REASONING ABOUT ACTIONS:
A MODEL-THEORETIC APPROACH

by

Mayu Ishida

B.Sc. Honours, Simon Fraser University, 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Fall 2006

APPROVAL

Name:	Mayu Ishida
Degree:	Master of Science
Title of thesis:	Reasoning about Actions: A Model-Theoretic Approach
Examining Committee:	Dr. Jim Delgrande Chair
	Dr. Evgenia Ternovska Senior Supervisor
	Dr. David G. Mitchell Supervisor
	Dr. Arvind Gupta SFU Examiner
Date Approved:	August 2, 2006

Abstract

A knowledge-based agent reasons with its knowledge and answers queries while performing various tasks. We consider the case where we describe the agent's knowledge in a propositional fragment of the situation calculus and queries in a fragment of ID-logic, the extension of first-order logic with inductive definitions. This fragment of ID-logic is equivalently as expressive as the alternation-free μ-calculus. We formulate the agent's reasoning process as the following question: does the representation T of the agent's knowledge logically entail the query ϕ (i.e., $T \models \phi$)? We provide an efficient algorithm for this task, using a model-theoretic approach: we construct from T a canonical model $\mathfrak{F}^T$ of the agent's knowledge and ask whether $\mathfrak{F}^T$ satisfies ϕ. Using this approach, the agent can answer the query in time linear with respect to both the size of T and the size of ϕ.

Acknowledgements

I would like to thank Dr. Eugenia Ternovska for giving me financial support and supervision, and my past and present colleagues in the Computational Logic Lab for giving me their fellowship, especially, Eric Evangelista, Dr. Yongmei Liu, and Dr. Antonina Kolokolova for proof-reading my thesis. Moreover, I would like to thank Dr. Omar Elesseily and Catalina Ip, my chiropractor and massage therapist, respectively, for treating my neck, shoulders, and back, so that I can continue to work on a computer. At last, but not least, I would like to thank my family and friends for lending me their ears, kicking me in the butt, and above all, encouraging me to go on and finish my Master's degree.

Contents

Chapter 1

Introduction

Artificial Intelligence (AI) is the study of intelligent behaviour [42]. Its purpose is to understand the principles that make intelligent behaviour possible in natural and artificial environments. AI researchers, based on their hypotheses about the principles, construct *agents*, entities that can perceive, query, and act in a given environment. Moreover, AI researchers investigate and redesign their agents so that the agents do not simply mimic humans but are capable of intelligent behaviour on their own. In other words, AI researchers do not only observe the external behaviour of intelligent entities, but also examine and improve executable models of intelligent behaviour.

Symbolic logic is an important mathematical apparatus for AI, for it allows AI researchers to express and unify their ideas in a precise way [17]. Moreover, a solid grounding in symbolic logic provides a framework for interpreting, understanding, and building the discipline of AI. For example, if we want to develop systems that use and manipulate *declarative knowledge*, a kind of knowledge that we describe in a language of symbolic logic, we should take into account the prior results on proof theory and model theory.

An intelligent agent needs knowledge about its environment so that it can make good decisions. A *knowledge-based agent* formalizes its knowledge about the environment into sentences of a *representation language*, a language of symbolic logic to declare knowledge, and stores them in its knowledge base. Moreover, a knowledge-based agent reasons with its knowledge: given a query, the agent derives new knowledge from what it already knows, and, using the new knowledge, answers the query. Queries are given to the agent as formulae of a *query language*, a language of symbolic logic to state queries. The advantage of a knowledge-based agent is the flexibility of its declarative knowledge. Declarative knowledge

can be easily modified and used for different purposes, even purposes that are not anticipated when it is assembled.

In this thesis, we rest our theoretical foundations on symbolic logic, and conduct the analysis of a knowledge-based agent through *logical entailment*: given a set T of sentences and a sentence ϕ in a language of symbolic logic, does every structure that satisfies each sentence in T satisfy the sentence ϕ (i.e., $T \models \phi$)? We formalize an agent's reasoning process as the following problem of logical entailment: given a knowledge base T in a representation language and a query ϕ in a query language, does T logically entail ϕ?

To achieve tractable reasoning, we need to choose our representation and query languages wisely. Symbolic logics are comparable on two dimensions: expressiveness and complexity. Given two logics L_1 and L_2, L_1 is *less expressive* than L_2 if every problem expressible in (a language of) L_1 is also expressible in (a language of) L_2, and L_1 is *equally as expressive* as L_2 if L_1 is less expressive than L_2 and L_2 is less expressive than L_1. A logic's (worst-case) complexity is the complexity of the most difficult problem expressible in (a language of) the logic. If logic L_1 is less expressive than logic L_2, then the complexity of L_1 is either less or equal to that of L_2, and hence, we cannot reduce the complexity by increasing the expressiveness [37].

It is often difficult to strike a balance between expressiveness and complexity in our representation and query languages. On one hand, an expressive logic is desirable because it enables us to state a wide range of phenomena, and hence, we wish to increase the expressiveness of our representation and query languages. On the other hand, too expressive a logic (e.g., the first-order logic) leads to intractable or even undecidable reasoning. One technique for handling computationally intractable problems is to add some constraints to the description of the problem, so that the search space for the answers becomes small or structured enough to admit fast algorithms. In the context of reasoning with knowledge, this technique leads to the restriction of the expressiveness of representation and query languages [5]. Logic L_1 is a *fragment* of logic L_2 if every sentence of L_1 is a sentence of L_2. In this thesis, we shall restrict the *situation calculus* and *ID-logic*, two rather expressive formalisms, and present their fragments as our representation and query languages.

An agent's environment evolves over time as a result of its actions. We hence consider and model the agent's environment, or rather, the agent's *knowledge* about its environment as a dynamic system. Many formalisms have been invented or suggested for modelling dynamic systems: process algebras, temporal and dynamic logics, finite automata, just to

name a few. However, both theoretically and practically, none of these formalisms seems as versatile as the situation calculus [32, 35]. McCarthy presented the situation calculus as a formalism for reasoning about instantaneous and discrete actions and as a way of formally describing dynamic systems. Instead of dealing with explicit time, the situation calculus deals with *situations*, which denote the sequences of actions. Reiter and his collaborators [40] developed the theoretical foundation of the situation calculus. Instead of considering a situation as a world state, as McCarthy did, Reiter considered it as a history that is a sequence of executed actions.

Initially, the situation calculus was a theoretical tool, without much practical use. It was the language of choice for investigating technical issues, such as the frame problem [33], which is to represent what remains the same in a dynamic system after a certain action is executed. Specifically, the representational frame problem is to represent the effects (or non-effects) of actions on a dynamic system with a small knowledge base.[1] Reiter's partial solution to the representational frame problem in the situation calculus [39], which is easy to implement by logic programming, has increased the popularity of the situation calculus as a representation language. Moreover, Reiter [40] suggested that the situation calculus had more potential than was initially perceived, and demonstrated his hypothesis mainly by extending the formalism to incorporate features such as concurrency, procedures, and probability in ways that provide for efficient implementations. In Chapter 2, we shall present the *propositional situation calculus*, a fragment of the situation calculus, as our representation language.

As for a query language, we consider a fragment of ID-logic [9]. ID-logic is the extension of first-order logic with inductive definitions, and provides a uniform and succinct way of representing various forms of inductive definitions. Denecker and Ternovska [9] observed that complex non-monotone inductive definitions occur not only in mathematics but also in common-sense reasoning, and that inductive definitions cannot be expressible in first-order logic. In Chapter 3, we shall present QL', a fragment of ID-logic, as our query language.

In this thesis, we formalize the agent's knowledge as a set $\mathcal{D}$ of sentences in a language of the propositional situation calculus, a query as a sentence ϕ of a language of QL' and the agent's reasoning process as the problem of logical entailment (i.e., $\mathcal{D} \models \phi$). Moreover, we solve the problem of logical entailment by a model-theoretic approach: we construct from

[1] Much related to the representational frame problem is the inferential frame problem, which is to project the result of a sequence of actions in a short time, but we do not discuss it any further in this thesis.

$\mathcal{D}$ a finite canonical model $\mathfrak{F}^{\mathcal{D}}$ of the agent's knowledge through the *filtration* operation, a model-theoretic operation of collapsing a possibly infinite structure into a quotient structure. We prove that $\mathcal{D}$ logically entails ϕ if and only if $\mathfrak{F}^{\mathcal{D}}$ satisfies ϕ; that is,

$$\mathcal{D} \models \phi \iff \models_{\mathfrak{F}^{\mathcal{D}}} \phi.$$

In other words, if formalized in the propositional situation calculus and QL', the problem of logical entailment is reducible to the problem of *model-checking*: does a model $\mathfrak{A}$ of a computing system satisfy a formal specification ψ (i.e., $\models_{\mathfrak{A}} \psi$)? In other words, to answer whether $\mathcal{D}$ logically entailments ϕ, instead of model-checking ϕ and each (possibly infinite) structure in the (possibly infinite) set of models of $\mathcal{D}$, we model-check ϕ and $\mathfrak{F}^{\mathcal{D}}$. We call this result the *reducibility theorem*. Furthermore, we present a reasoning algorithm that solves the problem of logical entailment by reducing it to the problem of model-checking. This algorithm is based on the linear-time model-checking algorithm for Datalog LITE [20], and runs in time linear with respect to both the size of $\mathcal{D}$ and the size of ϕ. Moreover, this algorithm does not assume that the agent has the complete knowledge of its initial environment.

