axiom of $\mathtt{O}_{Sit}$ and the "$\rightarrow$"-part of the first axiom of $\mathtt{O}_{Sit}$ are both valid.

Furthermore, if the selection rule of SLDE-resolution always selects the subgoal that is leftmost then the second clause defining $Less_1$ can be viewed as a *backward interpreter* in the sense that in every SLDE-refutation of some query $Less_1(s_1, s_2)$ we have an ordered sequence of situations leading from s_2 to s_1 represented by the subgoals, as depicted in Figure 6.1 where $a_1,\ldots,a_n$ are some action terms and $s',\ldots,s^{(n)}$ some situation terms. The goal situation is decreased until it is a successor of the initial situation.

Instead of representing $\mathtt{O}_{Sit}$ as a backward interpreter we may equally choose a *forward interpreter*, or any combination of the two. In the following chapter we will be particularly interested in applying the following forward interpreter:

$$\begin{array}{l} Less'_1(s_1, s_2) \leftarrow s_2 = do(x, s_1), Poss_1(x, s_1). \\ Less'_1(s_1, s_2) \leftarrow s = do(x, s_1), Poss_1(x, s_1), Less'_1(s, s_2). \end{array} \tag{6.2}$$

Then SLDE-refutations of some query $Less'_1(s_1, s_2)$ contain the ordered sequence of subgoals depicted in Figure 6.2.

$$\begin{array}{l}
\leftarrow \mathit{Less}_1(s_1, s_2) \\
\vdots \\
\qquad \{s_2 / do(a_1, s')\} \\
\leftarrow \mathit{Less}_1(s_1, s') \\
\vdots \\
\qquad \{s / do(a_2, s'')\} \\
\leftarrow \mathit{Less}_1(s_1, s'') \\
\vdots \\
\vdots \\
\qquad \{s^{(n-1)} / do(a_{n-1}, s^{(n)})\} \\
\leftarrow \mathit{Less}_1(s_1, s^{(n)}) \\
\vdots \\
\qquad \{s^{(n)} / do(a_n, s_1)\} \\
\vdots \\
\square
\end{array}$$

Figure 6.1: The sequence of subgoals encountered for query $\mathit{Less}_1(s_1, s_2)$ using a backward interpreter.

In order to represent the domain dependent axioms we will redefine the shortcut *Holds* of Equation 2.2 to use the predicate *AssSt* instead of function *state*. We call the new shortcut Holds_1:

$$\mathit{Holds}_1(z, s) \stackrel{\text{def}}{=} \mathit{AssSt}_1(s, z_1), z \circ v =_{St} z_1. \tag{6.3}$$

Every action precondition axiom of some $\mathcal{FC}_L$ domain as defined in Definition 2.5.1 is represented in the following straightforward way:

$$\begin{aligned}\mathit{Poss}_1(x, s) \leftarrow x = a(\bar{Y}), &\mathit{Holds}_1(St_a^-(\bar{Y}), s),\\ &\mathit{Holds}_1(St_{a,1}^=, s), \ldots, \mathit{Holds}_1(St_{a,m_a}^=, s).\end{aligned} \tag{6.4}$$

The representation of the state update axioms, which define the function *state*, causes several difficulties. On one hand, we have to ensure that *AssSt* is interpreted as a function. I.e. if there are states z_1, z_2 such that $\mathit{AssSt}(s, z_1)$ and $\mathit{AssSt}(s, z_2)$ are in the model, then $z_1 =_{St} z_2$. On the other hand, in the smallest Herbrand-E-model *AssSt* represents only a partial function, since it is only defined for those situations $do(x, s)$ where $\mathit{Poss}(x, s)$ is valid. The latter

$$\begin{array}{l}
\leftarrow Less'_1(s_1, s_2) \\
\vdots \\
\leftarrow Less'_1(do(a_n, s_1), s_2) \\
\vdots \\
\leftarrow Less'_1(do(a_{n-1}, do(a_n, s_1)), s_2) \\
\vdots \\
\vdots \\
\leftarrow Less'_1(do(a_2, \ldots do(a_n, s_1)\ldots), s_2) \\
\vdots \\
\square
\end{array}$$

Figure 6.2: The sequence of subgoals encountered for query $Less'_1(s_1, s_2)$ using a forward interpreter.

does not impose a serious restriction, since in the Definition 2.7.3 of the associated transition system, only situations $do(x, s)$ where $Poss(x, s)$ is valid are considered.

$$\begin{aligned} AssSt_1(s, z) \leftarrow s = do(a(\bar{Y}), s_1), Poss_1(a(\bar{Y}), s_1), AssSt_1(s_1, z_1), \\ z \circ St_a^- =_{St} z_1 \circ St_a^+. \end{aligned} \tag{6.5}$$

By these axioms the state associated with some situation is determined precisely by the state associated with the preceding situation. But to ensure that *AssSt* is interpreted as a function we have to specify the state associated with the initial situation:

$$AssSt_1(s, z) \leftarrow s = s0, z =_{St} g. \tag{6.6}$$

Where g should be a ground term of sort St. If g would instead contain variables, all instances of g are valid states associated with $s0$, and hence, *AssSt* can not be interpreted as a function. Consequently, in the above representation of $\mathcal{FC}_L$ domains it is not possible to solve planning problems where the initial state is not completely known.

Despite this, it is also possible to represent the function *state* in definite E-programs implicitly rather than explicitly by the predicate *AssSt*. Note that every predicate which describes a relation $\mathcal{R}$ on a set S of situations and sets $O_1, \ldots, O_n$ of other objects induces also a relation $\mathcal{R}'$ on S, the states associated with S, and $O_1, \ldots, O_n$ where $(s_1, o_1, \ldots, o_n) \in \mathcal{R}$ iff $(s_1, state(s_1), o_1, \ldots, o_n) \in \mathcal{R}'$. Consequently, we may extend every predicate such that it contains for every parameter which represents a situation also a parameter representing the associated state. However, we have to ensure that this state indeed corresponds to the one associated by the function *state*.

Since state update axioms define a particular relation between states that are associated with subsequent situations, we may also view them as defining a relation *Succc* : *Sit* × *St* × *Sit* × *St*, where $(s_1, z_1, s_2, z_2) \in \mathit{Succ}$ if $z_1 = \mathit{state}(s_1)$, $z_2 = \mathit{state}(s_2)$, $s_2 = do(a, s_1)$ and $\mathit{Poss}(x, s, z_1)$ for some action term x. Using this relation we represent $\mathtt{O}_{Sit}$ as the following forward interpreter (the subscript 2 denotes this new representation):

$$\begin{aligned} &\mathit{Less}_2(s_1, z_1, s_2, z_2) \leftarrow \mathit{Succ}_2(s_1, z_1, s_2, z_2). \\ &\mathit{Less}_2(s_1, z_1, s_2, z_2) \leftarrow \mathit{Succ}_2(s_1, z_1, s, z), \mathit{Less}_2(s, z, s_2, z_2). \end{aligned} \tag{6.7}$$

We also define *Holds*$_2$ which redefines the shortcut *Holds* to contain an additional parameter to represent the state associated with situation s.

$$\mathit{Holds}_2(z_1, s, z) \overset{\text{def}}{=} (z_1 \circ v =_{St} z). \tag{6.8}$$

Note that the parameter s is actually not used in the definition of *Holds*$_2$.

All action precondition axioms which define the relation *Poss* are redefined to extend *Poss* with the additional state parameter. The new predicate is called *Poss*$_2$:

$$\begin{aligned} \mathit{Poss}_2(x, s, z) \leftarrow\ & x = a(\bar{Y}), \mathit{Holds}_2(St_a^-, s, z), \\ & \mathit{Holds}_2(St_{a,1}^=, s, z), \dots, \mathit{Holds}_2(St_{a,m_a}^=, s, z). \end{aligned} \tag{6.9}$$

Again, the parameter s is actually not required to define the relation *Poss*$_2$. This is due to the fact that in $\mathcal{FC}_L$ domains the executability of an action may not depend on the history of action executions (Markov property).

Finally, the state update axioms have to be modified to define the relation *Succ* rather than the function *state*:

$$\begin{aligned} \mathit{Succ}_2(s_1, z_1, s_2, z_2) \leftarrow\ & s = do(a(\bar{Y}), s_1), \mathit{Poss}_2(a(\bar{Y}), s_1, z_1), \\ & z_2 \circ St_a^- =_{St} z_1 \circ St_a^+. \end{aligned} \tag{6.10}$$

Due to the Markov property, every action sequence leading from some situation with associated state z_1 to some other situation with associated state z_2 is also executable in any other situation with associated state z_1 and also leads to some situation with state z_2. Consequently, it is sufficient to define the predicate *Less* in terms of action sequences and the states associated with the initial and the goal situation, respectively. This allows us to simplify the above definite E-program further. The interpretation of *Succ* in a Herbrand-E-model (of the above representation of a $\mathcal{FC}_L$ domain) does not depend on the structure of the situation s_1 but on the last executed action in situation s_2. Hence, we may simplify its representation by omitting the situation parameters and add a new parameter which represents the last executed action. The predicate $Less_3$ redefines $Less_2$ accordingly:

$$\begin{array}{l} Less_3(s_1, z_1, s_2, z_2) \leftarrow s_2 = do(x, s_1), Succ_3(z_1, x, z_2). \\ Less_3(s_1, z_1, s_2, z_2) \leftarrow s = do(x, s_1), Succ_3(z_1, x, z), Less_3(s, z, s_2, z_2). \end{array} \tag{6.11}$$

Suppose we are particularly interested in queries $Less_3(s_1, z_1, s_2, z_2)$ where s_1 and s_2 are uninstantiated, i.e. we only want to know whether there *exist* situations s_1 and s_2 such that there is an action sequence leading from s_1 to s_2 and the states associated with s_1 and s_2 are z_1 and z_2, respectively. Then we may also define a binary relation *Reachable* as

$$Reachable(z_1, z_2) \leftarrow Less_3(s_1, z_1, s_2, z_2).$$

which may equivalently be defined without referring to *Less*:

$$\begin{array}{l} Reachable(z_1, z_2) \leftarrow Succ_3(z_1, x, z_2). \\ Reachable(z_1, z_2) \leftarrow Succ_3(z_1, x, z), Reachable(z, z_2). \end{array} \tag{6.12}$$

We will use the latter definition of *Reachable* to simplify the presentation of our approach for solving the CPP.

As mentioned before the definition of the shortcut *Holds* does not depend on the situation parameter, hence, we may omit it:

$$Holds_3(z_1, z) \stackrel{\text{def}}{=} z_1 \circ z_2 =_{St} z. \tag{6.13}$$

Equally, we may simply omit the situation parameter in all representations of precondition axioms *Poss*:

$$\begin{array}{l} Poss_3(x, z) \leftarrow x = a(\bar{Y}), Holds_3(St_a^-, z), \\ \qquad\qquad Holds_3(St_{a,1}^=, z), \ldots, Holds_3(St_{a,m_a}^=, z). \end{array} \tag{6.14}$$

The successor state axioms are represented, as argued above, without situation parameters and an additional action parameter x.

$$\begin{array}{l} Succ_3(z_1, x, z_2) \leftarrow x = a(\bar{Y}), Poss_3(x, z_1), \\ \qquad\qquad z_2 \circ St_a^- =_{St} z_1 \circ St_a^+. \end{array} \tag{6.15}$$

In the following chapter we will focus on two particular classes of such representations of $\mathcal{FC}_L$ domains. The first class is the one which corresponds to $\mathcal{FC}_{PL}$ domains, i.e. those $\mathcal{FC}_L$ domains where actions are constants (the sequences $\bar{Y}$ of variables in all precondition and successor state axioms are empty). In this case we may use simpler clauses of the form

$$Poss_3(x, z) \leftarrow x = a, Holds_3(St_a^- \circ St_a^=, z). \tag{6.16}$$

based on the Formula 2.4 to define $Poss_3(\_,\_)$. From Section 5.3 we know that the planning problem and in particular the conjunctive planning problem are decidable for such domains. The second class is the one which corresponds to simple Fluent Calculus domains, i.e. those $\mathcal{FC}_L$ domains where for every action a all terms $St^{=}_{a,i}$ that appear in the precondition axiom[1] of a are equal to 1°. However, from Section 5.5 we know that both, planning problem and conjunctive planning problem, are undecidable.

Example 6.2.1 (Airport domain as definite E-program) *Consider the $\mathcal{FC}_{PL}$ domain description of Example 2.5.4. We may represent this domain as the following definite* E*-program (we assume additionally the domain independent Clauses 6.11 and the shortcut defined in Formula 6.13):*

$$
\begin{array}{l}
Poss_3(queue\_b, z) \leftarrow \top.\\
Poss_3(queue\_s, z) \leftarrow \top.\\
Poss_3(land\_b, z) \leftarrow Holds(bay^2 \circ plane\_q\_b \circ runway, z).\\
Poss_3(land\_s, z) \leftarrow Holds(bay \circ plane\_q\_s \circ runway, z).\\
Poss_3(take\_off\_b, z) \leftarrow Holds(plane\_l\_b \circ runway, z).\\
Poss_3(take\_off\_s, z) \leftarrow Holds(plane\_l\_s \circ runway, z).
\end{array}
$$

$$
\begin{array}{ll}
Succ_3(z_1, x, z_2) \leftarrow & x = queue\_b, Poss_3(x, z_1), z_2 =_{St} z_1 \circ plane\_q\_b.\\
Succ_3(z_1, x, z_2) \leftarrow & x = queue\_s, Poss_3(x, z_1), z_2 =_{St} z_1 \circ plane\_q\_s.\\
Succ_3(z_1, x, z_2) \leftarrow & x = land\_b, Poss_3(x, z_1),\\
& z_2 \circ bay^2 \circ plane\_q\_b =_{St} z_1 \circ plane\_l\_b \circ passenger^5.\\
Succ_3(z_1, x, z_2) \leftarrow & x = land\_s, Poss_3(x, z_1),\\
& z_2 \circ bay \circ plane\_q\_s =_{St} z_1 \circ plane\_l\_s \circ passenger^3.\\
Succ_3(z_1, x, z_2) \leftarrow & x = take\_off\_b, Poss_3(x, z_1),\\
& z_2 \circ plane\_l\_b =_{St} state(s) \circ bay^2.\\
Succ_3(z_1, x, z_2) \leftarrow & x = take\_off\_s, Poss_3(x, z_1),\\
& z_2 \circ plane\_l\_s =_{St} state(s) \circ \circ bay.
\end{array}
$$

□

Example 6.2.2 (Abstract $\mathcal{FC}_{PL}$ domain as definite E-program) *The following definite* E*-program (together with the usual domain independent clauses) corresponds to the $\mathcal{FC}_{PL}$ domain $\mathcal{D}_p$ of Example 2.5.5 wrt signature Σ_p.*

$$
\begin{array}{l}
Poss_3(x, z) \leftarrow x = a1, Holds_3(f1, z).\\
Poss_3(x, z) \leftarrow x = a2, Holds_3(f1, z).\\
Poss_3(x, z) \leftarrow x = a3, Holds_3(f2, z).\\
Poss_3(x, z) \leftarrow x = a4, Holds_3(f3, z).\\
Poss_3(x, z) \leftarrow x = a5, Holds_3(f4, z).\\
Poss_3(x, z) \leftarrow x = a6, Holds_3(f5, z).
\end{array}
$$

[1]Note that due to Equation 2.3 $\mathcal{FC}_{PL}$ domains may also be represented in this way. The corresponding precondition and state update axioms are defined by $St'^{-}_{a} = St^{-}_{a} \circ St^{=}_{a}$ and $St'^{+}_{a} = St^{+}_{a} \circ St^{=}_{a}$.

$$\begin{aligned}
&Succ_3(z_1, x, z_2) \leftarrow x = a1, Poss_3(x, z_1), z_2 \circ f1 =_{St} z_1 \circ f2.\\
&Succ_3(z_1, x, z_2) \leftarrow x = a2, Poss_3(x, z_1), z_2 \circ f1 =_{St} z_1 \circ f4.\\
&Succ_3(z_1, x, z_2) \leftarrow x = a3, Poss_3(x, z_1), z_2 \circ f2 =_{St} z_1 \circ f3 \circ f3.\\
&Succ_3(z_1, x, z_2) \leftarrow x = a4, Poss_3(x, z_1), z_2 \circ f3 =_{St} z_1 \circ f2.\\
&Succ_3(z_1, x, z_2) \leftarrow x = a5, Poss_3(x, z_1), z_2 \circ f4 =_{St} z_1 \circ f5 \circ f5.\\
&Succ_3(z_1, x, z_2) \leftarrow x = a6, Poss_3(x, z_1), z_2 \circ f5 =_{St} z_1 \circ f4.
\end{aligned}$$