Ternovska [45] and Liu and Levesque [31] showed work similar to this thesis. Ternovska [45] proposed a fragment of the situation calculus that allows second-order quantification over the uncountable action domain, and hence, the uncountable situation domain, and proved the decidability of the fragment by reducing the problem of logical entailment in the fragment to the decidable problem of emptiness for a tree automata. This work involves the construction of a canonical tree automaton that corresponds to a theory of actions in the situation calculus.

Liu and Levesque [31] proposed a methodology to establish the tractability of reasoning with expressive first-order knowledge bases. The methodology consists of defining a logic that is a fragment of the first-order logic with the following two properties: a) the problem of logical entailment in this logic coincides with the problem of model-checking against a small number of characteristic models; and b) the problem of model-checking itself is tractable for queries with a bounded number of variables in this logic. They applied the methodology to reasoning with first-order knowledge bases that include disjunctive information, and proved that the reasoning is tractable if both the knowledge base and the query use a bounded number of variables.

In Chapter 2, we shall present the propositional situation calculus, our representation

language. We shall also describe how to formalize the agent's knowledge as a set of sentences in the representation language and how to construct a finite canonical model of the agent's knowledge from the set of sentences. In Chapter 3, we shall present QL', our query language, and show its expressiveness. In Chapter 4, we shall prove the reducibility theorem and present our reasoning algorithm. In Chapter 5, we shall discuss two possible applications of the results of this thesis: planning under uncertainty and verification of robotics programs. Finally, in Chapter 6, we shall summarize this thesis and indicate the directions of our future research.

Chapter 2

Filtration

In this thesis, we formalize the agent's knowledge as a set $\mathcal{D}$ of sentences in a language of the propositional situation calculus, a query as a sentence ϕ of a language of QL', and the agent's reasoning process as the problem of logical entailment (i.e., $\mathcal{D} \models \phi$). Moreover, we solve the problem of logical entailment by a model-theoretic approach: we construct from $\mathcal{D}$ a finite canonical model $\mathfrak{F}^T$ of the agent's knowledge through the filtration operation, a model-theoretic operation that collapses a possibly infinite structure into a quotient structure. In this chapter, we present the propositional situation calculus, our representation language. We also describe how to formalize the agent's knowledge as a set of sentences in the representation language and how to construct a finite canonical model of the agent's knowledge from the set of sentences.

We shall first present the situation calculus, a well-known representation language in the knowledge representation community, and the propositional situation calculus as a fragment of the situation calculus. We shall also describe how to formalize the agent's knowledge in the representation language, as a *basic action theory*, a set of sentences that follow certain schema. Secondly, we shall describe the filtration operation on a possibly infinite structure. Thirdly, we shall describe the filtration operation on a basic action theory $\mathcal{D}$ in the representation language: we call the resulting structure the filtration of the set of models of $\mathcal{D}$ and denote it by $\mathfrak{F}^{\mathcal{D}}$. Finally, we shall prove that, $\mathfrak{F}^{\mathcal{D}}$ is a finite canonical model of the agent's knowledge such that the computational behaviour of $\mathfrak{F}^{\mathcal{D}}$ is equivalent to that of every model of $\mathcal{D}$.

2.1 The Propositional Situation Calculus

In this section, we first present the syntax and semantics of the situation calculus and show how to axiomatize a set of dynamic systems by a set of sentences in a language of the situation calculus. We then present the propositional situation calculus, our representation language, as a fragment of the situation calculus.

2.1.1 The Situation Calculus

A language of the situation calculus is a language of second-order logic with equality. It has three disjoint sorts: the *situation* sort, the *action* sort, and the *object* sort. We use the action terms to represent changes in the world, the situation terms to represent finite sequences of actions, which we call histories, and the object terms to represent any other kinds of objects that possibly exist in the world. We shall denote the domain of situations, the domain of actions, and the domain of objects by Sit, Act, and Obj, respectively.

A vocabulary τ of the situation calculus contains two function symbols that return values of the situation sort: a constant symbol $S_0 : \rightarrow \text{Sit}$ that denotes the initial situation and a binary function symbol $do : \text{Act} \times \text{Sit} \rightarrow \text{Sit}$. Let a and s be an action term and a situation term, respectively. A situation term $do(a, s)$ denotes a situation reachable from a situation s by the execution of an action a. We can also interpret the situation term $do(a, s)$ as a sequence of actions that we construct by adding the action a to the sequence of actions that s denotes. Note that S_0 and do are the only function symbols in τ that return values of the situation sort.

Moreover, τ may contain two binary predicate symbols $\sqsubset$ and $Poss$. The symbol $\sqsubset$ denotes a partial order on Sit; given two situations s and s' in Sit, $s \sqsubset s'$ means that we can obtain the sequence s' of actions from the sequence s of actions by adding one or more actions to the front of s. The symbol $Poss$ denotes a binary relation on a set $\text{Act} \times \text{Sit}$; given an action a in Act and a situation s in Sit, $Poss(a, s)$ means that it is possible to execute the action that a denotes in the situation that s denotes.

Depending on an application domain, τ contains a countable, possibly infinite number of the following symbols:

- For some integer r in $\mathbb{N}$, predicate symbols of arity r of the following sort:

$$(\text{Act} \cup \text{Obj})^r$$

such as $Human(Joe)$ and $Sensing(smell)$. They denote situation-independent relations.

- For some integer r in $\mathbb{N}$, function symbols of arity r of the following sort:

$$(\text{Act} \cup \text{Obj})^r \rightarrow \text{Obj}$$

such as $sqrt(x)$ and $time(Jane, 100m\_run)$. They denote situation-independent functions.

- For some integer r in $\mathbb{N}$, function symbols of arity r of the following sort:

$$(\text{Act} \cup \text{Obj})^r \rightarrow \text{Act}$$

such that $pick\_up(x)$ and $do\_laundry(wash, dry, y)$. They denote *action functions* or *action constants* if $r = 0$.

- For some integer r in $\mathbb{N}$, predicate symbols of arity $r + 1$ of the following sort:

$$(\text{Act} \cup \text{Obj})^r \times \text{Sit}.$$

They denote *relational fluents*. A relational fluent is a situation-dependent relation that changes its truth values from situation to situation. Furthermore, for some integer r in $\mathbb{N}$, function symbols of arity $r + 1$ of the following sort:

$$(\text{Act} \cup \text{Obj})^r \times \text{Sit} \rightarrow \text{Act} \cup \text{Obj}.$$

They denote *functional fluents*. A functional fluent is a situation-dependent function that returns different values from situation to situation. Relational and functional fluents take only one argument of the situation sort that is always its last argument.

In a dynamic system, attributes of an object and relationships between objects change over time, and relational and functional fluents capture such dynamics. For example, consider the following block world: in the initial situation, blocks A and B are on the table, and nothing is on either A or B. The truth value of $On(A, B, S_0)$ should be true, whereas the truth value of $On(A, B, do(put\_on(B, A), do(pick\_up(B), S_0)))$ should be false.

2.1.2 Basic Action Theory

We can succinctly axiomatize knowledge about an application domain, especially, the properties of actions and situations, via a *basic action theory* in a language of the situation calculus. A basic action theory consists of five parts: unique name axioms for actions, the foundational axioms for situations, action precondition axioms, successor state axioms, and the description of the initial situation. We shall describe the parts sequentially.

Definition 2.1.1 (Unique Name Axioms). *Unique name axioms* for a sort S, $\mathcal{D}_{\text{una}}(S)$, are sentences of the following schema: for every pair of distinct function symbols f and g of the sort S,

$$\forall \bar{x} \forall \bar{y}[\neg f(\bar{x}) = g(\bar{y})],$$

and, for every function symbol f of the sort S with an arity greater than zero,

$$\forall \bar{x} \forall \bar{y}[f(\bar{x}) = f(\bar{y}) \supset \bar{x} = \bar{y}].$$

Unique name axioms for actions are then $\mathcal{D}_{\text{una(Act)}}$.

Definition 2.1.2 (Foundational Axioms for Situations). The *foundational axioms for situations* consist of the following axioms:

- The unique name axioms for situations: $\mathcal{D}_{\text{una}}(\text{Sit})$.
- The domain closure axiom for situations:

$$\forall P[P(S_0) \wedge \forall s \forall a[P(s) \supset P(do(a, s))] \supset \forall s P(s)].$$

- Two axioms to define a partial order on Sit that the symbol $\sqsubset$ denotes:

$$\forall s[\neg s \sqsubset S_0], \tag{2.1}$$

$$\forall s \forall s'[s \sqsubset do(a, s') \equiv s \sqsubseteq s'] \tag{2.2}$$

  where $\sqsubseteq$ is an abbreviation for $s \sqsubset s' \vee s = s'$.