□

Example 6.2.3 (Brick domain as definite E-program) *Consider the $\mathcal{FC}_L$ domain description of Example 2.5.1. We may represent this domain as the following definite* E*-program:*

$$Poss_3(x, z) \leftarrow x = mv\_brick(x_1, y_1, x_2, y_2), Holds(brick\_at(x_1, y_1), z).$$

$$\begin{aligned}
Succ_3(z_1, x, z_2) \leftarrow\ & x = mv\_brick(x_1, y_1, x_2, y_2), Poss_3(x, z_1),\\
& z_2 \circ brick\_at(x_1, y_1) =_{St} z_1 \circ brick\_at(x_2, y_2).
\end{aligned}$$

□

Example 6.2.4 (Abstract $\mathcal{FC}_L$ domain as definite E-program) *The following definite* E*-program (together with the usual domain independent clauses) corresponds to the $\mathcal{FC}_L$ domain $\mathcal{D}_l$ of Example 2.5.2 wrt signature Σ_l.*

$$\begin{aligned}
&Poss_3(x, z) \leftarrow x = a1, Holds_3(f1, z).\\
&Poss_3(x, z) \leftarrow x = a3(y), Holds_3(f2, z).\\
&Poss_3(x, z) \leftarrow x = a4(y), Holds_3(f3(y), z).
\end{aligned}$$

$$\begin{aligned}
Succ_3(z_1, x, z_2) \leftarrow\ & x = a1, Poss_3(x, z_1), z_2 \circ f1 =_{St} z_1 \circ f2(0).\\
Succ_3(z_1, x, z_2) \leftarrow\ & x = a3(y), Poss_3(x, z_1),\\
& z_2 \circ f2(y) =_{St} z_1 \circ f3(foo(y)) \circ f3(foo(foo(y))).\\
Succ_3(z_1, x, z_2) \leftarrow\ & x = a4(y), Poss_3(x, z_1), z_2 \circ f3(y) =_{St} z_1 \circ f2(y).
\end{aligned}$$

□

6.3 Conjunctive Planning by SLDE-resolution

In [68] it has been shown that SLDE-resolution is sound and complete for an instance of CPP in simple $\mathcal{FC}$ domains, i.e. every solution of such a CPP is entailed by SLDE-resolution. However, even for $\mathcal{FC}_{PL}$ domains represented as definite E-programs the SLDE-tree may contain infinite derivations and consequently, the search for a plan may not terminate. To demonstrate this we represent the Formula 2.10 which is used to state planning problems as two queries. If there is a correct answer to one of them, the CPP can be solved and the corresponding plan can be deduced from the corresponding SLDE-refutation. Here, we define the particular class of CPP's considered in [68] by two queries. Let g_1 and g_2 denote state terms which describe the initial and the goal state, respectively.

The first query has a correct answer substitution iff the initial state is already a goal state (in this case the plan is empty):

$$\leftarrow z_1 =_{St} g_1, z_2 =_{St} g_2 \circ z, z_1 =_{St} z_2. \qquad (6.17)$$

The second query has a correct answer substitution iff there is a situation s_1 and a situation s_2 such that $s_1 < s_2$ and g_1 is the state associated with s_1 and $Holds_3(g_2, z_2)$ where z_2 is the state associated with s_2.

$$\leftarrow z_1 =_{St} g_1, z_2 =_{St} g_2 \circ z, Reachable(z_1, z_2). \qquad (6.18)$$

In $\mathcal{FC}_{PL}$ domains g_1 and g_2 should be both ground. There it is also sufficient to consider the single ground term g_2 as a description of the goal state, since by Equation 2.3 any conjunction of *Holds*-statements, which may characterise the goal state, can be transformed into a single one ($Holds_3(g_2, z_2) \Leftrightarrow z_2 =_{St} g_2 \circ z$ in the queries for CPP). On the other hand, if g_1 is considered to contain some variable z' such that $g_1 =_{St} g_1' \circ z'$, then the CPP would be trivially solvable according to the first query: the resulting equation $g_1' \circ z' =_{St} g_2 \circ z$ has always a simple correct answer substitution $\{z'/g_2, z/g_1'\}$. In fact, any conjunctive planning problem wrt $\mathcal{FC}_{PL}$ domains using the atomic propositions L_Σ can be reduced to a finite number of Queries 6.17 and 6.18: According to Section 5.3 it is sufficient to consider a finite number of (conjunctive) planning problems where each initial state description is of the form $Holds(g^p) \wedge \overline{Holds(g^n)}$ for some ground state terms g^p, g^n. Additionally, from the Proof 5.3.1 we know that an $\mathcal{FC}_{PL}$ domain can be transformed such that only a finite number of reachability problems (where the initial marking is completely specified) needs to be checked[2].

The following example demonstrates that there are indeed infinite derivations possible if conjunctive planning problems have to be solved.

Example 6.3.1 (Example 6.2.2 continued) *Let $g_1 = f2$ and $g_2 = f4$. If we repeatedly apply the actions a_3 and a_4 in alternation, we obtain an infinite*

[2]Recall: For every fluent f not bounded by $\overline{Holds(g^n)}$ we introduce a new action producing f and introduce a new limit for f by modifying g^n. Then we check all initial markings defined by the lower bounds and the new upper bounds on the fluents for reachability.

derivation for the second CPP query (we use the leftmost selection rule):

$$\begin{array}{l}
\leftarrow \underline{Reachable}(f2, f4 \circ z) \\
\leftarrow \underline{Succ_3}(f2, x, z), Reachable(z, f4 \circ z') \\
\qquad \{x/a3, z/f_3 \circ f_3\} \\
\leftarrow \underline{Reachable}(f3 \circ f3, f4 \circ z') \\
\leftarrow \underline{Succ_3}(f3 \circ f3, x, z), Less_3(z, f4 \circ z') \\
\qquad \{x/a4, z/f2\} \\
\leftarrow \underline{Reachable}(f3 \circ f2, f4 \circ z') \\
\leftarrow \underline{Succ_3}(f3 \circ f2, x, z), Reachable(z, f4 \circ z') \\
\qquad \{x/a3, z/f3 \circ f3 \circ f3\} \\
\vdots
\end{array}$$

□

Indeed, for but the most trivial examples, SLDE-resolution will loop if no plan exists, and will (due to depth-first exploration) often fail to find a plan if one exists. Note that the problem is not specific to SLDE-resolution. An alternative way of implementing $\mathcal{FC}$ domains would be to represent the equational theory as a *rewrite system* [4] and then use *narrowing*. But, for the same reasons as SLDE-resolution, narrowing will not always terminate. On one hand, infinite derivations can be caused by applying action sequences which do not alter the state of the system. In principle, such loops could be automatically detected and avoided, for instance by *tabling methods* [133]. But, on the other hand, the above example shows that derivations might be infinite even when they do not contain this kind of loops. Infinite derivations as in Example 6.3.1 may occur due to the potentially infinite state space in $\mathcal{FC}_{PL}$: the number of terms of sort St that are not equivalent wrt the equational theory `AC1` is not bounded. If in the above example the action sequence $a3, a4$ is executed n-times in a situation with associated state $f2$, the state associated with the final situations is equivalent to $f2 \circ f3^n$.

To enable the solution of the CPP or similar problems despite of potentially infinite SLDE-derivations we propose in the next chapter to use certain program transformation techniques.

Chapter 7

Conjunctive Planning by Partial Deduction

In recent years there has been considerable interest in model checking of systems with infinite state spaces, also called *infinite model checking* (e.g., [1, 111, 35, 32, 157, 26]). Since the conjunctive planning problem for $\mathcal{FC}_{PL}$ domain is a particular model checking problem of a certain class of infinite state systems, general results about infinite model checking apply also to CPP for $\mathcal{FC}_{PL}$.

One of the key issues of model checking of infinite systems is *abstraction* [21]. Abstraction allows to approximate an infinite system by a finite one, and if proper care is taken the results obtained for the finite abstraction will be valid for the infinite system. However, an important question when attempting to perform infinite model checking in practice is: How can one *automatically* obtain an abstraction which is finite, but still as precise as required? A potential solution to this problem is to apply existing techniques for the *automatic* control of *(logic) program specialisation* [91] and many others. The aim of program specialisation is to improve the efficiency of a program by pre-evaluating it for particular parameters. In the context of logic programming program specialisation is also called *partial deduction.* Similar to model checking of infinite systems, in program specialisation in general and partial deduction in particular, one faces the following (quite extensively studied) problem: To be able to produce efficient specialised programs, *infinite* computation trees have to be abstracted in a *finite* but also as *precise* as possible way.

To apply partial deduction the system to be verified has to be modelled as a logic program by means of an interpreter [52, 96]. Thereby, the interpreter describes how the states of the system change by executing transitions. By applying partial deduction to the interpreter we expect a finite abstraction of the possibly infinite state space of the system to be generated. This abstraction may then be used to verify system properties of interest. This approach proved already to be quite powerful as it was possible to obtain decision procedures for the coverability problem, if "typical" existing specialisation algorithms are applied to logic programs that encode Petri nets [94]. It is even possible to precisely mimic well known Petri net algorithms (by Karp–Miller [76] and by Finkel [39]) when the program specialisation techniques are slightly weakened. The results of [94] refer to *forward* algorithms only, i.e. algorithms which construct, beginning from some initial state, an abstract representation of the whole reachability tree of a Petri net. However, for some classes of systems such exhaustive algorithms are not necessary or even not precise enough to decide coverability [1, 40, 41]. In such cases partial deduction may often be successfully applied as well [93], thereby mimicking well known *backward algorithms* [40].

According to Section 4.3, we may try to apply algorithms from the Petri net theory to $\mathcal{FC}_{PL}$ domains in order to tackle the conjunctive planning problem. We may also apply the partial deduction methods of [93, 94] to the logic programming representation[1] of the Petri net that correspond to the $\mathcal{FC}_{PL}$ domain in question. The disadvantage of these approaches is that they hardly scale up to other $\mathcal{FC}$ domains. In contrast to this, the partial deduction approach in general applies to any system that can be represented as a definite logic program. Applying partial deduction directly on logic programming representations of $\mathcal{FC}$ domains is thus more promising, as it is not restricted to $\mathcal{FC}_{PL}$ domains, but works with any $\mathcal{FC}$ domain that can be represented as a definite logic program (although it might no longer provide a decision procedure). Thus, the idea of this chapter is to apply partial deduction to fluent calculus domains in order to decide the conjunctive planning problem for $\mathcal{FC}_{PL}$ specifications and to provide a useful procedure for more general $\mathcal{FC}$ domains. To this end, there are several problems that need to be solved for this approach to work:

- $\mathcal{FC}$ domains rely on `AC1`-unification, but unification under an equational theory is not directly supported by partial deduction as used in [94] and one would have to apply partial deduction to a meta-interpreter implementing `AC1`-unification. Although this is theoretically feasible, this is still problematic in practice for efficiency and precision reasons (see, e.g., [92]). A more promising approach is to extend partial deduction so that it can handle an equational theory.

- Another problem is connected to an inherent limitation of "classical" partial deduction, which relies on a rather crude domain for expressing calls: in essence a term represents all its instances. This was sufficient for handling Petri nets in [94] (where a term such as `[0,s(X)]` represents all Petri net markings with no tokens in place 1 and at least 1 token in place 2), but it is not sufficient to handle $\mathcal{FC}$ domains whose state expressions are more involved. To solve this we propose to use so called ***abstract partial deduction*** [90] with an abstract domain based upon regular types [158], and extend it to cope with equational theories.

[1]A program which contains another program (e.g., a Petri net) as data is also called *Meta-program.* See for example [65] for applications of Meta-programming in artificial intelligence using logic programs.

Although in this work we are mainly interested in applying partial deduction to the $\mathcal{FC}$ domains based upon `AC1`, we present the generalised partial deduction independently of the particular equational theory (as long as it is finitary). However, the use of this general method in practice relies on an efficient unification procedure. If such a procedure cannot be provided and/or one wishes to specialise the underlying equational theory, other approaches, e.g., based on narrowing [60, 3], should be considered. The reason we extend classical partial deduction for SLDE–resolution rather than building on top of [3], is that we actually do not wish to modify the underlying equational theory. As we will see later in this chapter, this leads to a simpler theory with simpler correctness results, and also results in a tighter link with classical partial deduction used in [94]. This also means that it is more straightforward to integrate abstract domains as described in [90] (no abstract specialisation exists as of yet for narrowing-based approaches).

In the remainder of the chapter, we thus develop a partial deduction method which considers both equational theories and regular type information. The method will then enable us to solve conjunctive planning problems in the simple Fluent Calculus. In particular, we show that our method is actually complete for conjunctive planning problems in $\mathcal{FC}_{PL}$. We believe that our approach can also be used for more complex systems, without changing much of the algorithm, e.g., in cases where completeness can not be guaranteed due to general undecidability.

Note that it has been proposed to use partial deduction already in [24] for the failure dedection in theorem proving, and in [25] specifically for the detection of unsolvable conjunctive planning problems and to solve postdiction problems in the Fluent Calculus. Similarly to our approach, the method utilises the idea of regular approximation. However, although [25] can only deal with $\mathcal{FC}_{PL}$ domains, it is not a decision procedure for conjunctive planning problems. In contrast to that, our approach provides a decision procedure and scales up to non-propositional domains (the simple $\mathcal{FC}$).

7.1 Partial Deduction for definite E-Programs

The general idea of partial deduction of ordinary logic programs [101] is to construct, given a query $\leftarrow Q'$ of interest, a finite number of finite but possibly incomplete SLD-trees which "cover" the possibly infinite SLD-tree for $D \cup \{\leftarrow Q'\}$ (and thus also all SLD-trees for all instances of $\leftarrow Q'$). The derivation steps in these SLD-trees are the computations which have been pre-evaluated and the clauses of the specialised program are then extracted by constructing one specialised clause (called *resultant*) per branch.

While the initial motivation for partial deduction was program specialisation, one can also use partial deduction as a *top-down flow analysis* of the program under consideration. Indeed, partial deduction will unfold the initial query of interest until it spots a dangerous growth, at which point it will generalise the offending calls and restart the unfolding from the thus obtained more general call. Provided a suitably refined control technique is used (e.g., [95, 23]), one can guarantee termination as well as a precise flow analysis. As was shown in [93] such a partial deduction approach is powerful enough to provide a decision procedure for coverability problems for (reset) Petri nets and bears resemblance to the Karp-Miller procedure [76]. In the context of the CPP, the initial query of interest would be Query 6.18 (we do not need to consider Query 6.17 as it can not result in an infinite derivation)

$$\leftarrow \mathit{Reachable}(g_1, g_2 \circ z).$$

where g_1 and g_2 are the initial and the goal state respectively and one would hope to obtain as a result a flow analysis from which it is clear whether the CPP has a solution.

The abstraction in "classical" partial deduction techniques relies on the relation "instance-of" and the associated most specific generalisation.

Definition 7.1.1 *Let $\Sigma = (SORT, \preceq, FUN, REL)$ be a signature, X a variable declaration wrt Σ. Furthermore, let $P \subseteq A_\Sigma(X)$. We call some $p \in A_\Sigma(X)$ a* most specific generalisation *(msg) of P if*

1. *every $q \in P$ is an instance of P, and*
2. *if for some $p' \in A_\Sigma(X)$ every $q \in P$ is an instance of p', then p is also instance of p'.*

For any two atoms of the same predicate there exists always a unique (up to variants) most specific generalisation. For example, we have that the *msg* of $\{\mathit{Less}(\mathit{zero}, \mathit{succ}(\mathit{zero})), \mathit{Less}(\mathit{zero}, \mathit{succ}(\mathit{succ}(\mathit{zero})))\}$ is $\mathit{Less}(\mathit{zero}, \mathit{succ}(x))$. The *msg* can be easily computed [120, 87]; this process is also referred to as *anti-unification* or *least general generalisation.* It has the important property, that for every atom A, there are no infinite chains of strictly more general atoms [71].

Unfortunately, as it has been demonstrated in [93] among others, "classical" partial deduction techniques may not be precise enough if state descriptions are complex. Similar problems occur if states are represented using non-empty equational theories, since abstractions just based on the "instance-of" relation and the associated most specific generalisation may be too crude (c.f., also [90]).

Example 7.1.1 (Example 6.2.2 continued) *The msg of $\mathit{Reachable}(f2, z_1)$ and $\mathit{Reachable}(f3 \circ f2, z_2)$ is $\mathit{Reachable}(f2 \circ x, z')$. This is quite unsatisfactory, as x can represent* any *term, i.e., also terms containing other fluents such as $f4$. In the context of CPP this means that any action can potentially be executed from $f2 \circ x$, and we have thrown away too much information for the generalisation to be useful. For example, if our goal state is $f4$, we would not be able to prove that we cannot solve the CPP from the initial state $f2$.* □

In classical partial deduction there is no way of overcoming this problem, due to its inherent limitation that a call must represent all its instances (the same holds for narrowing-based partial evaluation [3]). Fortunately, this restriction has been lifted, e.g., in the abstract partial deduction framework of [90] and [45]. In essence, partial deduction and conjunctive partial deduction [23] are extended by working on *abstract conjunctions* on which *abstract unfolding* and *abstract resolution* operations are defined:

- An abstract conjunction is linked to the concrete domain of "ordinary" conjunctions via a concretisation function γ. In contrast to classical partial deduction, γ can be much more refined than the "instance-of" relation. For example, an abstract conjunction can be a couple (Q, τ) consisting of a concrete conjunction Q and some type[2] information τ, and $\gamma((Q, \tau))$ would be all the instances of Q which respect the type information τ. We could thus disallow f_4 to be an instance of x in Example 7.1.1 by defining the type of x to be the set of terms which can be constructed using the constant f_3 and the function $\circ$, only.
- An abstract unfolding operation maps an abstract conjunction A to a set of *concrete* resultants $H_i \leftarrow B_i$, which have to be totally correct for all possible calls in $\gamma(A)$.
- For each such resultant $H_i \leftarrow B_i$ the abstract resolution will produce an *abstract* conjunction Q_i approximating all the possible resolvent goals which can occur after resolving an element of $\gamma(A)$ with $H_i \leftarrow B_i$.

It is to this framework, suitably adapted to cope with SLDE-resolution, that we turn to remedy our problems. We will actually only consider abstract atoms consisting of a concrete atom together with some type information. The latter will be represented by *deterministic regular unary logic (RUL) programs* [158, 46] extended for equational theories. To extend *RUL* programs to an SLDE-setting, we want to take the standard definition of an *RUL* program and simply examine it in the context of our equational theory. Unfortunately, this is slightly problematic. Indeed, take a simple *RUL* program R that represents the type $\{f1 \circ f2\}$. Now, the interpretation of R in `AC1` will represent the type $\{f1 \circ f2, f2 \circ f1\}$, which is not tuple-distributive[3] (it does not contain $f1 \circ f1$ nor $f2 \circ f2$). This will have implications for decidability and efficiency (the main reason for using *RUL* programs) and we solve this by using just those *RUL* programs where the equational theory is already compiled in:

Definition 7.1.2 *A* canonical regular unary clause *is a clause of the form*

$$t_0(f(x_1, \ldots, x_n)) \leftarrow t_1(x_1) \wedge \cdots \wedge t_n(x_n)$$

[2]A type is simply a decidable set of terms closed under substitution.

[3]A set T of terms is tuple-distributive if whenever $f(a,b), f(c,d) \in T$, also $f(a,d), f(b,c) \in T$ holds.

where $n \geq 0$ and $x_1, \dots, x_n$ are distinct variables. A canonical regular unary logic (RUL) program *is a program R consisting of a finite set of regular unary clauses, in which no two different clause heads have a common instance.*

Let E *be an equational theory. We call R* E-compatible *if for all terms s, whenever $R \models t(s)$ then for all terms s' with $s =_{\mathrm{E}} s'$, also $R \models t(s')$.*

The set of ground terms r such that $R \models t(r)$ is denoted by $\tau_R(t)$. A ground term r is of type t in R iff $r \in \tau_R(t)$. Given a (possibly non-ground) conjunction T, we write $R \models \forall(T)$ iff for all ground instances T' of T, $R \cup \{\leftarrow T'\}$ has an SLD-refutation.

So, to solve Example 7.1.1 one could use the following AC1-compatible RUL program, representing all states using just the fluent $f3$, and give the variable x in Example 7.1.1 the type t_3:

$$
\begin{aligned}
&t_3(1^\circ).\\
&t_3(f3).\\
&t_3(x \circ y) \leftarrow t_3(x) \wedge t_3(y).
\end{aligned}
$$

Given two RUL-programs R_1, R_2, there exist efficient procedures for checking inclusion, computing the intersection and computing an upper bound using well known algorithms on corresponding automata [158]. Because of our definition, we can simply re-use the first two procedures to efficiently decide inclusion and compute the intersection of RUL programs (the intersection of E-compatible RUL programs will still be E-compatible). Given two RUL programs R_1, R_2 and two types t_1, t_2, we will denote by $(R_1, t_1) \sqcap (R_2, t_2)$ the couple (R_3, t_3) obtained by the latter procedure (i.e., we have $\tau_{R_3}(t_3) = \tau_{R_1}(t_1) \cap \tau_{R_2}(t_2)$). We will not make use of the upper bound and provide our own generalisation mechanism.

Definition 7.1.3 *Given some RUL program R, a* type conjunction (in R) *is simply a conjunction of the form $t_1(x_1) \wedge \dots \wedge t_n(x_n)$, where all the x_i are variables (not necessarily distinct) and all the t_i are defined in R. We also define the notation $types_T(x) = \{t_j \mid t_j(x) \in atoms(T)\}$.*

E.g., let $T = t(x) \wedge t'(z)$ be a type conjunction, where t and t' are defined by some RUL program R. Then $types_{t(x) \wedge t'(z)}(z) = \{t'\}$.

We now define the abstract domain used to instantiate the framework of [90]:

Definition 7.1.4 *We define the* $RULE$-domain $(\mathcal{AQ}, \gamma, \mathrm{E})$ *to consist of an equational theory* E, abstract conjunctions *of the form $\langle Q, T, R\rangle \in \mathcal{AQ}$ where Q is a concrete conjunction, R an E-compatible RUL-program, and T a type conjunction in R such that $T = t_1(x_1) \wedge \dots \wedge t_n(x_n)$, where* Vars$(Q) = \{x_1, \dots, x_n\}$.[4] *The* concretisation function γ *is defined by: $\gamma(\langle Q, T, R\rangle) = \{Q\theta \mid R \models \forall(T\theta)\}$.*

[4]When writing, e.g., *Vars*$(Q) = \{x_1, \dots, x_n\}$ we assume that all x_i are distinct.

We now define simplification and projection operations for type conjunctions. This will allow us to apply substitutions to abstract conjunctions as well as to define an (abstract) unfolding operation. As the above definition requires every variable to have exactly one type, the type of a variable z occurring in a substitution such as $\{x/z, y/z\}$ has to be determined by type intersection.

Definition 7.1.5 *Let R be some RUL program. The relation $\rightsquigarrow_R$ which maps type conjunctions to type conjunctions is defined as follows:*

- $t_1 \wedge t_2 \rightsquigarrow_R s_1 \wedge s_2$ *if* $t_1 \rightsquigarrow_R s_1$, $t_2 \rightsquigarrow_R s_2$, $s_1 \neq fail$, *and* $s_2 \neq fail$
- $t(x) \rightsquigarrow_R t(x)$ *if* x *is a variable*
- $t(c) \rightsquigarrow_R true$ *if* c *is a constant with* $c \in \tau_R(t)$
- $t(f(r_1, \dots, r_n)) \rightsquigarrow_R s_1 \wedge \dots \wedge s_n$ *if* $t(f(x_1, \dots, x_n)) \leftarrow t_1(x_1) \wedge \dots \wedge t_n(x_n) \in R$ *and* $t_i(r_i) \rightsquigarrow_R s_i$, *for* $i = 1, \dots, n$
- $t(r) \rightsquigarrow_R fail$ *otherwise.*

We define a projection *which projects a type conjunction T in the context of a RUL program on a concrete conjunction Q, resulting in new abstract conjunction: $proj(Q, T, R) = \langle Q, S', R' \rangle$, where $T \rightsquigarrow_R S$ and*

- $S' = S$, $R' = \emptyset$ *if* $S = fail$ *or* $\mathrm{Vars}(Q) = \emptyset$,
- *otherwise* $S' = t_1(x_1) \wedge \dots \wedge t_n(x_n)$ *where* $\mathrm{Vars}(Q) = \{x_1, \dots, x_n\}$ *and with* $types_S(x_i) = \{t_{i_1}, \dots, t_{i_k}\}$, $(R_i, t_i) = (R, t_{i_1}) \sqcap \dots \sqcap (R, t_{i_k})$ *for* $i = 1, \dots, n$. *In this case* $R' = R_1 \cup \dots \cup R_n$.

We now define the application of substitutions to abstract conjunctions:

$$\langle Q, T, R \rangle \theta = proj(Q\theta, T, R).$$

For example, using the *RUL* program R above, we have $t_3(f3 \circ z \circ z) \rightsquigarrow_R true \wedge t_3(z) \wedge t_3(z)$. We would thus have for $\theta = \{x/(f3 \circ z \circ z)\}$ that $\langle p(x), t_3(x), R \rangle \theta = \langle p(f3 \circ z \circ z), t_3(z), R \rangle$.

To extend the notion of instantiation pre-order of Definition 1.1.16 to abstract conjunctions the subset relation between types has to be considered:

Definition 7.1.6 *Let $A = \langle Q, T, R \rangle$, $A' = \langle Q', T', R' \rangle$ be abstract conjunctions in the RULE domain $(\mathcal{AQ}, \gamma, \mathrm{E})$. We call A' a RULE-*instance *of A, denoted by $A' \leq_{RULE} A$ iff*

1. *there exists a substitution θ such that $A\theta = \langle Q', T'', R'' \rangle$ and*

2. for all $x \in$ Vars(Q') with types$_{T'}(x) = \{t'\}$ and types$_{T''}(x) = \{t''\}$, we have $\tau_{R'}(t') \subseteq \tau_{R''}(t'')$.

We define $<_{RULE}$ and $=_{RULE}$ accordingly.

In the above example, $\langle p(f3 \circ z \circ z), t_3(z), R\rangle <_{RULE} \langle p(x), t_3(x), R\rangle$.

Definition 7.1.7 *An* unfolding rule *is a function which, given a definite* E*-program (D, E) and a goal $\leftarrow Q$, returns a finite, non-trivial*[5] *and possibly incomplete SLDE-tree for (D, E) and $\leftarrow Q$.*

Let τ be a finite (possibly incomplete) SLDE-tree for (D, E), $\leftarrow Q$. Let $\leftarrow G_1, \ldots, \leftarrow G_m$ be the goals in the leaves of the non-failing branches of τ. Let $\theta_1, \ldots, \theta_n$ be the computed answer substitutions of the SLDE-derivations from $\leftarrow Q$ to $\leftarrow G_1, \ldots, \leftarrow G_n$, respectively. Then the set of SLDE-resultants, *resultants(τ), is defined to be the set of clauses $\{Q\theta_1 \leftarrow G_1, \ldots, Q\theta_n \leftarrow G_n\}$.*

We can now define an *abstract unfolding* and an *abstract resolution* in the *RULE*-domain. When a conjunction of the *RULE* domain is unfolded, the information concerning the types of variables can be used to reduce the number of resultants. Additionally, we will use Definition 7.1.5 to determine the types of leaf conjunctions.

Definition 7.1.8 *Let (D, E) be a definite* E*-program, $\langle Q, T, R\rangle$ an abstract conjunction in the RULE domain $(\mathcal{AQ}, \gamma, \mathrm{E})$, U an unfolding rule. We define the abstract unfolding and resolution operations aunf$(_)$, ares$(_)$ as follows:*

- $aunf(\langle Q, T, R\rangle) = \{Q\theta \leftarrow B \mid Q\theta \leftarrow B \in resultants(U(Q)) \wedge T\theta \not\rightsquigarrow_R fail\}$
- $ares(\langle Q, T, R\rangle) = \{proj(B, T\theta, R) \mid Q\theta \leftarrow B \in aunf(\langle Q, T, R\rangle)\}$

The following is a generic algorithm for abstract partial deduction, which structures the abstract conjunctions to be specialised in a *global tree* (see also e.g. [95]). The abstract conjunction associated with some node L is denoted by *label(L)*. The algorithm is parametrised by a covering test *covered*, a *whistle* detecting potential infinite loops, an a generalisation operation *abstract* and a function *partition* to separate conjunctions into sub-conjunctions.

Algorithm 7.1.1 (*generic partial deduction algorithm*)

Input: a definite E-program (D, E), an abstract conjunction A of $\langle \mathcal{AQ}, \gamma, \mathrm{E}\rangle$.

Output: a set of abstract conjunctions $\mathcal{A}$, a specialised program D', a global tree λ

[5] A trivial SLDE-tree has a single node where no literal has been selected for resolution.

```
Initialisation: $\lambda$ := a "global" tree with a single unmarked node, labelled $A$
repeat
  pick an unmarked or abstracted leaf node $L$ in $\lambda$
  if $covered(L, \lambda)$ then mark $L$ as processed
  else
    if $whistle(L, \lambda) = \top$ then
        mark $L$ as abstracted
        $label(L) := abstract(L, \lambda)$
    else
      mark $L$ as processed
      for all $A \in ares(label(L))$ do
        for all $A' \in partition(A)$ do
          add a new unmarked child $C$ of $L$ to $\lambda$
          $label(C) := A'$
until all nodes are processed
output $\mathcal{A} := \{aunf(label(A)) \mid A \in \lambda\}$, $D' := \{aunf(A) \mid A \in \mathcal{A}\}$, and $\lambda$
```

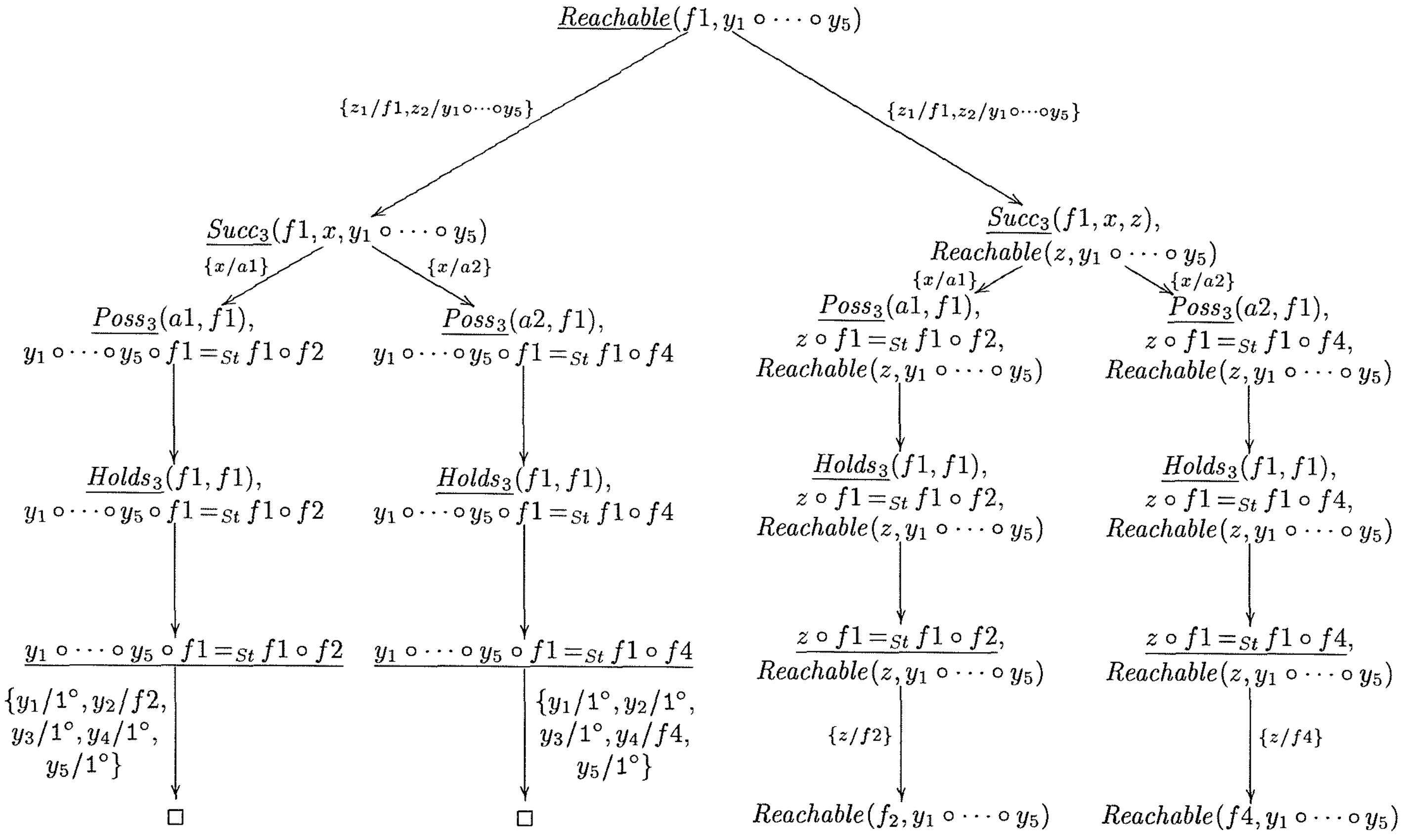
Now, we define a particular instance of the above algorithm which is suitable to solve conjunctive planning problems for Fluent Calculus domains. To this end we instantiate the Algorithm 7.1.1 in such a way that it imitates the Algorithm 4.2.1. In Section 7.2 we will prove that our instantiation is indeed correct.