The domain closure axiom for situations implies that the situation domain Sit is the smallest set that includes the initial situation S_0 and is closed under the *do* function, for every subset of Sit that includes the initial situation S_0 and is closed under the *do* function is equal to Sit itself. We do not impose any restrictions on the action domain Act: it can be

finite, countable, or even uncountable. For example, Act will be uncountable if an action function has an argument that ranges over real numbers (e.g., the time of action execution).[1]

The foundational axioms for situations are domain-independent. They provide the basic properties of situations in any axiomatization of particular fluents and actions. We can represent the situation domain of a model of the foundational axioms for situations by a tree with the branching degree equal to the cardinality of the action domain. If the action domain is finite, the situation domain composes a finitely branching tree.

In a model of the foundational axioms for situations, two situations are identical if and only if they are reachable from the initial situation by the same sequence of actions. It is possible that two situations are different but have the same truth values to all fluents. Thus, we cannot identify a situation by the set of fluents that hold in the situation; that is, by a *state*. The proper way to understand a situation is as a *history*, a finite sequence of actions; two situations are identical if and only if they denote identical histories. This is the major reason for using the term "situation" instead of "state"; the latter carries with it the connotation of a snapshot of the world. In the situation calculus, situations are not snapshots, they are finite sequences of actions. While states can repeat themselves—the same snapshot of the world can happen more than once—situations cannot.

Before presenting the domain-dependent parts of a basic action theory, namely, action precondition axioms, successor state axioms, and the description of the initial situation, we need to define the uniformity of a formula of the situation calculus. We say that a formula ϕ of the situation calculus is *uniform* in a situation term σ if ϕ satisfies the following conditions:

- ϕ mentions neither *Poss* nor $\sqsubset$.
- There is no quantification over variables of the situation sort in ϕ.
- There is no equality between situation terms in ϕ.
- If ϕ mentions a term of the situation sort as the situation argument of a fluent symbol, then it should be σ.

Definition 2.1.3 (Action Precondition Axiom). An *action precondition axiom* for an

[1] If Act is uncountable, then Sit will be uncountable.

action function that a symbol a denotes is a sentence of the following schema:

$$\forall \bar{x} \forall s[Poss(a(\bar{x}), s) \equiv \Pi_a(\bar{x}, s)]$$

where $\Pi_a(\bar{x}, s)$ is a formula that is uniform in s, and the free variables of $\Pi(\bar{x}, s)$ are among $\bar{x}$ and s.

The uniformity of Π_a in s ensures that the current situation determines the precondition of the action's executability.

Definition 2.1.4 (Successor State Axioms). A *successor state axiom* for a relational fluent that a symbol F denotes is a sentence of the following schema:

$$\forall \bar{x} \forall a \forall s[F(\bar{x}, do(a, s)) \equiv \Phi_F(\bar{x}, a, s)],$$

where $\Phi_F(\bar{x}, a, s)$ is a formulae that is uniform in s, and the free variables of $\Phi_F(\bar{x}, a, s)$ are among $\bar{x}$, a, and s. Moreover, a successor state axiom for a functional fluent that a symbol f denotes is a sentence of the following schema:

$$\forall \bar{x} \forall y \forall a \forall s[f(\bar{x}, do(a, s)) = y \equiv \phi_f(\bar{x}, y, a, s)],$$

where $\phi_f(\bar{x}, a, s)$ is a formulae that is uniform in s, and the free variables of $\phi_f(\bar{x}, a, s)$ are among $\bar{x}$, a, and s.

The uniformity of Φ_F in s ensures that the current situation determines the truth value of the relational fluent in the successor situation. This is call the *Markov property* in control and systems theory.[2] Moreover, the uniformity of ϕ_f in s ensures the Markov property of the value of the functional fluent.

For example, suppose that we are given the following successor state axiom of the block world domain:

$$\forall a \ \forall s \forall x Broken(x, do(a, s)) \equiv a = drop(x, s) \vee Broken(x, s) \wedge \neg a = repair(x).$$

It tells us that a block will get broken if the agent drops it, and that a block will remain broken if it is already broken and the agent does not repair it.

[2]For the non-Markov extension of the situation calculus, see Gabaldon [15].

Definition 2.1.5 (Description of the Initial Situation). The *description of the initial situation* is a set of first-order sentences that are uniform in S_0. We often call it the *initial database* because it represents the agent's initial knowledge about an application domain.

For example, suppose that we are given the following initial database of the block world domain: $\forall x \neg Broken(x, S_0)$. It states that no objects are broken at the beginning.

The initial database may contain sentences that mention no situation terms at all; for example, non-temporal facts such as $Mountain(MtEverest)$ and $\forall x[dog(x) \supset mammal(x)]$.

We say that the initial database is *incomplete* if it does not provide complete knowledge of the initial situation. For example, suppose that there are two blocks A and B in the block world domain, and that we are given the following initial database: $NothingOnTop(A)$. It states that there is initially no block on top of the block A, but states nothing about the block B. The initial database does not provide the agent with complete knowledge of the initial situation of the block world domain, and hence it is incomplete.

Definition 2.1.6 (Basic Action Theory). Suppose that $\mathcal{D}$ is the following set of sentences in a language of the situation calculus:

$$\mathcal{D} := \mathcal{D}_{\text{una(Act)}} \cup \mathcal{D}_{\text{f}} \cup \mathcal{D}_{\text{apa}} \cup \mathcal{D}_{\text{ss}} \cup \mathcal{D}_{S_0},$$

where:

- $\mathcal{D}_{\text{una(Act)}}$ is a set of unique name axioms for actions.
- $\mathcal{D}_{\text{f}}$ is the set of foundational axioms for situations.
- $\mathcal{D}_{\text{apa}}$ is a set of action precondition axioms.
- $\mathcal{D}_{\text{ss}}$ is a set of successor state axioms.
- $\mathcal{D}_{S_0}$ is the initial database.

We say that $\mathcal{D}$ is a *basic action theory* if each successor state axiom for a functional fluent in $\mathcal{D}$ satisfies the following *consistency property for functional fluents*: if a successor state axiom for a functional fluent that a symbol f denotes is

$$\forall \bar{x} \forall y \forall a \forall s[f(\bar{x}, do(a, s)) = y \equiv \phi_f(\bar{x}, y, a, s)],$$

then

$$\mathcal{D}_{\text{una(Act)}} \cup \mathcal{D}_{S_0} \models \forall \bar{x}[\exists y \phi_f(\bar{x}, y, a, s) \wedge \forall y \forall y'[\phi_f(\bar{x}, y, a, s) \wedge \phi_f(\bar{x}, y', a, s) \supset y = y']].$$

The consistency property for functional fluents ensures that the condition that defines the value of the functional fluent in the successor situation does indeed define the value, and that the value is unique. It hence prevents inconsistency in the successor state axiom for the functional fluent. Moreover, the consistency property for functional fluents leads to the following theorem:

Theorem 2.1.7 (Relative Satisfiability [41]). *A basic action theory $\mathcal{D}$ is satisfiable if and only if $\mathcal{D}_{\text{una(Act)}} \cup \mathcal{D}_{S_0}$ is satisfiable.*

The preceding theorem ensures that, if the initial database and unique name axioms for actions are satisfiable, then we will not introduce unsatisfiability by augmenting these axioms with the foundational axioms for situations, action precondition axioms, and successor state axioms.

Given a basic action theory $\mathcal{D}$, we shall let $\mathcal{D}_{\text{una(Act)}}$, $\mathcal{D}_{\text{f}}$, $\mathcal{D}_{\text{apa}}$, $\mathcal{D}_{\text{ss}}$, and $\mathcal{D}_{S_0}$ denote the set of unique name axioms for actions, the set of foundational axioms for situations, the set of action precondition axioms, the set of successor state axioms, and the initial database, respectively.

2.1.3 Syntax of The Propositional Situation Calculus

The propositional situation calculus is a fragment of the situation calculus, and is a representation language that we use to axiomatize an agent's knowledge base in this thesis. A vocabulary τ of the propositional situation calculus is a vocabulary of the situation calculus that consists only of S_0, do, a finite number of action constant symbols, and a finite number of unary, relational fluent symbols. The propositional situation calculus does not consider the partial order on a set of situations, and hence τ does not include the symbol $\sqsubset$.[3] Moreover, the propositional situation calculus assumes that every action is always executable, and hence τ does not include the symbol $Poss$.[4]

In the τ-language of the propositional situation calculus, the foundational axioms for situations consist only of the unique name axioms for situations and the domain closure axioms for situations. We do not need the two axioms 2.1 and 2.2 for the symbol $\sqsubset$ because

[3]The partial order on a set of situations is definable in our query language QL, which we shall introduce in Chapter 3.

[4]It is not difficult to lift this assumption if we allow only action preconditions that are simple (e.g. a conjunction of literals).

τ does not contain $\sqsubset$. Moreover, we do not have any precondition axioms for actions because τ does not contain *Poss*. As for successor state axioms and the initial database, the propositional situation calculus imposes further syntactic restrictions on their schemata, which we shall discuss sequentially.

We shall call atomic formulae or their negations *literals*, which originally mean propositional symbols or their negations.

Definition 2.1.8 (Propositional Successor State Axiom). A *propositional successor state axiom* for a unary relational fluent that a symbol F denotes is an axiom of the following schema:

$$\forall a \forall s[F(do(a,s)) \equiv \gamma_F^+(a,s) \vee F(s) \wedge \neg\gamma_F^-(a,s)],$$

where $\gamma_F^+(a,s)$ and $\gamma_F^-(a,s)$ are first-order formulae of the disjunctive normal form that are uniform in s, and the free variables in $\gamma_F^+(a,s)$ and $\gamma_F^-(a,s)$ are among a and s. Moreover, each conjunct in $\gamma_F^+(a,s)$ and $\gamma_F^-(a,s)$ takes the following form:

$$a = act \wedge \phi(s)$$

where *act* is an action constant symbol, ϕ is a conjunction of literals, and s is the only free variable that occurs in ϕ.

The uniformity of γ_F^+ and γ_F^- ensures the Markov property of the truth value of the unary relational fluent. Moreover, $\gamma_F^+(a,s)$ formalizes the circumstances that cause the unary relational fluent to be true whereas $\gamma_F^-(a,s)$ formalizes the circumstances that cause it to be false.[5]

Definition 2.1.9 (Propositional Description of the Initial Situation). The *propositional description of the initial situation* is a set of literals that are uniform in S_0. We shall also call it the *propositional initial database*.

Definition 2.1.10 (Basic Action Theory of the Propositional Situation Calculus). Suppose that $\mathcal{D}$ is the following set of sentences in a language of the propositional situation calculus:

$$\mathcal{D} := \mathcal{D}_{\text{una(Act)}} \cup \mathcal{D}_{\text{f}} \cup \mathcal{D}_{\text{pss}} \cup \mathcal{D}_{\text{p}S_0},$$

where:

[5]We can alternatively encode simple action preconditions (e.g. a conjunction of literals) in γ_F^+ and γ_F^-.

- $\mathcal{D}_{\text{una(Act)}}$ is a set of unique name axioms for actions.
- $\mathcal{D}_{\text{f}}$ is the set of foundational axioms for situations.
- $\mathcal{D}_{\text{apa}}$ is a set of action precondition axioms.
- $\mathcal{D}_{\text{pss}}$ is a set of propositional successor state axioms.
- $\mathcal{D}_{\text{p}S_0}$ is the propositional initial database.

We say that $\mathcal{D}$ is a *basic action theory* of the propositional situation calculus.

Example 2.1.11. Suppose that τ is the following vocabulary of the propositional situation calculus:

$$\tau := \{S_0, do, drop\_vase, repair\_vase, VaseBroken\}$$

where *drop_vase* and *repair_vase* are action constant symbols and *VaseBroken* is a unary relational fluent symbol.

In the τ-language of the propositional situation calculus, the unique name axiom for actions is $\neg drop\_vase = repair\_vase$. The following successor state axiom

$$\forall a \forall s[VaseBroken(do(a, s)) \equiv a = drop\_vase \vee VaseBroken(s) \wedge \neg a = repair\_vase]$$

means that the vase will be broken if it is dropped, or if it is already broken and does not get repaired. Moreover, the following description of the initial situation

$$\neg VaseBroken(S_0)$$

means that the vase is initially unbroken.

2.2 Filtration

In the preceding section, we have presented the propositional situation calculus, our representation language, and have shown how to formalize the agent's knowledge as a basic action theory in the representation language. In this section, we describe the filtration operation, a model-theoretic operation that collapses a possibly infinite structure into a quotient structure. We shall first describe the filtration operation on a possibly infinite labelled transition system, which is indeed a relational structure. We shall then describe the filtration operation on a possibly infinite structure over a vocabulary of the propositional situation calculus.

2.2.1 Transitional Version

We describe the filtration operation on a possibly infinite labelled transition system. Suppose that ρ and σ are a set of action symbols and a set of proposition symbols, respectively.

Definition 2.2.1 (Labelled Transition System). A *labelled transition system* is the following tuple:

$$(\mathrm{St}, \{R_a\}_{a\in\rho}, \mathcal{L})$$

where St is a non-empty set, R_a is a binary relation on a set St $\times$ St, and $\mathcal{L}$ is a unary function that maps an element of St to a subset of σ. We call St a *state set*, the elements of St *states*, R_a a *transition relation*, and $\mathcal{L}$ a *labelling function.*

Given a labelled transition system $(\mathrm{St}, \{R_a\}_{a\in\rho}, \mathcal{L})$, we define a non-empty binary relation R_σ on a set St $\times$ St such that $(st, st') \in R_\sigma$ if and only if, for each symbol P in σ,

$$P \in \mathcal{L}(st) \iff P \in \mathcal{L}(st');$$

that is, $\mathcal{L}(st) = \mathcal{L}(st')$. Moreover, R_σ is an equivalence relation. We denote the equivalence class of a state st in St with respect to R_σ by $|st|_\sigma$, or simply by $|st|$ if σ is clear from the context.

Definition 2.2.2 (Filtration—the Transitional Version). Given a labelled transition system $\mathfrak{M}$, a *filtration* of $\mathfrak{M}$ through σ is a labelled transition system $(\mathrm{St}^f, \{R_a{}^f\}_{a\in\rho}, \mathcal{L}^f)$ that satisfies the following conditions:

(i) $\mathrm{St}^f = \{|st| : st \in \mathrm{St}^{\mathfrak{M}}\}$; that is, St^f is the set of equivalence classes of states in $\mathrm{St}^{\mathfrak{M}}$ with respect to R_σ.

(ii) For each state $|st|$ in St^f, $\mathcal{L}^f(|st|) = \mathcal{L}^{\mathfrak{M}}(st)$.

(iii) If, for some symbol a in ρ, $(st_1, st_2) \in R_a{}^{\mathfrak{M}}$, then $(|st_1|, |st_2|) \in R_a{}^f$.

We denote a filtration of $\mathfrak{M}$ through σ by $\mathfrak{M}^f_\sigma$, or by $\mathfrak{M}^f$ if σ is clear from the context.

The filtration operation which we consider in this thesis is the modification of the filtration operation that Blackburn et al. presented [3]. In the original filtration operation, the equivalence relation on a set St $\times$ St is definable with respect to any subformula-closed set of formulae of a modal language. The filtration operation in this thesis is the special case of the original filtration operation such that the equivalence relation on St $\times$ St is definable only with respect to a set of propositions.

2.2.2 Structural Version

We describe the filtration operation on a possibly infinite structure over a vocabulary of the propositional situation calculus. The structural version of the filtration operation is a slight modification of the transitional version. In the structural version, we define a non-empty, binary relation on each domain of a given structure. Suppose that τ is a vocabulary of the propositional situation calculus, and that ρ and σ are the set of action constant symbols in τ and the set of unary relational fluent symbols in τ, respectively.

Given a τ-structure $\mathfrak{A}$ which satisfies the unique name axioms for actions, we define a non-empty binary relation R_σ on a set $\mathrm{Sit}^{\mathfrak{A}} \times \mathrm{Sit}^{\mathfrak{A}}$ such that $(s, s') \in R_\sigma$ if and only if, for each symbol F in σ,

$$s \in F^{\mathfrak{A}} \iff s' \in F^{\mathfrak{A}}.$$

Similarly, we define a non-empty binary relation R_ρ on a set $\mathrm{Act}^{\mathfrak{A}} \times \mathrm{Act}^{\mathfrak{A}}$ such that $(a, a') \in R_\rho$ if and only if, for each symbol act in ρ,

$$a = act^{\mathfrak{A}} \iff a' = act^{\mathfrak{A}}.$$

Moreover, R_σ and R_ρ are equivalence relations. We denote the equivalence class of a situation s in $\mathrm{Sit}^{\mathfrak{A}}$ with respect to R_σ by $|s|_\sigma$, or simply by $|s|$ if σ is clear from the context. Similarly, we denote the equivalence class of an action a in $\mathrm{Act}^{\mathfrak{A}}$ with respect to R_ρ by $|a|_\rho$, or simply by $|a|$ if ρ is clear from the context.

We shall let $\tau \backslash S_0$ denote the set $\tau - \{S_0\}$.

Definition 2.2.3 (Filtration—the Structural Version). Given a τ-structure $\mathfrak{A}$ which satisfies the unique name axioms for actions, a *filtration* of $\mathfrak{A}$ is a $(\tau \backslash S_0)$-structure $\mathfrak{F}$ that satisfies the following conditions:

(i) $\mathrm{Sit}^{\mathfrak{F}} = \{|s| : s \in \mathrm{Sit}^{\mathfrak{A}}\}$; that is, $\mathrm{Sit}^{\mathfrak{F}}$ is the set of equivalence classes of situations in $\mathrm{Sit}^{\mathfrak{A}}$ with respect to R_σ.

(ii) $\mathrm{Act}^{\mathfrak{F}} = \{|a| : a \in \mathrm{Act}^{\mathfrak{A}}\}$; that is, $\mathrm{Act}^{\mathfrak{F}}$ is the set of equivalence classes of actions in $\mathrm{Act}^{\mathfrak{A}}$ with respect to R_ρ.

(iii) For each situation $|s|$ in $\mathrm{St}^{\mathfrak{F}}$ and each symbol F in σ,

$$|s| \in F^{\mathfrak{F}} \iff s \in F^{\mathfrak{A}}.$$

(iv) For each action a in $\text{Act}^{\mathfrak{A}}$ and each symbol act in ρ, if $act^{\mathfrak{A}} = a$, then $act^{\mathfrak{F}} = |a|$.