Algorithm 7.1.2 (a partial deduction algorithm for the Fluent Calculus)

unfolding

According to Definition 7.1.8, $aunf(\_)$ requires an unfolding rule $U(\_)$. In terms of model checking the unfolding of a goal that represents a state of some (infinite) state system describes which immediate successor states the system may reach and how this can be achieved[6]. For the Fluent Calculus we use the following simple unfolding rule: Let $\langle Q, T, R\rangle$ be an abstract conjunction in the *RULE* domain and $(D, \mathtt{E})$ be a definite E-program. We define $U(Q)$ to be the maximal SLDE-tree τ such that every predicate p except the equality symbol defined by $\mathtt{E}$ ($=_{St}$ in the example) is selected at most once in every branch of τ. Clearly, the resulting SLDE-tree is finite if the equational theory is finitary.

[6] In general unfolding may also preselect states that do not have to be processed.



$\underline{Reachable}(f1, y_1 \circ \cdots \circ y_5)$
$\{z_1/f1, z_2/y_1 \circ \cdots \circ y_5\}$
$\{z_1/f1, z_2/y_1 \circ \cdots \circ y_5\}$
$\underline{Succ_3}(f1, x, y_1 \circ \cdots \circ y_5)$
$\underline{Succ_3}(f1, x, z), Reachable(z, y_1 \circ \cdots \circ y_5)$
$\{x/a1\}$
$\{x/a2\}$
$\{x/a1\}$
$\{x/a2\}$
$\underline{Poss_3}(a1, f1), y_1 \circ \cdots \circ y_5 \circ f1 =_{St} f1 \circ f2$
$\underline{Poss_3}(a2, f1), y_1 \circ \cdots \circ y_5 \circ f1 =_{St} f1 \circ f4$
$\underline{Poss_3}(a1, f1), z \circ f1 =_{St} f1 \circ f2, Reachable(z, y_1 \circ \cdots \circ y_5)$
$\underline{Poss_3}(a2, f1), z \circ f1 =_{St} f1 \circ f4, Reachable(z, y_1 \circ \cdots \circ y_5)$
$\underline{Holds_3}(f1, f1), y_1 \circ \cdots \circ y_5 \circ f1 =_{St} f1 \circ f2$
$\underline{Holds_3}(f1, f1), y_1 \circ \cdots \circ y_5 \circ f1 =_{St} f1 \circ f4$
$\underline{Holds_3}(f1, f1), z \circ f1 =_{St} f1 \circ f2, Reachable(z, y_1 \circ \cdots \circ y_5)$
$\underline{Holds_3}(f1, f1), z \circ f1 =_{St} f1 \circ f4, Reachable(z, y_1 \circ \cdots \circ y_5)$
$\underline{y_1 \circ \cdots \circ y_5 \circ f1 =_{St} f1 \circ f2}$
$\underline{y_1 \circ \cdots \circ y_5 \circ f1 =_{St} f1 \circ f4}$
$\underline{z \circ f1 =_{St} f1 \circ f2}, Reachable(z, y_1 \circ \cdots \circ y_5)$
$\underline{z \circ f1 =_{St} f1 \circ f4}, Reachable(z, y_1 \circ \cdots \circ y_5)$
$\{y_1/1^\circ, y_2/f2, y_3/1^\circ, y_4/1^\circ, y_5/1^\circ\}$
$\{y_1/1^\circ, y_2/1^\circ, y_3/1^\circ, y_4/f4, y_5/1^\circ\}$
$\{z/f2\}$
$\{z/f4\}$
$\square$
$\square$
$Reachable(f_2, y_1 \circ \cdots \circ y_5)$
$Reachable(f4, y_1 \circ \cdots \circ y_5)$

Figure 7.1: The incomplete SLDE-tree for Example 7.1.2.

Example 7.1.2 (Example 6.2.2 continued) *We define Δ_{Σ_p} as the RULE domain $(\mathcal{AQ}, \gamma, \mathtt{AC1})$ where every abstract conjunction $C \in \mathcal{AQ}$ is of the form $\langle Reachable(g, v), T, R\rangle$. Thereby $v = y_1 \circ \cdots \circ y_5$ where each variable y_i for $i = 1, \ldots, 5$ is restricted by the type conjunction T such that $t_{fi} \in atoms(T)$. The term g is of sort St and for all variables $x \in Vars(g)$, $t_f(x) \in atoms(T)$ for some fluents $f \in Fl_{\Sigma_p}$. R consists of predicates (t_f) for each fluent $f \in Fl_{\Sigma_p}$:*

$$t_f(1^\circ).$$
$$t_f(f).$$
$$t_f(x \circ y) \leftarrow t_f(x) \wedge t_f(y).$$

The SLDE-tree computed by U (using the leftmost selection rule) for the abstract conjunction $\langle Reachable(f1, y_1 \circ \cdots \circ y_5), t_{f1}(y_1) \wedge \cdots \wedge t_{f5}(y_5), R\rangle$ wrt the definite E*-program corresponding to domain $\mathcal{D}_p$ is depicted in Figure 7.1 (we use Definition 6.12 of Reachable$(_, _)$).* □

With this particular unfolding rule we ensure that every goal of the form *Reachable*(g_1, g_2) is unfolded in such a way that every incomplete leaf node of the resulting SLDE-tree consists of a subgoal of the same form. In the next section we will use this property to prove completeness of our partial deduction algorithm by showing a correspondence between the subgoals of the form *Reachable*(g_1, g_2) with pseudo-markings in Petri net algorithms.

covered

In partial deduction algorithms the function *covered* is used to check whether some subgoal has been already encountered before. This ensures termination of the analysis and, if *covered* is defined appropriately, it often may reduce the number of subgoals to be visited, considerably. Here, we define *covered* simply using the equivalence relation on abstract conjunctions to ensure termination: Let L be a node labelled by an abstract conjunction in the *RULE* domain $(\mathcal{AQ}, \gamma, \mathtt{E})$ and λ a tree labelled by elements of $\mathcal{AQ}$. Then we define *covered*(L, λ) as true iff there is an ancestor L' of L such that $label(L') =_{RULE} label(L)$.

Instead of using this check for equivalence with ancestors, only, we may use a more refined approach. On one hand, we may compare the subgoal L to *all* subgoals L' in the tree. On the other hand, in $\mathcal{FC}_{PL}$ domains it is sufficient that $label(L') \leq_{RULE} label(L)$ for L to be covered. This is possible due to the monotonicity property of Petri nets (see [39] and [94] for details). It has been shown that this more sophisticated approach may help reducing the size of the final tree enormously.

whistle

The function *whistle* is used to detect whether the repeated extension of an incomplete derivation (by unfolding the last subgoal) might result in an infinite derivation. Due to the undecidability of the halting problem, which can be encoded as a definite logic program, it is not always possible to precisely predict infinite derivations[7]. Consequently, to ensure that all infinite SLDE-derivations are detected (which is required for the partial deduction algorithm to terminate) *whistle* will have to recognise unfolding of some subgoals as dangerous, even when they do not result in infinite derivations. To this end, for partial deduction this "over-approximation" is is often defined in terms of a *homeomorphic embedding relation* [140]. The homeomorphic embedding relation is defined on the set of atoms of the definite logic program. In principle, we could employ this relation in a Fluent Calculus setting as well. However, as we use an equational theory, it is more natural to extend the homeomorphic embedding accordingly:

Definition 7.1.9 *Let $A = \langle Q, T, R\rangle, A' = \langle Q', T', R'\rangle$ be abstract conjunctions in the RULE domain $(\mathcal{AQ}, \gamma, \mathtt{E})$. For the purposes of this definition we suppose that $\wedge$ is handled by* E *as an associative, commutative and idempotent function symbol. Let $=_S$ be the equality symbol defined by* E. *We say that A is* homeomorphically embedded *in A', $A \trianglelefteq_E A'$, iff $Q \trianglelefteq_E Q'$ where $\trianglelefteq_E$ on expressions is inductively defined as follows:*

1. *$x \trianglelefteq_E y$ if x, y variables with $\tau_R(types_T(x)) \subseteq \tau_{R'}(types_{T'}(y))$*
2. *$c \trianglelefteq_E y$ for all variables y and constants c, with $c \in \tau_R(types_{T'}(y))$*
3. *$r \trianglelefteq_E f(s_1, \ldots, s_n)$ if there exists s' with $\mathtt{E} \models s' =_S r$ and there exists $f', s'_1, \ldots, s'_m$ with $\mathtt{E} \models f'(s'_1, \ldots, s'_m) =_S f(s_1, \ldots, s_n)$ such that $s' \trianglelefteq_E s'_i$ for some $1 \leq i \leq m$*
4. *$f(r_1, \ldots, r_m) \trianglelefteq_E g(s_1, \ldots, s_n)$ if there exists $f', r'_1, \ldots, r'_k$ with $\mathtt{E} \models f'(r'_1, \ldots, r'_k) =_S f(r_1, \ldots, r_m)$ and $\mathtt{E} \models f'(s'_1, \ldots, s'_k) =_S g(s_1, \ldots, s_n)$ and $\forall i \in \{1, \ldots, k\} : r'_i \trianglelefteq_E s'_i$.*

Note that for point 3. we may have $n = 0$, and for point 4. m, n, k can be 0. Intuitively, $s \trianglelefteq_E t$, means that we can obtain s from t by "striking out" certain sub-terms and by using the equational theory to re-write s and t. E.g., for E equals ACl, we have $f \circ g \trianglelefteq_E g \circ h \circ f$ and $f \circ g \trianglelefteq_E g \circ h \circ x$ for $types_{T'}(x) = t_f$ where R' is a RUL program as in Example 7.1.2. Note that in general, an equational theory might define two terms also to be equal even when they are constructed using different function symbols. This is taken into account in Definition 7.1.9 by considering a function f' in point 3. and 4.

Of course, $\trianglelefteq_E$ will be quite expensive to compute ([102]). Although in the context of AC1 and in particular $\mathcal{FC}_{PL}$ we may introduce a lot of optimisations to obtain an efficient implementation (e.g. sorting fluents and defining a normal form for terms), there will be a trade-off between the computational effort to

[7]Note that for $\mathcal{FC}_{PL}$ domains it is in fact possible to predict precisely when a derivation is potentially infinite while already for simple $\mathcal{FC}$ domains it is not.

detect infinite derivations, which may reduce the size of the resulting tree, and the speed of the partial deduction algorithm.

The original homeomorphic relation (and, if $\subseteq$ is a well-quasi order on the set of types, also our refined homeomorphic relation of Definition 7.1.9) is a well-quasi order and can thus be used to ensure termination of program specialisation techniques [140]. We apply $\trianglelefteq_E$ as follows. Let L be a node labelled by an abstract conjunction in the $RULE$ domain $(\mathcal{AQ}, \gamma, \mathrm{E})$ and λ a tree labelled by elements of $\mathcal{AQ}$. We define $whistle_{\trianglelefteq_E}(L, \lambda) = \top$ iff L is not marked as abstracted and there is an ancestor L' of L such that $label(L') \trianglelefteq_E label(L)$.

abstract

If an incomplete derivation was recognised as being "dangerous" by the function *whistle*, some of the possibly infinite derivations need to be represented in a finite way. However, for most applications of partial deduction considered so far it is sufficient to represent only the (possibly infinite) set of subgoals – the information about their order as described by any particular derivation is discarded. To represent this set of subgoals in a finite way a single, more general subgoal is used. This subgoal is chosen based on the condition that many of the subgoals occurring on an infinite derivation (as predicted by *whistle*) should be instances of the new more general subgoal. On one hand, this condition should ensure termination by allowing only a finite number of generalisation steps before all subgoals are covered. On the other hand, if the new subgoal is chosen such that it covers also many subgoals which do not occur on an infinite derivation (as predicted by *whistle*), important information about the actually reachable subgoals might be lost. In classical partial deduction the new subgoal is an atom which represents the most specific generalisation wrt a subset of the subgoals encountered on the "dangerous" derivation. As we have argued, to adapt the approach to the Fluent Calculus, we have to increase the precision of the most specific generalisation by introducing a type system. Since our abstraction mechanism may in general introduce new types during the partial deduction process we have to ensure that abstractions of types may occur only finitely often, i.e. the type system should be *well-founded.* Additionally, to ensure that $\trianglelefteq_E$ is a well-quasi order, the relation $\subseteq$ must be a well-quasi order on the type system.

Definition 7.1.10 *Let* E *be an equational theory and* $\mathcal{T}$ *a set of tuples* (t, R) *where* R *is a* RUL *program and* t *a predicate defined in* R. *We call* $\mathcal{T}$ *a* well-formed type system *iff*

1. *there is no infinite sequence* $(t_1, R_1), (t_2, R_2), \ldots$ *of elements of* $\mathcal{T}$ *such that* $\tau_{R_i}(t_i) \subset \tau_{R_{i+1}}(t_{i+1})$ *for all* $i \geq 1$, *and*
2. *there is no infinite sequence* $(t_1, R_1), (t_2, R_2), \ldots$ *of elements of* $\mathcal{T}$ *such that* $\tau_{R_i}(t_i) \not\subseteq \tau_{R_j}(t_j)$ *for all* $i \neq j$.

Now we can define a generalisation and a most specific generalisation in the $RULE$-domain for some well-formed type system:

Definition 7.1.11 *Let* E *be an equational theory, $\mathcal{T}$ be a well-formed type system and $\mathcal{A}$ a set of abstract conjunctions in $(\mathcal{AQ}, \gamma, \mathrm{E})$. The abstract conjunction $M = \langle Q, T, R\rangle$ is called a* $RULE$-generalisation *of $\mathcal{A}$ wrt $\mathcal{T}$ iff*

1. *for all $t(x) \in T$ we have $(t, R) \in \mathcal{T}$,*
2. *for all $A \in \mathcal{A}$, $A \leq_{RULE} M$.*

Furthermore, M is called a most specific $RULE$-generalisation *of $\mathcal{A}$ wrt $\mathcal{T}$, denoted by $M \in msg_{\mathcal{T}}(\mathcal{A})$, iff there exists no M' such that conditions 1, 2 hold for M' and $M' <_{RULE} M$.*

For example, $\mathcal{A} = \{\langle f3, true, \emptyset\rangle, \langle f3 \circ f3, true, \emptyset\rangle\}$ and using the single type defined by the RUL program R for t_3 we get $msg_{\{(t_3,R)\}}(\mathcal{A}) = \{\langle f3 \circ x, t_3(x), R\rangle\}$. Again, for other equational theories than (AC1) and more complicated type systems a most specific generalisation might be difficult to compute (and may not be unique). To accelerate convergence (and to simplify our completeness proof for CPP later on), we actually choose an element $M' = \langle Q, T, R\rangle$ of $msg_{\mathcal{T}}(\mathcal{A})$ and then remove the maximum number of sub-terms from Q so that the resulting abstract conjunction is still more general than M' (in the sense of $\leq_{RULE}$). We will denote the result by $nmsg_{\mathcal{T}}(\mathcal{A})$. For example, we would instead of using $M' = \langle f3 \circ x, t_3(x), R\rangle$ use the more general $nmsg_{\mathcal{T}}(\mathcal{A}) = \langle x, t_3(x), R\rangle$. This loses some precision, but convergence is accelerated, and actually no vital information for the CPP is lost!

Let L be a node labelled by an abstract conjunction in the $RULE$ domain $(\mathcal{AQ}, \gamma, \mathrm{E})$ and λ a tree labelled by elements of $\mathcal{AQ}$. Let $\mathcal{L}$ denote the set of all ancestors of L in λ such that $L' \in \mathcal{L}$ iff $label(L') \trianglelefteq_E label(L)$. Furthermore, let $\mathcal{A}$ denote the set of labels of $\mathcal{L}$. Then we define $abstract(L, \lambda) = nmsg_{\mathcal{T}}(\mathcal{A})$.

partition

For definite logic programs in general, the unfolding of some atom representing a subgoal might result in a tree where some leaf is labelled by a conjunction of atoms rather than by a single atom. However, by using the unfolding rule as defined above all leaf nodes are guaranteed to be single atoms in Fluent Calculus domains. Consequently, we define $partition(\_)$, which is used in classical partial deduction to split the conjunctions into atoms, simply as follows:

Definition 7.1.12 *Let $A = \langle Q, T, R\rangle$ be an abstract conjunction in $(\mathcal{AQ}, \gamma, \mathrm{E})$ and $\mathcal{T}$ a well-formed type system. Then $partition(A) = atoms(nmsg_{\mathcal{T}}(\{A\}))$.*

To solve the problem of conjunctions appearing in leaf nodes in a more general way, we may follow alternatively the approach of *conjunctive partial deduction* [23].

7.2 Completeness wrt $\mathcal{FC}_{PL}$

In Section 4.3 we have shown that Petri net algorithms can be used to decide temporal properties of propositional Fluent Calculus domains. In particular, to every propositional $\mathcal{FC}$ domain with completely defined initial state there exists a bisimilar Petri net. Furthermore, the conjunctive planning problem for the propositional $\mathcal{FC}$ can be expressed as a formula in the temporal logic CTL (CTL respects bisimulation). The same formula is known to describe coverability properties of Petri nets. Coverability problems can be decided using the Karp-Miller tree (see Section 4.2). In this section we will show that this tree can also be generated by the partial deduction Algorithm 7.1.1 using the instantiations of Algorithm 7.1.2. By doing so, we prove that the proposed partial deduction method is complete for conjunctive planning problems for $\mathcal{FC}_{PL}$ domains.

Theorem 7.2.1 *Let $\mathcal{D}$ be a $\mathcal{FC}_{PL}$ domain description wrt some signature Σ, Δ the RULE domain $(\mathcal{AQ}, \gamma, \mathtt{AC1})$ where every abstract conjunction $C \in \mathcal{AQ}$ is of the form $\langle Reachable(g, v), T, R\rangle$. Thereby $v = y_1 \circ \cdots \circ y_n$ where for each $f_i \in \{f_1, \ldots, f_n\} = T_{\Sigma,Fl}(\emptyset)$, $t_{f_i}(y_i) \in atoms(T)$ (with $i = 1, \ldots, n$). Let the term g be of sort St and for all variables $x \in Vars(g)$, $t_f(x) \in atoms(T)$ for some fluent $f \in T_{\Sigma,Fl}(\emptyset)$. R consists of predicates (t_f) for each fluent $f \in T_{\Sigma,Fl}(\emptyset)$:*

$$t_f(1^{\circ}).$$
$$t_f(f).$$
$$t_f(x \circ y) \leftarrow t_f(x) \wedge t_f(y).$$

Furthermore, let St_{init} be some ground term of sort St and $\mathcal{M}$ a model of $\mathcal{D}$ where $state^{\mathcal{M}}(s0^{\mathcal{M}}) =_{St} St_{init}$. Then the partial deduction algorithm applied to the definite $\mathtt{E}$-program $(P, \mathtt{E})$ representing $\mathcal{D}$, Δ and $A = \langle Reachable(St_{init}, v_1 \circ \cdots \circ v_n), t_{f_1}(v_1) \wedge \cdots \wedge t_{f_n}(v_n), R\rangle$ will produce a global tree λ which is isomorphic to a Karp-Miller coverability tree of the Petri net $\mathcal{P}(\mathcal{D}, \mathcal{M})$.

We will now formally prove that the algorithm 7.1.1 with the instantiation of Algorithm 7.1.2 can be used to decide coverability problems. For this we need to establish a link between pseudo-markings in the Karp-Miller tree and abstract conjunctions produced by partial deduction.

Let $\mathcal{D}$ be some $\mathcal{FC}_{PL}$ domain wrt signature Σ, $(D, \mathtt{E})$ the definite $\mathtt{E}$-program corresponding to $\mathcal{D}$ and $\mathcal{M}$ some model of $\mathcal{D}$. Let $T_{\Sigma,Fl}(\emptyset) = \{f_1, \ldots, f_n\}$ be the fluents and $T_{\Sigma,Act}(\emptyset) = \{a_1, \ldots, a_m\}$ the actions defined in Σ. According to Section 4.3, the Petri net $\mathcal{P}(\mathcal{D}, \mathcal{M}) = (P, T, E, W, m_0)$ is given by associating an unique place $P(f) \in P$ to each $f \in T_{\Sigma,Fl}(\emptyset)$. To every state update axiom $\mathcal{D}$, and thereby to every pair of clauses clause of the form

$$Poss_3(x, z) \leftarrow x = a, Holds_3(St_a^- \circ St_a^=, z).$$

$$Succ_3(z_1, x, z_2) \leftarrow x = a, Poss_3(x, z_1), z_2 \circ St_a^- =_{St} z_1 \circ St_a^+.$$

in $(D, \mathtt{E})$, for $a \in T_{\Sigma,Act}(\emptyset)$ a transition $T(a) \in T$ is associated. The set $\mathtt{E}$ of edges and the weight function W are defined wrt the fluents appearing in the state update axioms as described in Section 4.3.

Let Δ be the *RULE* domain $(\mathcal{AQ}, \gamma, \mathtt{AC1})$ where every abstract conjunction $C \in \mathcal{AQ}$ is of the form $\langle \mathit{Reachable}(g, v), T, R\rangle$ where $v = y_1 \circ \cdots \circ y_n$, g is a term of sort St and for all variables $x \in \mathit{Vars}(g)$ and $y \in \mathit{Vars}(v)$ $t_f(x) \in \mathit{atoms}(T)$ and $t_{f'}(y) \in \mathit{atoms}(T)$ for some fluents $f, f' \in T_{\Sigma,Fl}(\emptyset)$. R consists of the clauses defining (t_f) for each fluent $f \in T_{\Sigma,Fl}(\emptyset)$. Then, we define the pseudo-marking C^μ such that for each $f \in T_{\Sigma,Fl}(\emptyset)$:

$$C^\mu(f) = \begin{cases} \omega & \text{if } \exists x.\,(x \in \mathit{Vars}(g) \land t_f(x) \in \mathit{atoms}(T)) \\ |g, f| & \text{otherwise} \end{cases}$$

Accordingly, the initial marking corresponding to some ground state term St_{init} is given by

$$\langle \mathit{Reachable}(St_{init}, y_1 \circ \cdots \circ y_n), t_{f_1}(y_1) \land \cdots \land t_{f_n}(y_n), R\rangle^\mu$$

Additionally, we associate with every pseudo-marking m and *RULE* program R as defined above an abstract conjunction $m^\alpha = \langle \mathit{Reachable}(g, v), T, R\rangle$ such that for every fluent $f \in T_{\Sigma,Fl}(\emptyset)$, the term g contains f exactly $m(P(f))$ times if $m(P(f)) \neq \omega$. For every $f \in T_{\Sigma,Fl}(\emptyset)$ with $m(P(f)) = \omega$, g contains a variable x and T a type declaration $t_f(x)$. Additionally, $v = y_1 \circ \cdots \circ y_n$ and $t_{f_i}(y_i) \in \mathit{atoms}(T)$ for all $1 \leq i \leq n$.

To prove that the tree generated by our partial deduction algorithm is isomorphic to the Karp-Miller tree, we show that any node generated by the partial deduction procedure is also generated by the Karp-Miller procedure and vice versa. To this end we establish correspondences between the relations used in the definition of Algorithm 7.1.2 and the relations used in the defintion of Algorithms 4.2.1. Then we show that the steps performed by one algorithm are also performed by the other.

Lemma 7.2.2 *Let L_1,L_2 be some nodes of the tree λ which is labelled by abstract conjunctions of Δ. Let $C_1 = \langle \mathit{Reachable}(g_1, v), T_1, R\rangle = \mathit{label}(L_1)$ and $C_2 = \langle \mathit{Reachable}(g_2, v), T_2, R\rangle = \mathit{label}(L_2)$. Then $C_1 =_{RULE} C_2$ iff ${C_1}^\mu = {C_2}^\mu$.*

Proof 7.2.2 This follows using the mappings $\_^\alpha$ and $\_^\mu$ between markings and abstract conjunctions as defined above and the fact that $(C^\mu)^\alpha =_{RULE} C$ for markings $C = \langle \mathit{Reachable}(g, v), T, R\rangle$ and from the Definition 7.1.6, $C_1 =_{RULE} C_2$ iff for all fluents f holds either 1. the number of f in g_1 and g_2 must be equal, or 2. there are variables x in g_1 and y in g_2 such that $t_f(x) \in T_1$ and $t_f(y) \in T_2$. □

Lemma 7.2.3 *Let L be some node of the tree λ which is labelled by abstract conjunctions of Δ. Let $C = \langle \mathit{Reachable}(g, v), T, R\rangle = \mathit{label}(L)$ and $C_0, C_1, \ldots, C_n$*

is the sequence of labels of the ancestors of L in λ where C_0 is the label of the root node. whistle(L, λ) = $\top$ iff there is some L_k labelled C_k, $0 \leq k \leq n$, with ${C_k}^{\mu} \leq C^{\mu}$.

Proof 7.2.3 According to the definition, *whistle* returns $\top$ iff there is some ancestor L_k of L labelled C_k such that $C_k \trianglelefteq_E C$. If g does not contain a variable of type t_f for fluent $f \in T_{\Sigma,Fl}(\emptyset)$ and $C_k \trianglelefteq_E C$, then from case 4 of the definition of $\trianglelefteq_E$ follows ${C_k}^{\mu}(P(f)) \leq C^{\mu}(P(f))$, otherwise, by case 1 and 2, follows ${C_k}^{\mu}(P(f)) \leq C^{\mu}(P(f))$ if the corresponding subterm of C_k consists only of f or some variable of type t_f. Now, let $C_k = \langle Reachable(g_k, v_k), T_k, R\rangle$ be an abstract conjunction in Δ_{Σ} with $C_k \trianglelefteq_E C$. Then either, for each fluent f, case 4 can be applied ${C_k}^{\mu}(P(f))$ times, or case 1 or 2 can be applied where x is of type t_f. Considering the equational theory AC1, case 4 can only be applied if both function symbols are identical, i.e. $C^{\mu}(P(f))$ contains f at least ${C_k}^{\mu}(P(f))$ times. On the other hand, case 1 or 2 can be applied only if $C^{\mu}(P(f)) = \omega$ and if the corresponding subterm of C_k consists only of copies of f or of a variable of type t_f. □

Lemma 7.2.4 *Let L be some node of the tree λ which is labelled by abstract conjunctions of Δ. Let $C = \langle Reachable(g, v), T, R\rangle$ = label(L) and $\{C_1, \ldots, C_n\}$ is the sequence of labels of ancestors of L in λ such that $C_i \trianglelefteq_E C$ for all $1 \leq i \leq n$. Let the type system $\mathcal{T}$ consist of all pairs (t, R) such that t is a predicate in R. C' = abstract(L, λ) iff ${C'}^{\mu} = m'$ and m' is defined as follows: if for some fluent f there exists an ancestor C_k of C in λ such that ${C_k}^{\mu} < C^{\mu}$ and ${C_k}^{\mu}(f) < C^{\mu}(f)$, $m'(f) = \omega$, otherwise $m'(f) = C^{\mu}(f)$.*

Proof 7.2.4 Note that $\mathcal{T}$ is a finite set such that for any two $(t_1, R), (t_2, R) \in \mathcal{T}$, $\tau_R(t_1) \cap \tau_R(t_2) = \emptyset$. Furthermore, every $\tau_R(t)$ with $(t, R) \in \mathcal{T}$ consists only of terms constructed by combining one particular fluent $f \in T_{\Sigma,Fl}(\emptyset)$ using $\circ$. Hence, from the definition of *abstract* and $nmsg_{\mathcal{T}}$ follows that C' contains a variable of type t_f iff there is some ancestor L_k of L labelled C_k such that $C_k \trianglelefteq_E C$ and ${C_k}^{\mu}(f) < C^{\mu}(f)$. On one hand, condition 2 in Definition 7.1.11 ensures that C and C_k are both instances of $C' = \langle Reachable(g', v), T', R\rangle$. This is true iff ${C_k}^{\mu}(f) < {C'}^{\mu}(f)$ and $C^{\mu}(f) < {C'}^{\mu}(f)$ and hence, g' must contain a variable of type t_f if ${C_k}^{\mu}(f) < C^{\mu}(f)$. Furthermore, from definition of $nmsg_{\mathcal{T}}$, a fluent f must not occur in g' if g' contains a variable of type t_f. On the other hand, from Definition 7.1.11 follows, that g' must not contain a variable of type t_f if ${C_k}^{\mu}(f) = C^{\mu}(f)$ for all ancestors with label C_k and ${C_k}^{\mu} < C^{\mu}$. In this case, since $C \trianglelefteq_E C'$, the same number of copies of fluent f occur in g' as in g. □

Lemma 7.2.5 *Let L_1,L_2 be some nodes of the tree λ which is labelled by abstract conjunctions of Δ. Let $C_1 = \langle Reachable(g_1, v), T_1, R\rangle$ = label(L_1) and $C_2 = \langle Reachable(g_2, v), T_2, R\rangle$ = label(L_2). $C_2 \in$ partition(ares(aunf(L_1))) iff there is an action a such that ${C_1}^{\mu}[T(a)\rangle {C_2}^{\mu}$.*

Proof 7.2.5 The procedures *ares*() and *aunf*() can be simplified, since a variable may never have two or more types, conjunctions of types do not have to be computed. *ares*() and *aunf*() unfold and ensure type of variables, only. According to the unfolding rule, which unfolds until every occurring predicate is unfolded once, an atom $Reachable(g_1, v)$ can be unfolded iff there is an action a such that $Poss_3(a, g_1)$ is fulfilled (i.e. it is unfolded into the empty goal). If $Reachable(g_1, v)$ can be unfolded, then the single subgoal $Reachable(g_1', v)$ where $g_1 =_{St} v \circ St_a^-$ and $g_1' =_{St} v \circ St_a^+$ is generated. According to the AC1 unification, $Poss_3(a, g_1)$ is fulfilled iff $g_1 =_{St} v \circ St_a^- \circ St_a^=$ iff either $|St_a^-, f| + |St_a^=, f| \leq |g_1, f|$ or there is a variable of type t_f in g_1, for each $f \in T_{\Sigma,Fl}(\emptyset)$. Consequently, by definition of W for $\mathcal{P}(\mathcal{D}, \mathcal{M})$ in Section 4.3, $g_1 =_{St} v \circ St_a^- \circ St_a^=$ iff $T(a)$ is enabled in ${C_1}^\mu$. Furthermore, if g_1 does not contain a variable of type t_f, it holds $|g_1', f| = |g_1, f| - |St_a^-, f| + |St_a^+, f|$. Otherwise, the co-domain of any *mgeu* for g_1 and $v \circ St_a^-$ must contain a variable x such that $t_f(x)$. Let T_1' be the set of such type declarations. Then, with $C_1' = \langle Reachable(g_1', v), T_1', R\rangle$, it follows ${C_1}^\mu[T(a)\rangle {C_1'}^\mu$. However, g_1' may contain copies of a fluent f even if there is a variable x in g_1' with $t_f(x) \in T_1'$. Using the partition function with $nmsg_{\mathcal{T}}$, g_2 is defined as g_1' where such additional copies are removed. By this it is ensured that for every marking m with ${C_1}^\mu[T(a)\rangle m$, $m^\alpha =_{RULE} C_2$. □

Finally, using the lemmas above we can prove the main theorem by showing that our partial deduction procedure imitates each step of the Karp-Miller procedure.

Proof 7.2.1 **(Theorem 7.2.1)** According to definition of Algorithm 4.2.1 holds $U = \{node(r, m_0)\}$ where we set $m_0 = A^\mu$. Now, we show the correspondence between each step in Algorithm 4.2.1 and Algorithm 7.1.2. First, both algorithms terminate if no unprocessed nodes remain. Second, in Algorithm 7.1.2 a selected node L is marked processed if $covered(L, \lambda)$ is true. Let (k, m) be the selected node by Algorithm 4.2.1 with $m = label(L)^\mu$. According to Lemma 7.2.2, $covered(L, \lambda)$ iff there is an ancestor node (k_1, m_1) with $m = m_1$. In this case (k, m) is marked processed by Algorithm 4.2.1 (i.e. removed from the list of unprocessed nodes). Third, Algorithm 7.1.2 calls $abstract(L, \lambda)$ if $whistle(L, \lambda) = \top$. Using Lemma 7.2.3 $whistle(L, \lambda) = \top$ iff there is some ancestor L_k of L such that $label(L_k)^\mu \leq label(L)$. In Algorithm 4.2.1, abstraction is performed for every ancestor (k_1, m_1) of (k, m) with $m_1 < m$. Since the case $m_1 = m$ and $label(L_k) =_{RULE} label(L)$, respectively, has already been checked, it remains to be shown, that $C' = abstract(L, \lambda)$ iff ${C'}^\mu = m'$ and m' is defined as follows: if for some fluent f there exists an ancestor L_k of L in λ s.t. $label(L_k)^\mu < label(L)^\mu$ and $label(L_k)^\mu(f) < label(L)^\mu(f)$, $m(f) = \omega$, otherwise $m(f) = label(L)^\mu(f)$. This has been shown in Lemma 7.2.4. Finally, from Lemma 7.2.5 follows that $C_2 \in partition(ares(aunf(L_1)))$ iff there is an action $\mathcal{A}$ such that ${C_1}^\mu[T(a)\rangle {C_2}^\mu$.
□

7.3 Applications

Conjunctive Planning Problems in $\mathcal{FC}_{PL}$

Now we are ready to apply our partial deduction Algorithm 7.1.2 to $\mathcal{FC}_{PL}$ domains. To encode conjunctive planning problems we use the scheme of Clause 6.18. Since the definition of predicate *Reachable*(_, _) by the Clauses 6.12 describes a forward interpreter the unfolding performed by our algorithm will only depend on the first parameter, never on the second. Consequently, instead of encoding the goal state in the second parameter we may simply apply our algorithm to $\langle Reachable(g_1, v), T, R\rangle$, where g_1 is the ground term describing the initial state, v describes the set of all state terms, i.e. it is a term of the form $x_1 \circ \ldots \circ x_n$ which contains precisely one variable x_i for each fluent f_i of the domain. Accordingly, T is a type conjunction $t_1(x_1) \wedge \cdots \wedge t_n(x_n)$ and R contains a RUL program describing t_f as in Example 6.2.2 for each fluent f. Then, the global tree λ generated by our partial deduction algorithm will in fact contain the solutions to *all* CPP's for the initial state g_1. To verify if there is a plan leading from g_1 to a particular state $g_2 \circ z$ we simply check for λ containing a node L such that $\langle Reachable(g_2, v), T, R\rangle \trianglelefteq_E label(L)$ (where v, T, R are defined as above). If there is such a node L, then the CPP has a solution.