(v) If $do^{\mathfrak{A}}(a, s_1) = s_2$, then $do^{\mathfrak{F}}(|a|, |s_1|) = |s_2|$.

We denote a filtration of $\mathfrak{A}$ by $\mathfrak{A}^f$. (This notation does not include subscripts for ρ and σ because ρ and σ are clear once τ is given.)

Often, not every element in the action domain is identifiable by an action constant symbol. We call actions that we cannot identify *unnamed actions*. If $\mathfrak{A}$ is a τ-structure and $\text{Act}^{\mathfrak{A}}$ contains some unnamed actions, $\text{Act}^{\mathfrak{A}^f}$ will contain one unnamed action, and it is the equivalence class of unnamed actions in $\text{Act}^{\mathfrak{A}}$.

2.3 Filtration Operation on a Basic Action Theory

In the previous section, we have described the transitional and structural versions of the filtration operation. In this section, we describe the filtration operation on a basic action theory in a language of the propositional situation calculus. In the transitional and structural versions of the filtration operation, we extract the essence of a structure. Through the filtration operation on a basic action theory $\mathcal{D}$, we extract the essence of a set of structures (i.e., the set of models of $\mathcal{D}$).

Suppose that τ is a vocabulary of the propositional situation calculus, and that ρ and σ are the set of action constant symbols in τ and the set of unary relational fluent symbols in τ, respectively. We shall view ρ and σ as lexicographically (totally) ordered sets. We denote the smallest element in ρ as act_1, the second smallest element in ρ as act_2, and so on. Similarly, we denote the smallest element in σ as F_1, the second smallest element in σ as F_2, and so on. Moreover, we denote the cardinality of ρ and the cardinality of σ by m and n, respectively.

Given a basic action theory $\mathcal{D}$ in the τ-language of the propositional situation calculus, the filtration $\mathfrak{F}$ of the set of models of $\mathcal{D}$ has the situation domain $\text{Sit}^{\mathfrak{F}}$ and the action domain $\text{Act}^{\mathfrak{F}}$ that are the vector spaces, $\{0,1\}^n$ and $\{0,1\}^m$, respectively. For each situation $\vec{s}$ in $\text{Sit}^{\mathfrak{F}}$, $(\vec{s})_i$ denotes a unary function $\{1, 2, \ldots, n\} \mapsto \{0, 1\}$ that returns the value of the i-th coordinate of the vector $\vec{s}$. For example, for $n = 3$ and $\vec{s} = (0, 1, 1)$, $(\vec{s})_1 = 0, (\vec{s})_2 = 1, (\vec{s})_3 = 1$. Similarly, for each action $\vec{a}$ in $\text{Act}^{\mathfrak{F}}$, $(\vec{a})_i$ denotes a unary function $\{1, 2, \ldots, m\} \mapsto \{0, 1\}$ that returns the value of the i-th coordinate of the vector $\vec{a}$.

Given that a structure $\mathfrak{A}$ interprets the symbols in τ, we shall use the following notations: for each integer i from 1 to n, if s is an element of $\mathrm{Sit}^{\mathfrak{A}}$ and t is a term of the situation sort,

$$L_i^{\vec{s}}(t) = \begin{cases} F_i(t) & \text{if } s \in F_i{}^{\mathfrak{A}} \\ \neg F_i(t) & \text{otherwise;} \end{cases}$$

moreover, for each integer i from 1 to m, if a is an element of $\mathrm{Act}^{\mathfrak{A}}$ and t is a term of the action sort,

$$L_i^{\vec{a}}(t) = \begin{cases} t = act_i & \text{if } a = act_i{}^{\mathfrak{A}} \\ \neg t = act_i & \text{otherwise.} \end{cases}$$

Definition 2.3.1. Given a basic action theory $\mathcal{D}$ in the τ-language of the propositional situation calculus, the *filtration* of the set of models of $\mathcal{D}$ is a $(\tau \backslash S_0)$-structure $\mathfrak{F}$ that satisfies the following conditions:

(i) $\mathrm{Sit}^{\mathfrak{F}} = \{\vec{s} : \vec{s} \in \{0,1\}^n\}$; that is, $\mathrm{Sit}^{\mathfrak{F}}$ is the vector space $\{0,1\}^n$.

(ii) $\mathrm{Act}^{\mathfrak{F}} = \{\vec{a} : \vec{a} \in \{0,1\}^m,\ \|\vec{a}\| = 0 \text{ or } \|\vec{a}\| = 1\}$; that is, $\mathrm{Act}^{\mathfrak{F}}$ consists of the unit vectors and the zero vector of the vector space $\{0,1\}^m$.

(iii) For each situation $\vec{s}$ in $\mathrm{St}^{\mathfrak{F}}$ and each integer i from 1 to n,

$$\vec{s} \in F_1{}^{\mathfrak{F}} \iff (\vec{s})_i = 1.$$

(iv) For each action $\vec{a}$ in $\mathrm{Act}^{\mathfrak{F}}$ and each integer i from 1 to m, if $(\vec{a})_i = 1$, then $act_i{}^{\mathfrak{F}} = \vec{a}$.

(v) $do^{\mathfrak{F}}(\vec{a}, \vec{s_1}) = \vec{s_2}$ if, for some action variable x_a and for some situation variable x_s,

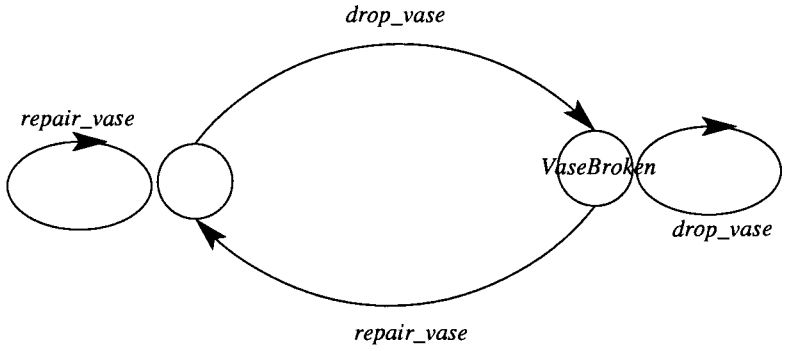
$$\left\{ \bigwedge_{i=1}^{m} L_i^{\vec{a}}(x_a), \bigwedge_{i=1}^{n} L_i^{\vec{s_1}}(x_s) \right\} \cup \mathcal{D}_{\mathrm{ss}} \vDash \bigwedge_{i=1}^{n} L_i^{\vec{s_2}}(do(x_a, x_s)).$$

We denote the filtration of $\mathcal{D}$ by $\mathfrak{F}^{\mathcal{D}}$, or by $\mathfrak{F}$ if $\mathcal{D}$ is clear from the context.

Example 2.3.2. We shall construct the filtration $\mathfrak{F}$ of the set of models of the basic action

theory in Example 2.1.11:

$$
\begin{aligned}
&\mathrm{Sit}^{\mathfrak{F}} = \{(0),(1)\},\\
&\mathrm{Act}^{\mathfrak{F}} = \{(0,0),(1,0),(0,1)\},\\
&drop\_vase^{\mathfrak{F}} = (1,0), repair\_vase^{\mathfrak{A}} = (0,1),\\
&VaseBroken^{\mathfrak{F}} = \{(1)\},\\
&do^{\mathfrak{F}}((0,0),(0)) = (0), do^{\mathfrak{F}}((0,0),(1)) = (1),\\
&do^{\mathfrak{F}}((1,0),(0)) = (1), do^{\mathfrak{F}}((0,1),(0)) = (0),\\
&do^{\mathfrak{F}}((1,0),(1)) = (0), do^{\mathfrak{F}}((0,1),(1)) = (1),
\end{aligned}
$$



The filtration $\mathfrak{F}^{\mathcal{D}}$ of the set of models of $\mathcal{D}$ is not a model of $\mathcal{D}$ per se. However, as we shall show in the next section, the computational behaviour of $\mathfrak{F}^{\mathcal{D}}$ is equivalent to that of each model of $\mathcal{D}$.

2.4 Computational Equivalence via Bisimulation

In the preceding section, we have described the filtration operation on a basic action theory in a language of the propositional situation calculus. In this section, we formalize the concept of computational equivalence via bisimulation, an equivalence relation on a set of computation models. As far as computational behaviours are concerned, it is adequate to model computing systems as labelled transition systems, and study these models up to bisimulation equivalence [36]. Moreover, we show that, given a basic action theory $\mathcal{D}$ in a language of the propositional situation calculus, the filtration $\mathfrak{F}^{\mathcal{D}}$ is a finite canonical model

of the agent's behaviour such that the computational behaviour of $\mathfrak{F}^{\mathcal{D}}$ is equivalent to that of each model of $\mathcal{D}$.

We shall first present a bisimulation on a set of labelled transition systems. Secondly, we shall present a bisimulation on a set of structures over a vocabulary of the propositional situation calculus. Finally, we shall show that, given a basic action theory $\mathcal{D}$ in a language of the propositional situation calculus, the filtration $\mathfrak{F}^{\mathcal{D}}$ of the set of models of $\mathcal{D}$ is bisimilar to each model of $\mathcal{D}$; that is, the computational behaviour of $\mathfrak{F}^{\mathcal{D}}$ is equivalent to that of each model of $\mathcal{D}$.