Example 7.3.1 (Example 6.2.2 continued) *We define the type system* $\mathcal{T} = \{(t_f, R) \mid f \in Fl_{\Sigma_p}\}$. *Additionally to the actions of Example 6.2.2 and the domain independent clauses, let the initial state be defined as* $g_1 = f1$. *Then, the tree of Figure 7.2 is generated by our partial deduction algorithm with input* Σ_p, Δ_{Σ_p} *and initial abstract conjunction* $\langle Reachable(f1, y_1 \circ \cdots \circ y_5), t_{f1}(y_1) \wedge \cdots \wedge t_{f5}(y_5), R\rangle$.

To simplify the picture the RUL programs and some of the type information have not been represented. The RUL programs do not change in this example for the initial abstract conjunction and the type information has been depicted as follows: v *represents* $y_1 \circ \cdots \circ y_5$ *and the type conjunction* T_j *for each node labelled* $\langle Reachable_j(u, v), T_j, R\rangle$ *contains* $T = t_{f1}(y_1) \wedge \cdots \wedge t_{f5}(y_5)$. $t_{f1}, \ldots, t_{f5}$ *are defined by the corresponding RUL programs of Example 7.1.2.*

For example, we can conclude from the tree that every fluent can be generated arbitrary often. But, e.g., it is impossible to reach a state where both, fluents $f2$ *and* $f4$ *exist. Furthermore, we can conclude that it is impossible to reach a state containing* $f4$ *from the state* $f2$ *(Example 6.3.1) as none of the descendent nodes of* $\langle Reachable(f2, v), T, R\rangle$ *is labelled by an abstract conjunction* L *such that* $\langle Reachable(f4, v), T, R\rangle \trianglelefteq_E L$. □

How to compute a plan

If a CPP has a solution as indicated by some node L then the substitutions on the path from the root node to L describe the actions which have to be

$$\begin{tikzcd}
 & & & \langle \mathit{Reachable}_1(f1,v),T,R\rangle \arrow[dll, "a1"'] \arrow[drr, "a2"] & & & \\
 & \langle \mathit{Reachable}_2(f2,v),T,R\rangle \arrow[d, "a3"'] & & & & \langle \mathit{Reachable}_3(f4,v),T,R\rangle \arrow[d, "a5"] & \\
 & \langle \mathit{Reachable}_{10}(f3\circ f3,v),T,R\rangle \arrow[d, "a4"'] & & & & \langle \mathit{Reachable}_5(f5\circ f5,v),T,R\rangle \arrow[d, "a6"] & \\
 & \langle \mathit{Reachable}_{11}(f2\circ x_1,v),T\wedge t_{f3}(x_1),R\rangle \arrow[dl, "a3"'] \arrow[dr, "a4"] & & & & \langle \mathit{Reachable}_6(f4\circ x_1,v),T\wedge t_{f5}(x_1),R\rangle \arrow[dl, "a6"'] \arrow[dr, "a5"] & \\
\langle \mathit{Reachable}_{12}(x_1,v),T\wedge t_{f3}(x_1),R\rangle \arrow[d, "a4"'] & & \textcircled{13} \arrow[loop below, "{a3,a4}"] & & \textcircled{8} \arrow[loop below, "{a5,a6}"] & & \langle \mathit{Reachable}_7(x_1,v),T\wedge t_{f5}(x_1),R\rangle \arrow[d, "a6"] \\
\langle \mathit{Reachable}_{14}(x_1\circ x_2,v),T\wedge t_{f3}(x_1)\wedge t_{f2}(x_2),R\rangle \arrow[loop below, "{a3,a4}"] & & & & & & \langle \mathit{Reachable}_9(x_1\circ x_2x,v),T\wedge t_{f5}(x_1)\wedge t_{f4}(x_2),R\rangle \arrow[loop below, "{a5,a6}"]
\end{tikzcd}$$

$$\langle \mathit{Reachable}_{13}(x_1\circ x_2,v),T\wedge t_{f3}(x_1)\wedge t_{f2}(x_2),R\rangle = \textcircled{13} \qquad \textcircled{8} = \langle \mathit{Reachable}_8(x_1\circ x_2,v),T\wedge t_{f5}(x_1)\wedge t_{f4}(x_2),R\rangle$$

Figure 7.2: Example 7.3.1.

executed. However, this sequence of actions does not represent a correct plan since it might contain subsequences that have to be executed several times to actually reach some of the states characterised by *label*(L). More precisely, every introduction of a new variable by ***abstract*** on the path from the root node to L is justified by some action sequence which can be executed arbitrary often. E.g., in Figure 7.2 ***abstract*** has introduced a new variable in node $\langle \mathit{Reachable}_{11}(f2 \circ x_1, v), T \wedge t_{f3}(x_3), R\rangle$ based on the fact that executing the action sequence $a3, a4$ increases the number of copies of $f3$ in any state that contains at least one $f2$. Furthermore, even when an action sequence can be executed arbitrary often, the number of possible executions might depend on the number of executions of some other action sequence that can be executed arbitrary often. E.g., in Figure 7.2 the action $a4$ may be executed arbitrary often in the states represented by $\langle \mathit{Reachable}_{11}(f2 \circ x_1, v), T \wedge t_{f3}(x_3), R\rangle$, thereby creating an unlimited number of copies of $f2$. However, as executing $a4$ arbitrarily often requires an unlimited number of copies of $f3$, the number of previous executions of the sequence $a3, a4$ determines how often $a4$ may be executed at most.

Consequently, in order to compute an actual plan, we have to

1. identify those action sequences on the path from the root to L that can be executed arbitrarily often, and
2. determine how often each repeatable action sequence has to be executed to fulfil the constraints imposed by subsequent repeatable action sequences and the goal state.

The first step can be realised simply by collecting some information during the execution of our partial deduction algorithm. In the following, let each B_i, for $i = 1, \ldots, m$, denote the action sequence which justifies the introduction of some variable, in the order of there appearance, of type t_{f_i} where f_i is a fluent.

For the second step we represent each action sequence described by the path from the root to L in the global tree as follows

$$A_1 B_1^{n_1} A_2 B_2^{n_2} \ldots A_m B_m^{n_m} A_{m+1}$$

Thereby A_i $(i = 1, \ldots, m+1)$ are action sequences such that $A_1, \ldots, A_{m+1}$ is the sequence of actions on the path from root to L and the end of each subsequence A_i $(i = 1, \ldots, m)$ is determined by the node where a new variable has been introduced.

Let A be some action sequence $a_1, \ldots, a_k$ and f some fluent. Then, we define[8]

$$|A, f| = \sum_{j=1}^{k}(|St^{+}_{a_i}, f| - |St^{-}_{a_i}, f|)$$

[8]This is equivalent to $\delta t(p)$ for the interleaving semantics in Petri nets, see Section 4.1.

$|A, f|$ simply denotes the change of the number of fluents f if A is executed.

Then, to fulfil the constraints described above, we solve a system of inequations, $i = 1, \ldots, m$:

$$n_i \geq \frac{|g_2, f_i| - |g_1, f_i| - \sum_{j=1}^{m+1}(\min(0, |A_j, f_i|) - \sum_{j=i}^{m} \min(0, |B_j, f_i|)n_j}{|B_i, f_i|}$$

Clearly, a set of $n_1, \ldots, n_m$ exists, since the system of inequalities has triangular form. Note that we may simplify the system further, e.g. by omitting the divisor $|B_i, f_i|$. However, the solutions may become bigger which might be undesirable.

Example 7.3.2 (Example 6.2.2 continued) *Consider again the domain of Example 6.2.2. Let $g_2 = f3^{13} \circ f2^9$. According to the tree of Figure 7.2 the conjunctive planning problem has several solutions, some of them are represented by the node L labelled $\langle Reachable_{14}(x_1 \circ x_2, v), T \wedge t_{f_3}(x_1) \wedge t_{f_2}(x_2), R\rangle$. The action sequences from the root node to L can be represented by*

$$A_1 B_1^{n_1} A_2 B_2^{n_2} A_3 = a1a3a4(a3a4)^{n_1} a3a4(a4)^{n_2}$$

Thereby, note that A_3 is an empty sequence. With

$$|A_1, f_3| = 1, |A_1, f_2| = 1, |A_2, f_3| = 1, |A_2, f_2| = 0,$$
$$|B_1, f_3| = 1, |B_1, f_2| = 0, |B_2, f_3| = -1, |B_2, f_2| = 1$$

we can derive the following system of inequations:

$$\begin{aligned} n_1 &\geq 13 + n_2 \\ n_2 &\geq 9 \end{aligned}$$

A solution is $n_2 = 9$, $n_1 = 22$, which results in the plan

$$a1a3a4(a3a4)^{22} a3a4(a4)^9$$

that leads to the state $f3^{15} \circ f2^{10}$, which contains the goal $f3^{13} \circ f2^9$. □

Example 7.3.3 (Airport domain continued) *Consider the airport domain of Example 2.5.4 and the conjunctive planning problem of Example 3.4.1. From the tree (not depicted due to its size) generated by our partial deduction algorithm we can find a successful action sequence from the root node $\langle Reachable(runway \circ bay^4, v), T, R\rangle$ (where v, T, R are defined as in the previous example but for all the fluents of the Airport domain) to some node $\langle Reachable(runway \circ bay^4 \circ x_1 \circ x_2, v), T \wedge t_{plane\_q\_b}(x_1) \wedge t_{passenger}(x_2), R\rangle$:*

$$A_1 B_1^{n_1} A_2 B_2^{n_2} A_3 = queue\_b^{n_1}(land\_b\ take\_off)^{n_2}$$

Thereby, note that A_1,A_2,A_3 are empty sequences. With

$$|B_1, \mathit{plane\_q\_b}| = 1, |B_1, \mathit{passenger}| = 0,$$
$$|B_2, \mathit{plane\_q\_b}| = -1, |B_2, \mathit{passenger}| = 5$$

we can derive the following system of inequations:

$$\begin{array}{lcl} n_1 & \geq & n_2 \\ n_2 & \geq & 22 \end{array}$$

A solution is $n_2 = 5$, $n_1 = 5$, which results in the plan

$$\mathit{queue\_b}^5 (\mathit{land\_b}\ \mathit{take\_off})^5$$

that leads to the state $\mathit{passenger}^{25} \circ \mathit{runway} \circ \mathit{bay}^4$, which contains the goal $\mathit{passenger}^{22}$. □

Conjunctive Planning Problems in the simple $\mathcal{FC}$

We may also apply our partial deduction Algorithm 7.1.2 to some simple $\mathcal{FC}$ domains. To encode conjunctive planning problems we use the scheme of Clause 6.18. However, there are conjunctive planning problems for simple $\mathcal{FC}$ domains using atomic propositions of L_Σ that can not be expressed in this way as g_1 of Clause 6.18 must be ground. We might relax this condition by allowing g_1 to contain variables which must be typed appropriately in T of the initial abstract conjunction $\langle \mathit{Reachable}(g_1, v), T, R\rangle$. But in general the set of formulas defined in this way is strictly weaker than Λ_{L_Σ} (as defined in Section 2.6). In the following we assume g_1 to be ground.

Similarly to $\mathcal{FC}_{PL}$ domains v in $\langle \mathit{Reachable}(g_1, v), T, R\rangle$ describes the set of all state terms, e.g. it might be a term of the form $x_1 \circ \ldots \circ x_n$ which contains precisely one variable x_i for each fluent function $f_i : \mathit{Obj}^i \to \mathit{Fl}$ of the domain. But, since we can not provide a complete method for simple $\mathcal{FC}$, we have to choose a structure of v which appears to be reasonable for the particular CPP (see Example 7.3.4). Accordingly, T is a type conjunction and R consists of appropriate RUL programs. Again, the global tree λ generated by our partial deduction algorithm will contain the solutions to several CPP's for the initial state g_1. To verify if there is a plan leading from g_1 to a particular state $g_2 \circ z$ we have to check for λ containing a node L such that $\langle \mathit{Reachable}(g_2, v), T, R\rangle \trianglelefteq_E \mathit{label}(L)$ (where v, T, R are defined as above). Note that in contrast to $\mathcal{FC}_{PL}$ domains, the term g_2 might contain variables as well. If there is such a node L, then the CPP has a solution.

Example 7.3.4 (Example 6.2.4 continued) *Additionally to the actions as defined in Example 6.2.4 and the domain independent clauses, let the initial state be defined as $St_{init} = f_1$. By Fl_{Σ_s} we denote the set of fluents in Σ_s.*

We define Δ_{Σ_s} to be the RULE domain $(\mathcal{AQ}, \gamma, \texttt{AC1})$ where every abstract conjunction $C \in \mathcal{AQ}$ is of the form $\langle Reachable(g, v), T, R\rangle$. Thereby $v = y_1 \circ \cdots \circ y_5$, g is a term of sort St and for all variables $x \in Vars(g)$ and $y \in Vars(v)$, $t_f(x) \in atoms(T)$ or $t_{foo}(x) \in atoms(T)$, and $t_{f'}(y) \in atoms(T)$ for some fluents $f, f' \in Fl_{\Sigma_s}$. R consists of (t_{fi}) of Example 7.1.2 for $i = 1, 4, 5$, and for $i = 2, 3$ and t_{foo}, respectively:

```
$t_{f_i}$(1°).                                   $t_{foo}$(0).
$t_{f_i}$($f_i(X)$)   :-  $t_{foo}$(X).             $t_{foo}$(s(X))   :-  $t_{foo}$(X).
$t_{f_i}$(Y ∘ X)   :-  $t_{foo}$(Y), $t_{f_i}$(X).
```

Assume that we are not interested in whether we can reach particular instances of t_{f2}. Instead we want to answer questions like: Is it possible to reach a state where at least four fluents $f_2(t_1)$, $f_2(t_2)$, $f_2(t_3)$, $f_2(t_4)$ exist, but $t_1, \ldots, t_4$ might represent different sub-terms? In this case we could specify the goal state simply as $g_2 = f_2(x_1) \circ f_2(x_2) \circ f_2(x_3) \circ f_2(x_4) \circ z$. By defining our type system accordingly, we may abstract away from the particular instance of $f_2(\_)$. To this end we define the type system as $\mathcal{T} = \{(t_f, R) \mid f \in Fl_{\Sigma_s}\} \cup \{(t_{foo}, R)\}$. Then, the tree in Figure 7.3 is generated by our partial deduction algorithm with input Σ_s, Δ_{Σ_s} and initial abstract conjunction $\langle Reachable(St_{init}, y_1 \circ \cdots \circ y_n), t_{f1}(y_1) \wedge \cdots \wedge t_{f5}(y_5), R\rangle$,

Again, to simplify the picture the RUL programs and some of the type information have not been represented. RUL programs do not change in this example and the type information has been depicted as follows: v represents $y_{f1} \circ \cdots \circ y_{f5}$ and the type conjunction T_j for each node labelled $\langle Reachable_j(u, v), T_j, R\rangle$ contains atoms $T = t_{f1}(y_1) \wedge \cdots \wedge t_{f5}(y_5)$. $t_{f1}, \ldots, t_{f5}$, t_{foo} are defined by the corresponding RUL programs (t_{f_i}) and (t_{foo}).

For example, we can conclude from the tree that it is indeed possible to generate states containing four, and in fact arbitrary many, instances of the fluent $f2$ (for example by the node labelled 13). But we cannot conclude whether we can generate a state containing arbitrary many copies of one particular instance of $f2$. □

What we get for "free"

Since the partial deduction method proposed in this chapter is a program transformation technique the result is a new program. Like the old program, this new program represents a sound and complete forward interpreter for the original domain description. It is specialised for the chosen particular initial state (or set of initial states) characterised by the initial abstract conjunction. Hence, once we have computed the global tree λ for some domain and some initial state then, on one hand, we have an accelerated interpreter, and on the other hand, the solutions for any CPP (some CPP's, in case of the simple $\mathcal{FC}$) can easily be extracted from λ.

$$\begin{tikzcd}
 & & \langle \mathit{Reachable}_1(f1,v),T,R'\rangle \arrow[dl, "a1"'] \arrow[drrr, "a2"] & & & \\
 & \langle \mathit{Reachable}_2(f2(0),v),T,R'\rangle \arrow[d, "a3(0)"'] & & & \langle \mathit{Reachable}_3(f4,v),T,R'\rangle \arrow[d, "a5"] & \\
 & \langle \mathit{Reachable}_{10}(f3(foo(0))\circ f3(foo(foo(0))),v),T,R'\rangle \arrow[d, "a4(foo(0))"'] & & & \langle \mathit{Reachable}_5(f5\circ f5,v),T,R'\rangle \arrow[d, "a6"] & \\
 & \langle \mathit{Reachable}_{11}(f2(x_1)\circ x_2,v),T\wedge t_{foo}(x_1)\wedge t_{f3}(x_2),R'\rangle \arrow[dl, "a3(y)"'] \arrow[dr, "a4(y)"] & & & \langle \mathit{Reachable}_6(f4\circ x_1,v),T\wedge t_{f5}(x_1),R'\rangle \arrow[dl, "a6"'] \arrow[dr, "a5"] & \\
\langle \mathit{Reachable}_{12}(x_2,v),T\wedge t_{f2}(x_2),R'\rangle \arrow[d, "a4(z)"'] & & \textcircled{13} \arrow[loop below, "{a3(y),a4(z)}"] & \textcircled{8} \arrow[loop below, "{a5,a6}"] & & \langle \mathit{Reachable}_7(x_1,v),T\wedge t_{f5}(x_1),R'\rangle \arrow[d, "a6"] \\
\langle \mathit{Reachable}_{14}(x_3\circ x_2,v),T\wedge t_{f2}(x_3)\wedge t_{f3}(x_2),R'\rangle \arrow[loop below, "{a3(y),a4(z)}"] & & & & & \langle \mathit{Reachable}_9(x_{f4}\circ x_{f5},v),T\wedge t_{f5}(x_1)\wedge t_{f4}(x_2),R'\rangle \arrow[loop below, "{a5,a6}"]
\end{tikzcd}$$

$$\langle \mathit{Reachable}_{13}(x_3\circ x_2,v),T\wedge t_{f3}(x_2)\wedge t_{f2}(x_3),R'\rangle = \textcircled{13} \qquad \textcircled{8} = \langle \mathit{Reachable}_8(x_2\circ x_1,v),T\wedge t_{f5}(x_1)\wedge t_{f4}(x_2),R'\rangle$$

Figure 7.3: Example 7.3.4.

Conclusion

The main goal of this thesis was the investigation of the possibilities for automatic reasoning in the Fluent Calculus. To achieve this goal we pursue a systematic approach for the analysis of reasoning about Fluent Calculus domains. The approach relates reasoning about Fluent Calculus domains to model checking of dynamic systems. To this end we have distinguished between the representation of knowledge describing a dynamic system and the representation of knowledge describing dynamic properties we wish to infer. The former knowledge is represented using a scheme of extended first-order languages ($\mathcal{FC}$ domains). The latter knowledge is represented using a scheme of monadic second order languages over trees (query logic $\mathcal{QL}$). Both schemes are connected by common semantics, namely Herbrand-$\mathrm{E}_{\mathcal{FC}}$-models. We have defined several fragments of $\mathcal{FC}$ domains by their syntactic properties and their relation to Fluent Calculus fragments proposed in the literature. We have focused on deterministic domains. The most general $\mathcal{FC}$ fragment we have considered is the non-propositional fragment $\mathcal{FC}_{LN}$ which allows the use of negation in precondition axioms. The fragment of $\mathcal{FC}_{LN}$ where negation in precondition axioms is disallowed has been defined as $\mathcal{FC}_L$. The propositional versions of $\mathcal{FC}_{LN}$ and $\mathcal{FC}_L$ have been denoted as $\mathcal{FC}_{PLN}$ and $\mathcal{FC}_{PL}$, respectively. By restricting the state terms to contain only single copies of fluents we have defined $\mathcal{FC}_P$ as a fragment of $\mathcal{FC}_{PLN}$.

As we have shown, Herbrand-$\mathrm{E}_{\mathcal{FC}}$-models can be characterised by a class of transition systems where transitions as well as states carry labels. Most dynamic properties of interest in reasoning about dynamic systems are invariant for bisimilar transition systems. Hence, we have proposed to compare the models of Fluent Calculus domains by means of bisimulation to investigate their expressive power. Applying this idea we have proven that $\mathcal{FC}_{LN}$ domains are more expressive than $\mathcal{FC}_L$ domains equipped with a specificity relation ($\mathcal{FC}_{L\leq}$ domains). We have also shown that if we are interested only in the solutions of planning problems many non-deterministic domain descriptions can be transformed into deterministic ones.

In the second part of this work we have demonstrated that the characterisation of Herbrand-$\mathrm{E}_{\mathcal{FC}}$-models by transition systems and the notion of bisimulation can also be used, on one hand, to establish relations between $\mathcal{FC}$ domains and well-known computational models, and on the other hand, to characterise formulas of the query logic by formulas of modal/temporal logics. Although we believe these relations are already interesting themselves we have applied them to show several new results concerning the possibilities of automatic reasoning about Fluent Calculus domains:

1. By using the equivalence of the propositional μ-calculus and the bisimulation invariant fragment of $\mathcal{QL}$ and the bisimulation equivalence between models of $\mathcal{FC}_P$ domains and 1-safe Petri nets we have shown that the entailment of formulas of the propositional μ-calculus is decidable for $\mathcal{FC}_P$ domains.
2. By characterising a first-order fragment of $\mathcal{QL}$ by the temporal logic CTL_U and by applying the bisimulation equivalence between models of $\mathcal{FC}_{PL}$ domains and Petri nets we have proven that the entailment of first-order formulas for all $\mathcal{FC}$ fragments except $\mathcal{FC}_P$ is undecidable. We have published this result in [88].
3. Again, by applying the bisimulation equivalence between models of $\mathcal{FC}_{PL}$ domains and Petri nets we have shown that both the planning problem and the extended planning problem (which can be expressed in CTL_U) are decidable. The result on the decidability of the planning problem has also been published in [88].
4. We have shown that the halting problem for any deterministic two-counter machine can be expressed as planning problems for a $\mathcal{FC}_{PLN}$ domain, a $\mathcal{FC}_L$ domain and a $\mathcal{FC}_{PL\leq}$ domain (the propositional version of $\mathcal{FC}_{L\leq}$), respectively. For all three fragments follow as consequences that the planning problem, the extended planning problem and the conjunctive planning problem are undecidable.

In the third part we have developed a method for solving conjunctive planning problems in logic programming representations of $\mathcal{FC}_L$ and $\mathcal{FC}_{PL}$ domains. The method is based on an approach for checking coverability of Petri nets by partial deduction which we have published in [94]. To apply partial deduction on $\mathcal{FC}$ domains we have extended the approach to handle an equational theory and to incorporate regular type approximation. By proving that our algorithm is complete for $\mathcal{FC}_{PL}$ domains, we have provided the first complete automatic reasoning method that solves conjunctive planning problems in $\mathcal{FC}_{PL}$. Of course, as we have shown, to solve conjunctive planning problems in $\mathcal{FC}_{PL}$ it is also possible to apply the Karp-Miller procedure to $\mathcal{FC}_{PL}$ domains translated into Petri nets. However, in contrast to the Karp-Miller procedure, our partial deduction method scales up to $\mathcal{FC}_L$ domains. Although it is impossible to find a general method to decide conjunctive planning problems in $\mathcal{FC}_L$ domains, partial deduction can produce useful results. As a side-effect our partial deduction method can also be applied to automatically generate optimised interpreters for

$\mathcal{FC}_{PL}$ and $\mathcal{FC}_L$ domains which are specialised for particular (sets of) initial situations. We have published these results in [89].

We hope that our results, based on the analysis of the computational properties implied by the structure of the represented knowledge in Fluent Calculus domains, may indeed help to develop effective highly specialised automatic algorithms for important problems of common sense reasoning.

Further Work

There are several areas in which this work can be continued. The Fluent Calculus has been extended to allow the representation of many more aspects of action and change than the ones investigated here. The extensions by indirect effects [149] and actions that cause continuous change [151] seem to be particularly interesting from an automatic reasoning point of view. Furthermore, there are many dynamic properties which are of interest in common sense reasoning (some of them have been mentioned in Section 2.1) while our work has been focused only on (extended/conjunctive) planning problems. If we wish to infer dynamic properties of continuous systems (e.g., if actions that cause continuous change are considered) it might also be necessary to extend the query logic.

Considering the similarities between propositional Fluent Calculus fragments and Petri nets it is promising to analyse the relation between non-propositional Fluent Calculus fragments and more powerful net models, e.g. predicate transition systems and coloured Petri nets. On the other hand, it might be interesting to investigate whether Fluent Calculus fragments defined by their relation to certain computational models can be used for common sense reasoning. For example, many dynamic properties like the planning problem are decidable for pushdown automata. The corresponding decision procedures could be easily adapted for an appropriately defined Fluent Calculus fragment.

Our partial deduction method of the last part can be extended as well. In [93] we have shown that partial deduction can be used to decide coverability for more general systems, e.g. Petri nets with reset arcs and well-structured transition systems. Hence, we might be able to improve our method to handle more complicated Fluent Calculus domains. Additionally, we have shown in [94] that partial deduction cannot only be used to calculate the Karp-Miller tree but also to generate the so-called *minimal coverability graph.* This graph contains similar information as the Karp-Miller tree but is often considerably smaller. For practical applications an adaption of our method of Part III to generate the minimal coverability graph could be very important.

In Section 3.4 we showed that certain formulas of the first-order logic over trees can be characterised by the temporal logic CTL_U. It would be interesting to find out whether the whole bisimulation invariant fragment of this first-order logic can be characterised by a simple temporal logic like CTL_U.

Bibliography

[1] P. A. Abdulla, K. Čerāns, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *11th IEEE Symposium on Logic in Computer Science*, pages 313–321, 1996.

[2] K. Abrahamson. Modal logic of concurrent nondeterministic programs. In *International Symposium on Semantics of Concurrent Computation*, Evian-les-Baines, July 1979.

[3] M. Alpuente, M. Falaschi, and G. Vidal. Partial evaluation of functional logic programs. *ACM Transactions on Programming Languages and Systems*, 20(4):768–844, July 1998.

[4] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, Cambridge, UK, 1998.

[5] F. Baader and J. H. Siekmann. Unification theory. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming.* Oxford University Press, Oxford, UK, 1993.

[6] A. B. Baker. A Simple Solution to the Yale Shooting Problem. In Hector J. Levesque Ronald J. Brachman and Raymond Reiter, editors, *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 11–20, Toronto, Canada, May 1989. Morgan Kaufmann.

[7] K. van Belleghem, M. Denecker, and D. de Schreye. On the relation between situation calculus and event calculus. *Journal of Logic Programming*, 31(1–3):3–37, April–June 1997.

[8] J. van Benthem. Correspondence theory. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Volume II: Extensions of Classical Logic*, volume 165 of *Synthese Library*, chapter II.4, pages 167–247. D. Reidel Publishing Co., Dordrecht, 1984.

[9] J. van Benthem. Temporal logic. In Dov Gabbay, C. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic*

Programming, Volume 4: Epistemic and Temporal Reasoning. Oxford University Press, Oxford, 1995.

[10] L. Bernardinello and F. De Cindio. A survey of basic net models and modular net classes. *Lecture Notes in Computer Science*, 609, 1992.

[11] W. Bibel. A deductive solution for plan generation. *New Generation Computing*, 4:115–132, 1986.

[12] W. Bibel. *Automated Reasoning: Essays in Honor of Woody Bledsoe*, chapter Perspectives on Automated Deduction, pages 77–104. Kluwer Academic, Utrecht, 1991.

[13] W. Bibel. Let's plan it deductively! *Artificial Intelligence*, 103(1–2):183–208, 1998.

[14] W. Bibel, S. Hölldobler, and T. Schaub. *Wissensrepräsentation und Inferenz: eine grundlegende Einführung.* Vieweg, Wiesbaden, 1993.

[15] S.-E. Bornscheuer and H. Lehmann. On the Combination of Partial Action Descriptions. In *Proceedings of the 11th Australian Joint Conference on Artificial Intelligence '98*, volume LNAI 1502. Springer, 1998.

[16] S.-E. Bornscheuer and M. Thielscher. Explicit and implicit indeterminism: Reasoning about uncertain and contradictory specifications of dynamic systems. *Journal of Logic Programming*, 31(1–3):119–155, April–June 1997.

[17] H.-J. Bürckert, A. Herold, D. Kaput, J. H. Siekmann, M. E. Stickel, M. Tepp, and H. Zhang. Opening the AC-unification race. *Journal of Automated Reasoning*, 4(4):465–474, December 1988.

[18] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1985.

[19] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum, New York, NY, 1978.

[20] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. In *Conference Record of the Tenth Annual ACM Symposium on Principles of Programming Languages*, pages 117–126. ACM, ACM, January 1983.

[21] E. M. Clarke and J. M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, December 1996.

[22] A. G. Cohn. A more expressive formulation of many sorted logic. *Journal of Automated Reasoning*, 3(2):113–200, June 1987.

[23] D. De Schreye, R. Glück, J. Jørgensen, M. Leuschel, B. Martens, and M. H. Sørensen. Conjunctive partial deduction: Foundations, control, algorithms and experiments. *Journal of Logic Programming*, 41(2 & 3):231–277, November 1999.

[24] D. A. de Waal and J. P. Gallagher. The applicability of logic program analysis and transformation to theorem proving. In Alan Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, volume 814 of *LNAI*, pages 207–221, Berlin, June/July 1994. Springer.

[25] D. A. de Waal and M. Thielscher. Solving deductive planning problems using program analysis and transformation. In M. Proietti, editor, *Proceedings of the International Workshop on Logic Program Synthesis and Transformation (LOPSTR)*, volume 1048 of *LNCS*, pages 189–203. Springer, September 1995.

[26] G. Delzanno and A. Podelski. Model checking in CLP. In R. Cleaveland, editor, *Proceedings of TACAS'99*, volume 1579 of *LNCS*, pages 223–239. Springer, 1999.

[27] H. Dreyfus. *What Computers Can't Do: A Critique of Artificial Reason.* Harper & Row, New York, 1972.

[28] H. Ehrig, B. Mahr, F. Cornelius, M. Grosse-Rhode, and P. Zeitz. *Mathematisch-strukturelle Grundlagen der Informatik.* Springer, 1999.

[29] S. Eilenberg and M.P. Schutzenberger. Rational sets in commutative monoids. *Journal of Algebra*, 13(2):173–191, 1969.

[30] E. A. Emerson. *Model checking and the mu-calculus*, volume 31 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, chapter 6. American Mathematical Society, 1997.

[31] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181, Noordweijkerhout, The Netherland, 14–18 July 1980. Springer-Verlag.

[32] E. A. Emerson and K. S. Namjoshi. On model checking for nondeterministic infinite state systems. In *13th IEEE Symposium on Logic in Computer Science*, pages 70–80, 1998.

[33] E. A. Emerson and A. Prasad Sistla. Deciding full branching time logic. *Information and Control*, 61(3):175–201, June 1984.

[34] J. Esparza. On the decidability of model checking for several mu-calculi and petri nets. In *CAAP: Colloquium on Trees in Algebra and Programming.* LNCS 787, Springer-Verlag, 1994.

[35] J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34(2):85–107, 1997.

[36] J. Esparza and S. Melzer. Model checking LTL using constraint programming. *Lecture Notes in Computer Science*, 1248, 1997.

[37] C. Evans. Negation-as-failure as an approach to the hanks and mcdermott problem. In *Proc. of the 2nd International Symposium on Artificial Intelligence*, 1989.

[38] R. E. Fikes and H. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:pages 189–205, 1971.

[39] A. Finkel. The minimal coverability graph for Petri nets. *Lecture Notes in Computer Science*, 674:210–243, 1993.

[40] A. Finkel and P. Schnoebelen. Fundamental structures in well-structured infinite transition systems. In *Proceedings of LATIN'98*, LNCS 1380, pages 102–118. Springer-Verlag, 1998.

[41] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 1999. To appear.

[42] M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer, New York, 1990.

[43] G. Frege. Über Sinn und Bedeutung. *Zeitschrift fuer Philosophie und Philosophische Kritik*, (100):25–50, 1892.

[44] A. M. Frisch. The substitutional framework for sorted deduction: Fundamental results on hybrid reasoning. *Artificial Intelligence*, 49(1–3):161–198, 1991.