2.4.1 Transitional Version

We present a bisimulation on a set of labelled transition systems. A labelled transition system is a standard way of thinking about computation. When we traverse a labelled transition system, we build a sequence of state transitions; in other words, we compute. When two labelled transition systems are bisimilar, they build the same sequence of state transitions, and hence, they are computationally equivalent.

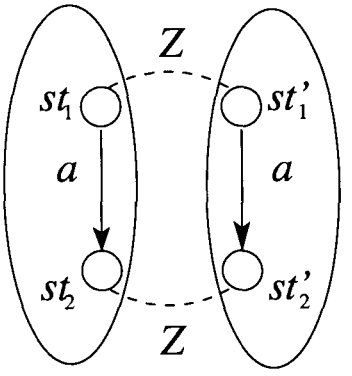
Definition 2.4.1 (Bisimulation—the Transitional Version). Suppose that ρ and σ are a set of action symbols and a set of proposition symbols, respectively. Given two labelled transition systems $\mathfrak{M}$ and $\mathfrak{N}$, a *bisimulation* between $\mathfrak{M}$ and $\mathfrak{N}$ is a non-empty binary relation Z on a set $St^{\mathfrak{M}} \times St^{\mathfrak{N}}$ that satisfies the following conditions:

(i) If $(st, st') \in Z$, then, for each symbol P in σ, $st \in P^{\mathfrak{M}} \iff st' \in P^{\mathfrak{N}}$.

(ii) If $(st_1, st_1') \in Z$ and, for some symbol a in ρ, $(st_1, st_2) \in {R_a}^{\mathfrak{M}}$, then there exists a state st_2' in $\mathrm{St}^{\mathfrak{N}}$ such that $(st_2, st_2') \in Z$ and $(st_1', st_2') \in {R_a}^{\mathfrak{N}}$ (the *forth condition*).

(iii) The converse of (ii): if $(st_1, st_1') \in Z$ and, for some symbol a in ρ, $(st_1', st_2') \in {R_a}^{\mathfrak{N}}$, then there exists a state st_2 in $\mathrm{St}^{\mathfrak{M}}$ such that $(st_2, st_2') \in Z$ and $(st_1, st_2) \in {R_a}^{\mathfrak{M}}$ (the *back condition*).

If there exists a bisimulation between two transition systems $\mathfrak{M}$ and $\mathfrak{N}$, then we say that $\mathfrak{M}$ and $\mathfrak{N}$ are *bisimilar*, and we denote it by $\mathfrak{M} \underline{\leftrightarrow} \mathfrak{N}$. Moreover, if there exists a bisimulation between two states st and st', then we say that st and st' are bisimilar. Bisimilar states have identical propositional information and matching transition possibilities; that is, whenever it is possible to make a transition in one state, it is possible to make a matching transition in the other.



2.4.2 Structural Version

We present a bisimulation on a set of structures over a vocabulary of the propositional situation calculus. The structural version of bisimulation is the slight modification of the transitional version. In the structural version, we have the back and forth conditions for unnamed actions.

Definition 2.4.2 (Bisimulation—the Structural Version). Suppose that ρ and σ are a set of action constant symbols and a set of unary relational fluent symbols, respectively. Moreover, suppose that two structures $\mathfrak{A}$ and $\mathfrak{B}$ interpret the symbols in ρ and σ and the function symbol do, and that $\mathfrak{A}$ and $\mathfrak{B}$ satisfy the unique name axioms for actions. A *bisimulation* between $\mathfrak{A}$ and $\mathfrak{B}$ is a non-empty binary relation Z on a set $\mathrm{Sit}^{\mathfrak{A}} \times \mathrm{Sit}^{\mathfrak{B}}$ that satisfies the following conditions:

(i) If $(s, s') \in Z$, then, for each symbol F in σ, $s \in F^{\mathfrak{A}} \iff s' \in F^{\mathfrak{B}}$.

(ii) If $(s_1, s'_1) \in Z$ and, for some symbol act in ρ, $do^{\mathfrak{A}}(act^{\mathfrak{A}}, s_1) = s_2$, then there exists a situation s'_2 in $\mathrm{Sit}^{\mathfrak{B}}$ such that $(s_2, s'_2) \in Z$ and $do^{\mathfrak{B}}(act^{\mathfrak{B}}, s'_1) = s'_2$ (the forth condition).

(iii) If $(s_1, s'_1) \in Z$ and, for some unnamed action a in $Act^{\mathfrak{A}}$, $do^{\mathfrak{A}}(a, s_1) = s_2$, then there exists a situation s'_2 in $\mathrm{Sit}^{\mathfrak{B}}$ such that $(s_2, s'_2) \in Z$ and, for some unnamed action a' in $Act^{\mathfrak{B}}$, $do^{\mathfrak{B}}(a', s'_1) = s'_2$ (the forth condition for an unnamed action).

(iv) The converse of (ii): if $(s_1, s'_1) \in Z$ and, for some symbol act in ρ, $do^{\mathfrak{B}}(act^{\mathfrak{B}}, s'_1) = s'_2$, then there exists a situation s_2 in $\mathrm{Sit}^{\mathfrak{A}}$ such that $(s_2, s'_2) \in Z$ and $do^{\mathfrak{A}}(act^{\mathfrak{A}}, s_1) = s_2$ (the back condition).

(v) The converse of (iii): if $(s_1, s_1') \in Z$ and, for some unnamed action a' in $Act^{\mathfrak{B}}$, $do^{\mathfrak{B}}(a', s_1') = s_2'$, then there exists s_2 such that $(s_2, s_2') \in Z$ and, for some unnamed action a in $Act^{\mathfrak{A}}$, $do^{\mathfrak{A}}(a, s_1) = s_2$ (the back condition for an unnamed action).

If there exists a bisimulation between two structures $\mathfrak{A}$ and $\mathfrak{B}$, then we say that $\mathfrak{A}$ and $\mathfrak{B}$ are *bisimilar*, and we denote it by $\mathfrak{A} \underline{\leftrightarrow} \mathfrak{B}$.

2.4.3 Finite Canonical Model of the Agent's Knowledge

We present a theorem that, given a basic action theory $\mathcal{D}$ in a language of the propositional situation calculus, the filtration $\mathfrak{F}^{\mathcal{D}}$ of the set of models of $\mathcal{D}$ is bisimilar to each model of $\mathcal{D}$. Thus, the filtration $\mathfrak{F}^{\mathcal{D}}$ is computationally equivalent to every model of $\mathcal{D}$, and hence, $\mathfrak{F}^{\mathcal{D}}$ is a finite canonical model of the agent's knowledge that $\mathcal{D}$ denotes.

Theorem 2.4.3. *Given a basic action theory $\mathcal{D}$ in a language of the propositional situation calculus, the filtration of the set of models of $\mathcal{D}$ is bisimilar to each model of $\mathcal{D}$.*

Proof. See Appendix A. □

The models of $\mathcal{D}$ are infinite tree-like structures, because of the foundational axioms [40]. Infinite structures are not desirable from the computational point of view. Through the filtration operation on $\mathcal{D}$, we extract the essence of the set of infinite models of $\mathcal{D}$ into a finite canonical structure $\mathfrak{F}^{\mathcal{D}}$.

Chapter 3

Query Language

In Chapter 2, we have presented the propositional situation calculus, our representation language. We have also described how to formalize the agent's knowledge as a set of sentences in the representation language and construct a finite canonical model of the agent's knowledge from the set of sentences. In this chapter, we present QL', our query language, and show its expressiveness.

Our query language QL' is a fragment of ID-logic. ID-logic is the extension of first-order logic with inductive definitions, and it provides a uniform and succinct way of representing various forms of inductive definitions that occur in mathematics and common-sense reasoning. The expressiveness of QL' surpasses that of the first-order logic. For example, we cannot express transitive closure in the first-order logic, but we can do so in QL'. Moreover, we can express in QL' *extended properties*, properties that account for non-determinism and possible failures of actions in a dynamic system. For example, we can express in QL' that the agent will *eventually* finish its given task despite possible interruptions.

We shall first present ID-logic, the extension of first-order logic with inductive definitions. We shall then present two fragments of ID-logic, QL and QL', where QL' is the extension of QL with limited quantification over actions. Moreover, we shall describe how to express queries as QL' sentences. Secondly, we shall present the full μ-calculus, a specification language well-known in the model-checking community, and its alternation-free fragment. Finally, we shall prove that QL is equally as expressive as the alternation-free μ-calculus. Then, by extending the proof of the expressiveness of QL, we shall prove that QL' is equally as expressive as QL with respect to a certain set of structures.

3.1 Query Language QL'

In this section, we present the syntax and semantics of ID-logic, the extension of first-order logic with inductive definitions, and then, QL and QL' as fragments of ID-logic. Moreover, we demonstrate by examples how to state queries in QL'.

3.1.1 ID-logic

We introduce a new binary connective $\leftarrow$ and call it the *definitional implication.* A *definition* Δ is a set of rules in the following form:

$$\forall \bar{x}[X(\bar{t}) \leftarrow \varphi],$$

where $\bar{x}$ is a tuple of variables, X is a predicate symbol (i.e., a predicate constant or variable) of an arity r, $\bar{t}$ is a tuple of terms of length r, and φ is a first-order formula that may contain free first-order or second-order variables.