[45] J. P. Gallagher and J. C. Peralta. Using regular approximations for generalisation during partial evaluation. In *Proceedings of the 2000 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'2000), Boston, Mass., (ed. J. Lawall)*, pages 44–51. ACM Press, January 2000.

[46] J.P. Gallagher and D. A. de Waal. Fast and precise regular approximations of logic programs. In Pascal Van Hentenryck, editor, *Proceedings of the Eleventh International Conference on Logic Programming*, pages 599–613. The MIT Press, 1994.

[47] J. H. Gallier and S. Raatz. Extending SLD resolution to equational horn clauses using E-unification. *Journal of Logic Programming*, 6(1-2):3–43, January 1989.

[48] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4):365–385, 1991.

[49] M. L. Ginsberg and D. E. Smith. Reasoning about action I: A possible worlds approach. *Artificial Intelligence*, 35:165–195, 1988.

[50] S. Ginsburg and E.H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.

[51] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50, 1987.

[52] R. Glück and M. Leuschel. Abstraction-based partial deduction for solving inverse problems – a transformational approach to software verification. In *Proceedings of the Third International Ershov Conference on Perspectives of System Informatics*, LNCS 1755, pages 93–100, Novosibirsk, Russia, 1999. Springer-Verlag.

[53] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. i. *Monatsh. Math. Phys.*, (38):173–198, 1931.

[54] G. Görz, editor. *Einführung in die künstliche Intelligenz.* Addison-Wesley, Bonn, 1995.

[55] C. Green. Application of theorem proving to problem solving. In *International Joint Conference on Artificial Intelligence*, pages 219–239, Los Altos, CA, 1969. Morgan Kaufmann Publishers.

[56] M. Hack. The recursive equivalence of the reachability problem and the liveness problem for petri nets and vector addition systems. In *Proceedings of 15th Annual IEEE Symposium on Switching and Automata Theory*, 1974.

[57] T. Hafer and W. Thomas. Computation tree logic CTL* and path quantifiers in the monadic theory of the binary tree. In Thomas Ottmann, editor, *Automata, Languages and Programming, 14th International Colloquium*, volume 267 of *Lecture Notes in Computer Science*, pages 269–279, Karlsruhe, Germany, 13–17 July 1987. Springer-Verlag.

[58] S. Hanks and D. McDermott. Default reasoning, nonmonotonic logic and the frame problem. In *National Conference on Artificial Intelligence of the American Association for AI (AAAI 86)*, pages 328–333, 1986.

[59] S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[60] M. Hanus. The integration of functions into logic programming. *Journal of Logic Programming*, 19 & 20:583–628, May 1994.

[61] D. Hauschildt. Semilinearity of the Reachability Set is Decidable for Petri nets. Technical Report FbI-HH-B-146/91, University of Hamburg, Department of Computer Science, Hamburg, 1990.

[62] P. J. Hayes. The second naive physics manifesto. In *Formal Theories of the Commonsense World*, pages 1–36. Ablex Publishing, 1985.

[63] R. Hecht-Nielsen. *Neurocomputing.* Addison-Wesley, Reading, 1990.

[64] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Jrnl. A.C.M.*, 23(1):137–161, January 1985.

[65] P. M. Hill and J. Gallagher. Meta-programming in logic progamming. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 421–498. Oxford University Press, January 1998.

[66] S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence.* Springer, 1989.

[67] S. Hölldobler and D. Kuske. The boudary between decidable and undecidable fragments of the fluent calculus. In *Proceedings of the International Conference on Logic Programming and Automated Reasoning (LPAR'2000)*, LNAI 1861. Springer-Verlag, 2000.

[68] S. Hölldobler and J. Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990.

[69] S. Hölldobler and H.-P. Störr. Solving the entailment problem in the fluent calculus with binary decision diagrams. In J. Lloyd, editor, *Proceedings of the International Conference on Computational Logic (CL'2000)*, LNCS 1861, London, UK, 2000. Springer-Verlag.

[70] S. Hölldobler and M. Thielscher. Computing change and specificity with equational logic programs. *Annals of Mathematics and Artificial Intelligence*, 14:99–133, 1995.

[71] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[72] J. Jaffar, J.-L. Lassez, and M. Maher. A Theory of Complete Logic Programs. *Journal of Logic Programming*, 1(3), 1984.

[73] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In U. Montanari and V. Sassone, editors, *CONCUR '96: Concurrency Theory, 7th International Conference*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277, Pisa, Italy, 26–29 August 1996. Springer-Verlag.

[74] Y. Kalinke and H. Lehmann. Computations in Recurrent Neural Networks: From Counters to Iterated Function Systems. In *Proceedings of the Australian Annual Conference on Artificial Intelligence '98*, LNAI 1502. Springer, 1998.

[75] S. Kambhampati. Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18(2), 1997.

[76] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3:147–195, 1969.

[77] B. Kosko. *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence.* Prentice-Hall, London, 1992.

[78] R. Kowalski. *Logic for Problem Solving.* The Computer Science Library, Artificial Intelligence Series. North Holland, New York, 1979.

[79] R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[80] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, (27):333–354, 1983.

[81] S. Kripke. Semantical analysis of modal logic I. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[82] Saul Kripke. Semantic considerations on modal logic. *Acta Philosophica Fennica*, 24:83–94, 1963.

[83] Saul A. Kripke. A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24(1):1–14, 1959.

[84] J. Lambek. The mathematics of sentence structure. *Amer. Math. Monthly*, 65(3):154–170, 1958.

[85] J.L. Lambert. Vector addition systems and semi–linearity. Technical report, Universite de Paris–Nord, centre Scientifique et Polytechnique, 1994.

[86] L. Lamport. "Sometimes" is sometimes "not ever": On the temporal logic of programs. In *Conference Record of the Seventh annual ACM Symposium on Principles of Programming Languages*, pages 174–185. ACM, ACM, January 1980.

[87] J.-L. Lassez, M.J. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan-Kaufmann, 1988.

[88] H. Lehmann and M. Leuschel. Decidability results for the propositional fluent calculus. In J. Lloyd, editor, *Proceedings of the International Conference on Computational Logic (CL'2000)*, LNCS 1861, London, UK, 2000. Springer-Verlag.

[89] H. Lehmann and M. Leuschel. Solving Planning Problems by Partial Deduction. In *Proceedings of the International Conference on Logic Programming and Automated Reasoning (LPAR'2000)*, LNCS 1955. Springer-Verlag, 2000.

[90] M. Leuschel. Program specialisation and abstract interpretation reconciled. In Joxan Jaffar, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming (JICSLP'98)*, Manchester, UK, June 1998. MIT Press.

[91] M. Leuschel. Logic program specialisation. In J. Hatcliff, Torben Æ. Mogensen, and Peter Thiemann, editors, *Partial Evaluation: Practice and Theory*, LNCS 1706, pages 155–188, Copenhagen, Denmark, 1999. Springer-Verlag.

[92] M. Leuschel and D. De Schreye. Logic program specialisation: How to be more specific. In H. Kuchen and S.D. Swierstra, editors, *Proceedings of the International Symposium on Programming Languages, Implementations, Logics and Programs (PLILP'96)*, LNCS 1140, pages 137–151, Aachen, Germany, September 1996. Springer-Verlag.

[93] M. Leuschel and H. Lehmann. Coverability of Reset Petri Nets and other Well-Structured Transition Systems by Partial Deduction. In J. Lloyd, editor, *Proceedings of the International Conference on Computational Logic (CL'2000)*, LNCS 1861, London, UK, 2000. Springer-Verlag.

[94] M. Leuschel and H. Lehmann. Solving Coverability Problems of Petri Nets by Partial Deduction. In Maurizio Gabbrielli and Frank Pfenning, editors, *Proceedings of PPDP'2000*, pages 268–279, Montreal, Canada, 2000. ACM Press.

[95] M. Leuschel, B. Martens, and D. De Schreye. Controlling generalisation and polyvariance in partial deduction of normal logic programs. *ACM Transactions on Programming Languages and Systems*, 20(1):208–258, January 1998.

[96] M. Leuschel and T. Massart. Infinite state model checking by abstract interpretation and program specialisation. In Annalisa Bossi, editor, Logic-Based Program Synthesis and Transformation. *Proceedings of LOPSTR'99*, LNCS 1817, pages 63–82, Venice, Italy, September 1999.

[97] C. I. Lewis. *A Survey of Symbolic Logic.* Univ. of California Press, Berkeley, Berkeley, 1918. Reprint of Chapters I–IV by Dover Publications, 1960, New York.

[98] V. Lifschitz. Formal theories of action. In Frank M. Brown, editor, *Proceedings of the 1987 Workshop on the Frame Problem*, pages 35–57, Los Altos, California, 1987. Morgan Kaufmann.

[99] F. Lin and Y. Shoham. Provably correct theories of action. In Kathleen Dean, Thomas L.; McKeown, editor, *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 349–354. MIT Press, July 1991.

[100] M. Livesey and J. Siekmann. Unification of bags and sets. Technical report, Institut für Informatik I, Universität Karlsruhe, 1976.

[101] J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *Journal of Logic Programming*, 11(3&4):217–242, October 1991.

[102] R. Marlet. *Vers une Formalisation de l'Évaluation Partielle.* PhD thesis, Université de Nice - Sophia Antipolis, December 1994.

[103] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984.

[104] J. McCarthy. Situations, actions and causal laws. AI-Memo 1, Stanford University, Artificial Intelligence Project, Stanford, CA, 1957.

[105] J. McCarthy. Situations, actions, and causal laws. Memo 2, Stanford University Artificial Intelligence Project, Stanford, California, 1963.

[106] J. McCarthy. Epistemological problems of artificial intelligence. *Proceedings IJCAI-77, Cambridge MA,1977. Also in "Readings in knowledge representation", ed. Ronald J. Brachman and Hector J. Levesque, Morgan Kaufman, 1985*, pages 1038–1044, 1977.

[107] J. McCarthy. First order theories of individual concepts and propositions. In J. E. Hayes, D. Mitchie, and L. I. Mikulich, editors, *Machine Intelligence 9*, pages 129–148. Ellis Horwood, Chichester, England, 1979.

[108] J. McCarthy. Applications of circumscription to formalizing common sense. *Artificial Intelligence*, 13:27–39, 1986.

[109] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463 - 502. Edinburgh University Press, 1969.

[110] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice Hall, London, 1967.

[111] F. Moller. Infinite results. In *Proceedings of CONCUR'96*, LNCS 1119, pages 195–216. Springer-Verlag, 1996.

[112] F. Moller and Rabinovich. On the expressive power of CTL. In *LICS: IEEE Symposium on Logic in Computer Science*, 1999.

[113] A. Newell. The knowledge level. *AI Magazine*, 2(2):1–20, 1981.

[114] D. Park. Fixpoint induction and proofs of program properties. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 59–78. Edinburgh University Press, 1969.

[115] R. Penrose. *The Emperor's New Mind.* Oxford University Press, Oxford, 1989.

[116] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Schriften des Institutes für Instrumentelle Mathematik, Bonn, 1962.

[117] J. Pinto and R. Reiter. Temporal reasoning in logic programming: A case for the situation calculus. In D. S. Warren, editor, *Proceedings of the Tenth International Conference on Logic Programming*, pages 203–221, Budapest, Hungary, 1993. The MIT Press.

[118] J. A. Pinto. *Temporal Reasoning in the Situation Calculus.* PhD thesis, Computer Science, University of Toronto, 1994.

[119] F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 46(3):325–361, May 1999.

[120] G. D. Plotkin. A note on inductive generalisation. *Machine Intelligence*, 5:153–163, 1970.

[121] G. D. Plotkin. Building in equational theories. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, number 7, pages 73–90, Edinburgh, Scotland, 1972. Edinburgh University Press.

[122] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pages 46–57, Providence, Rhode Island, 1977. IEEE, IEEE Computer Society Press.

[123] V. R. Pratt. Semantical considerations on floyd-hoare logic. In *17th Annual Symposium on Foundations of Computer Science*, pages 109–121. IEEE, October 1976.

[124] A. N. Prior. *Time and Modality.* Oxford University Press, Oxford, UK, 1957.

[125] A. N. Prior. *Past, Present and Future.* Oxford University Press, Oxford, 1967.

[126] M. O. Rabin. Decidability of second-order theories and automata on infitite trees. *Trans. of Amer. Math. Soc.*, 141:1–35, 1969.

[127] R. Reiter. *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, chapter The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. Academic Press, 1991.

[128] R. Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence*, 64(2):337–351, December 1993.

[129] G. Restall. *An Introduction to Substructural Logics.* Routledge, London, 2000.

[130] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[131] J. A. Robinson. A review of automatic theorem-proving. In *Proc. Annual Symposium in Applied Mathematics XIX*, pages 1–18, American Mathematical Society, Providence, 1967.

[132] G. Roth. *Das Gehirn und seine Wirklichkeit: kognitive Neurobiologie und ihre philosophischen Konsequenzen.* Suhrkamp, Frankfurt, 1994.

[133] K. Sagonas, T. Swift, and D. S. Warren. XSB as an efficient deductive database engine. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 442–453, Minneapolis, Minnesota, May 1994. ACM.

[134] P. Schroeder-Heister and K. Dosen. *Substructural Logic.* Clarendon Press, Oxford, 1993.

[135] H. Seiffert and G. Radnitzky, editors. *Handlexikon zur Wissenschaftstheorie.* Ehrenwirth Verlag, 1989.

[136] L. Shastri and V. Ajjanagadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, 16:417–494, 1993.

[137] J. C. Shepherdson. SLDNF-Resolution with Equality. *Journal of Automated Reasoning*, 8:297–306, 1992.

[138] H. Siegelmann and E. Sontag. Turing computability with neural nets. *Appl. Math. Lett.*, 4(6), 1991.

[139] J. H. Siekmann. Unification theory. *Journal of Symbolic Computation*, 7(3–4):207–274, March–April 1989.

[140] M. H. Sørensen and R. Glück. An algorithm of generalization in positive supercompilation. In J. W. Lloyd, editor, *Proceedings of ILPS'95, the International Logic Programming Symposium*, pages 465–479, Portland, USA, December 1995. MIT Press.

[141] C. Stirling. Modal and temporal logics. In D. Gabbay S. Abramsky and F. Guenther, editors, *Handbook of Logic in Computer Science 2*, pages 477–563. Oxford University Press, 1992.

[142] H.-P. Störr and M. Thielscher. A new equational foundation for the fluent calculus. In J. Lloyd, editor, *Proceedings of the International Conference on Computational Logic (CL'2000)*, LNCS 1861, London, UK, 2000. Springer-Verlag.

[143] E. Ternovskaia. Interval situation calculus. In B. Nebel and L. Dreschler-Fischer, editors, *Proceedings of the 18th German Annual Conference on Artificial Intelligence : KI-94: Advances in Artificial Intelelligence*, volume 861 of *LNAI*, Berlin, September 1994. Springer.

[144] E. Ternovskaia. Inductive Definability and the Situation Calculus. In B. Freitag, H. Decker, M. Kifer, and A. Voronkov, editors, *Transactions and Change in Logic Databases*, volume 1472 of *LNCS*. Springer, 1998.

[145] E. Ternovskaia. Automata theory for reasoning about actions. In T. Dean, editor, *Proceedings of IJCAI*. Morgan Kaufmann, August 1999.

[146] M. Thielscher. *Automatisiertes Schliessen über Kausalbeziehungen mit SLDENF-Resolution*. PhD thesis, Darmstadt University of Technology, 1994.

[147] M. Thielscher. Computing ramifications by postprocessing. In Chris Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1994–2000, San Francisco, 1995. Morgan Kaufmann.

[148] M. Thielscher. Causality and the qualification problem. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 51–62. Morgan Kaufmann, San Francisco, California, 1996.

[149] M. Thielscher. Ramification and causality. *Artificial Intelligence*, 89(1–2):317–364, 1998.

[150] M. Thielscher. From Situation Calculus to Fluent Calculus: State update axioms as a solution to the inferential frame problem. *AIJ*, 111(1–2):277–299, 1999.

[151] M. Thielscher. Modeling actions with ramifications in nondeterministic, concurrent, and continuous domains—and a case study. In H. Kautz and B. Porter, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI)*, pages 497–502, Austin, TX, July 2000. MIT Press.

[152] M. Thielscher. Representing the knowledge of a robot. In A. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 109–120, Breckenridge, CO, April 2000. Morgan Kaufmann.

[153] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236), 1950.

[154] C. Weidenbach. Extending the resolution method with sorts. In R. Bajcsy, editor, *Proceedings of IJCAI*, Chambéry, France, August 1993. Morgan Kaufmann.

[155] J. Weizenbaum. *Computer Power and Human Reason*. Freeman, San Francisco, 1976.

[156] D. S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.

[157] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *Proceedings of CAV'98*, LNCS, pages 88–97. Springer-Verlag, 1998.

[158] E. Yardeni and E. Shapiro. A type system for logic programs. *Journal of Logic Programming*, 10(2):125–153, February 1991.

Index