The definitional implication $\leftarrow$ must be distinguished from material implication $\supset$. A rule $\forall \bar{x}[X(\bar{t}) \leftarrow \varphi]$ in a definition does not correspond to the disjunction $\forall \bar{x}[X(\bar{t}) \vee \neg\varphi]$, although it implies it. Intuitively, definitional implication should be understood as the "if" found in rules in inductive definitions (e.g. Definition 3.1.3 consists of 5 such rules). Another important difference is that, unlike $\forall \bar{x}[X(\bar{t}) \vee \neg\varphi]$, $\forall \bar{x}[X(\bar{t}) \leftarrow \varphi]$ does not have a truth value if it is considered by itself.

In front of each rule, we allow only a universal quantifier. In the following rule $\forall \bar{x}[X(\bar{t}) \leftarrow \varphi]$, we call $X(\bar{t})$ and φ the *head* and the *body*, respectively, of the rule. A *defined symbol* of a definition Δ is a predicate symbol that occurs in the head of at least one rule of Δ; we call the other symbols *open symbols* of Δ.

Suppose that τ is a vocabulary that includes all the free symbols in a definition Δ. We denote the set of defined symbols of Δ by $\tau^{\mathrm{d}}_{\Delta}$ and the set of open symbols of Δ by $\tau^{\mathrm{o}}_{\Delta}$. The sets $\tau^{\mathrm{d}}_{\Delta}$ and $\tau^{\mathrm{o}}_{\Delta}$ form a partition of τ; that is, $\tau^{\mathrm{d}}_{\Delta} \cup \tau^{\mathrm{o}}_{\Delta} = \tau$ and $\tau^{\mathrm{d}}_{\Delta} \cap \tau^{\mathrm{o}}_{\Delta} = \emptyset$.

Syntax

We define *well-formed formulae* of ID-logic by the following (monotone) induction:

1. If X is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms, then $X(t_1, \ldots, t_n)$ is a formula.

2. If Δ is a definition, then Δ is a formula.

3. If ϕ and ψ are formulae, then so is $(\phi \wedge \psi)$.

4. If ϕ is a formula, then so is $(\neg\phi)$.

5. If ϕ is a formula, then $\exists\sigma\ \phi$ is a formula where the symbol σ can be either first-order or second-order.

A formula ϕ is an *ID-logic-formula over a vocabulary* τ if the set of free symbols in ϕ is a subset of τ. It is a FO(ID)$[\tau]$-formula if it does not contain any second-order quantifiers; otherwise, it is a SO(ID)$[\tau]$-formula. Moreover, $\phi \vee \psi$, $\phi \supset \psi$, and $\phi \equiv \psi$ are the abbreviations for $\neg(\neg\phi \wedge \neg\psi)$, $\neg\phi \vee \psi$, and $(\phi \supset \psi) \wedge (\psi \supset \phi)$, respectively.

Example 3.1.1. Suppose that τ is a vocabulary that consists of a constant symbol 0 and a unary function symbol s, and that 0 and s denote zero and the successor function, respectively. The following SO(ID)$[\tau]$ formula expresses that there exists a set that contains zero and is the smallest set closed under the successor function. The formula is equivalent to the domain closure axiom for situations (in Subsection 2.1.2):

$$\exists N \left[\begin{array}{ll} \forall x[N(x) \leftarrow x = 0], & \\ \forall x[N(s(x)) \leftarrow N(x)] & \wedge \forall x N(x) \end{array} \right].$$

The formula in Example 3.1.1 contains an existential quantifier over the second-order variable N. We can avoid second-order existential quantification by *skolemization*, a way of replacing with constants variables that are bound by existential quantifiers but are outside the scope of universal quantifiers. We can simply replace N with a predicate constant symbol in Example 3.1.1.

Semantics

The semantics of the ID-logic is the extension of classical logic semantics with the well-founded semantics from logic programming [46, 7, 14].

We shall define the well-founded model of a definition Δ that extends a $\tau^{\mathrm{o}}_{\Delta}$-structure $\mathfrak{A}$. For each defined symbol X of Δ, we construct the following formula $\varphi_X(\bar{x})$: given that $\forall\bar{y}_1[X(\bar{t}_1) \leftarrow \varphi_1], \ldots, \forall\bar{y}_m[X(\bar{t}_m) \leftarrow \varphi_m]$ are the rules of Δ with X in the head,

$$\varphi_X(\bar{x}) := \exists\bar{y}_1[\bar{x} = \bar{t}_1 \wedge \varphi_1] \vee \cdots \vee \exists\bar{y}_m[\bar{x} = \bar{t}_m \wedge \varphi_m],$$

where $\bar{x}$ is a tuple of new variables.

For every defined symbol Y that occurs in $\varphi_1, \ldots, \varphi_m$, we introduce a new predicate symbol Y' of the same arity as that of Y. We then construct a formula $\varphi'_X(\bar{x})$ from $\varphi_X(\bar{x})$ by substituting each negative occurrence of Y with Y'.

Suppose that $\tau = \tau^{\mathrm{o}}_{\Delta} \cup \tau^{\mathrm{d}}_{\Delta}$. For any pair $(\mathfrak{B}, \mathfrak{C})$ of τ-structures that extend $\mathfrak{A}$, we define a τ-structure $\mathfrak{B}_{\mathfrak{C}}$, the extension of $\mathfrak{A}$ that interprets each defined symbol X of Δ as the value of X in $\mathfrak{B}$ (i.e., $X^{\mathfrak{B}}$) and each new symbol X' as the value of X in $\mathfrak{C}$ (i.e., $X^{\mathfrak{C}}$). We construct the well-founded model of Δ that extends $\mathfrak{A}$ via the operator T_{Δ}. The operator T_{Δ} maps pairs $(\mathfrak{B}, \mathfrak{C})$ of τ-structures that extend $\mathfrak{A}$ to a τ-structure $\mathfrak{A}'$ that also extends $\mathfrak{A}$, such that, for each defined symbol X of Δ,

$$X^{\mathfrak{A}'} := \{\bar{a} | \mathfrak{B}_{\mathfrak{C}}(\bar{x} : \bar{a}) \models \varphi'_X(\bar{x})\}$$

where $\mathfrak{B}_{\mathfrak{C}}(\bar{x} : \bar{a})$ denotes that $\mathfrak{B}_{\mathfrak{C}}$ replaces each occurrence of the variable in $\bar{x}$ with the corresponding value in $\bar{a}$. In other words, the operator T_{Δ} evaluates positive occurrences of defined symbols by $\mathfrak{B}$ and negative occurrences of defined symbols by $\mathfrak{C}$.

In the lattice of τ-structures that extend $\mathfrak{A}$, the operator T_{Δ} is monotone in its first argument and anti-monotone in its second argument. We hence define the *stable operator* ST_{Δ}[1] as follows:

$$ST_{\Delta}(\mathfrak{C}) := \mathit{lfp}(\mathsf{T}_{\Delta}(\cdot, \mathfrak{C})).$$

This stable operator is anti-monotone, and hence its square is monotone and has least and greatest fixed-points. We denote $\mathit{lfp}(ST^2_{\Delta})$ and $\mathit{gfp}(ST^2_{\Delta})$ as $\mathfrak{A}^{\Delta\downarrow}$ and $\mathfrak{A}^{\Delta\uparrow}$, respectively.

Definition 3.1.2. A definition Δ is *total* in a $\tau^{\mathrm{o}}_{\Delta}$-structure $\mathfrak{A}$ if $\mathfrak{A}^{\Delta\downarrow} = \mathfrak{A}^{\Delta\uparrow}$. If Δ is total, then we call $\mathfrak{A}^{\Delta\downarrow}$ (or $\mathfrak{A}^{\Delta\uparrow}$) the Δ-*extension of* $\mathfrak{A}$ and abbreviate it as $\mathfrak{A}^{\Delta}$. Generally, Δ is total in a structure $\mathfrak{B}$ that interprets a subset of $\tau^{\mathrm{o}}_{\Delta}$ if Δ is total in each $\tau^{\mathrm{o}}_{\Delta}$-structure that extends $\mathfrak{B}$.

The purpose of a definition is to *define* its defined symbols, and hence we consider only total definitions.

Definition 3.1.3. Suppose that ϕ is a formula of ID-logic and $\mathfrak{A}$ is any structure that interprets all the free symbols of ϕ. We define $\mathfrak{A} \models \phi$ (in words, ϕ *is true in* $\mathfrak{A}$, or $\mathfrak{A}$ *satisfies* ϕ, or $\mathfrak{A}$ is a *model* of ϕ) by the following induction:

[1] This operator is often called the Gelfond-Lifschitz operator as introduced in [16].

1. $\mathfrak{A} \models X(t_1, .., t_n)$ iff $({t_1}^{\mathfrak{A}}, .., {t_n}^{\mathfrak{A}}) \in X^{\mathfrak{A}}$.
2. $\mathfrak{A} \models \Delta$ iff $\mathfrak{A} = (\mathfrak{A}|^{\tau^{\circ}_{\Delta}})^{\Delta\downarrow} = (\mathfrak{A}|^{\tau^{\circ}_{\Delta}})^{\Delta\uparrow}$ where $\mathfrak{A}|^{\tau^{\circ}_{\Delta}}$ denotes the restriction of $\mathfrak{A}$ to τ°_{Δ}.
3. $\mathfrak{A} \models \psi_1 \wedge \psi_2$ iff $\mathfrak{A} \models \psi_1$ and $\mathfrak{A} \models \psi_2$.
4. $\mathfrak{A} \models \neg\psi$ iff $\mathfrak{A} \not\models \psi$.
5. $\mathfrak{A} \models \exists\sigma\psi$ iff, for some value v of σ in the domain of $\mathfrak{A}$, $\mathfrak{A}(\sigma : v) \models \psi$.

Given a vocabulary τ and a set T of sentences in the τ-language of ID-logic, we say that a τ-structure $\mathfrak{A}$ satisfies T if $\mathfrak{A}$ satisfies each sentence ϕ in T, and we denote it by $\models_{\mathfrak{A}} T$.

The inductive definition in the preceding definition is a prototypical example of a non-monotone inductive definition, specifically a definition over a well-founded poset. It is the set of ID-logic formulae, the element of which the sub-formula relation orders. It contains non-monotone recursion in rule 4. It is also an example of a sort of induction that we can formalize in ID-logic.

3.1.2 Fragments QL and QL'

We shall present two fragments of ID-logic: QL and QL'. We transported the syntactic characteristics of Modal Datalog, a fragment of Datalog LITE [20], from the context of database theory to that of mathematical logic, and encapsulated them in QL. As we shall prove in Section 3.3, QL is equally as expressive as the alternation-free μ-calculus, just as Modal Datalog is. The other fragment of ID-logic, QL', is the extension of QL with limited quantification over actions. Since QL is a fragment of QL', QL' is more expressive than QL, and hence, since QL is equivalently as expressive as the alternation-free μ-calculus, QL' is more expressive than the alternation-free μ-calculus.[2] However, as we shall prove in Section 3.3, with respect to a certain set of structures, QL' is equally as expressive as the alternation-free μ-calculus.

We shall define the concept of stratification, which is integral to the syntax of QL and QL'.

[2]Whereas QL' allows quantification over unnamed actions, the alternation-free μ-calculus allows quantification over only named actions.

Definition 3.1.4 (Stratification). A series $\Delta_1, \Delta_2, \ldots, \Delta_n$ of definitions is a *stratification* of a definition Δ and, for each integer i from 1 to n, Δ_i is a *stratum* of Δ if the following conditions are satisfied:

- $\bigcup_{i=1}^{n} \Delta_i = \Delta$.
- Each defined symbol of Δ appears in only one of the definitions $\Delta_1, \Delta_2, \ldots, \Delta_n$.
- For each $i \in \{1, 2, \ldots, n\}$, if a predicate symbol appears negatively in the body of a rule in Δ_i, then the predicate symbol is either open or defined in Δ_j with $j < i$. We say that Δ_i is *higher* than Δ_j, and Δ_j is *lower* than Δ_i.

We say that a definition Δ is *stratifiable* if there exists a stratification of Δ. Moreover, the well-founded semantics coincides with the least fixed-point semantics for stratifiable definitions of ID-logic.

Well-formed formulae of QL and QL' are in the following schema: $\Delta \wedge X(S_0)$ where Δ is a definition of ID-logic and X is a defined symbol of Δ. The definition Δ is restricted in two ways: a) the open vocabulary τ_Δ^o of Δ should be a vocabulary of the propositional situation calculus 2.1.3; b) the rules of Δ should meet certain schemata.

Each rule of a definition of QL should meet one of the following schemata:

$$\forall s[H(s) \leftarrow \phi(s)],$$
$$\forall s[H(s) \leftarrow \phi(s) \wedge \exists s'[s' = do(act, s) \wedge \psi(s')]],$$
$$\forall s[H(s) \leftarrow \phi(s) \wedge \forall s'[s' = do(act, s) \supset \psi(s')]],$$

where $\phi(s)$ and $\psi(s')$ are quantifier-free first-order formulae in which s and s' are the only free variables appearing in $\phi(s)$ and $\psi(s')$, respectively. In the second and third schemata, the atomic formula $s' = do(act, s)$ works as a *guard*, a syntactic entity that captures the variables of the rule and relativizes the values which the variables can possibly take. We say that a rule in the second schema is *existentially guarded* because the guard $s' = do(act, s)$ is existentially quantified. Similarly, we say that a rule in the third schema is *universally guarded.*

Each rule of a definition of QL' should meet one of the schemata for QL plus the following schemata:

$$\forall s[H(s) \leftarrow \phi(s) \wedge \exists a \exists s'[s' = do(a, s) \wedge \psi(s')]],$$
$$\forall s[H(s) \leftarrow \phi(s) \wedge \forall a \forall s'[s' = do(a, s) \supset \psi(s')]],$$
$$\forall s[H(s) \leftarrow \phi(s) \wedge \exists a \exists s'[\chi(a) \wedge s' = do(a, s) \wedge \psi(s')]],$$
$$\forall s[H(s) \leftarrow \phi(s) \wedge \forall a \forall s'[\chi(a) \wedge s' = do(a, s) \supset \psi(s')]],$$

where $\phi(s)$ and $\psi(s')$ are quantifier-free first-order formulae in which s and s' of the situation sort are the only free variables appearing in $\phi(s)$ and $\psi(s')$, respectively, and $\chi(a)$ is a quantifier-free first-order formula in which a variable a of the action sort is the only free variable appearing in $\chi(a)$. In the preceding list of schemata, rules in the first and third schemata are existentially guarded, and rules in the second and fourth schemata are universally guarded. As we have said, QL' is the extension of QL with limited quantification over actions. This is reflected in the preceding, additional schemata for QL' that specify what form of quantification over actions we allow in QL'.

Given a vocabulary τ of the propositional situation calculus, the τ-language of QL' is the set of well-formed formulae of QL' with $\tau^{\circ}_{\Delta} = \tau$ for each definition of QL'.

Definitions of QL' in the Normal Form

To make readable the proofs in Appendix B and the algorithms in Chapter 4, we want to transform a definition of QL' to an equivalent definition in the *normal form* where each rule of the definition meets one of the following schemata:

$$\forall s[H(s) \leftarrow \phi'(s)]$$
$$\forall s[H(s) \leftarrow \exists s'[s' = do(act, s) \wedge X_{\psi}(s')]]$$
$$\forall s[H(s) \leftarrow \forall s'[s' = do(act, s) \supset X_{\psi}(s')]]$$
$$\forall s[H(s) \leftarrow \exists a \exists s'[s' = do(a, s) \wedge X_{\psi}(s')]]$$
$$\forall s[H(s) \leftarrow \forall a \forall s'[s' = do(a, s) \supset X_{\psi}(s')]]$$
$$\forall a[A_{\chi}(a) \leftarrow \chi'(a)]$$
$$\forall s[H(s) \leftarrow \exists a \exists s'[A_{\chi}(a) \wedge s' = do(a, s) \wedge X_{\psi}(s')]]$$
$$\forall s[H(s) \leftarrow \forall a \forall s'[A_{\chi}(a) \wedge s' = do(a, s) \supset X_{\psi}(s')]]$$

where ϕ' and χ' are the conjunctions of literals and H, A_{χ}, and X_{ψ} are predicate symbols, which are not necessarily distinct.

Indeed, $\forall s[H(s) \leftarrow \forall s'[s' = do(act, s) \supset X(s')]]$ abbreviates the following rules:

$$\forall s[H(s) \leftarrow \neg H'(s)] \forall s[H'(s) \leftarrow \exists s'[s' = do(act, s) \wedge \neg X(s')]]$$

where H' is a new predicate symbol. Moreover, $\forall s[H(s) \leftarrow \forall a \forall s'[s' = do(a, s) \supset X_{\psi}(s')]]$ abbreviates the following rules:

$$\forall s[H_1(s) \leftarrow \exists a \exists s'[s' = do(a, s) \wedge \neg X_{\psi}(s')]]$$
$$\forall s[H_2(s) \leftarrow \neg H_1(s)]$$

whereas $\forall s[H(s) \leftarrow \forall a \forall s'[A_\chi(a) \wedge s' = do(a,s) \supset X_\psi(s')]]$ abbreviates the following rules:

$$\forall s[H_1(s) \leftarrow \exists a \exists s'[A_\chi(a) \wedge s' = do(a,s) \wedge \neg X_\psi(s')]]$$
$$\forall s[H_2(s) \leftarrow \neg H_1(s)].$$

Given a definition Δ of QL', we can transform Δ to an equivalent definition in the normal form, in time linear to the total number of connectives in Δ. We shall demonstrate the transformation by the following example.

Example 3.1.5. Suppose that τ is a vocabulary of the propositional situation calculus that includes one action constant symbol *act* and three unary relational fluent symbols A, B, and C. Given the following definition in the τ-language of QL'

$$\left\{ \forall s[H(s) \leftarrow A(s) \wedge \neg B(s) \wedge C(s) \wedge \exists s'[s' = do(act,s) \wedge (D(s') \supset E(s') \wedge \neg F(s'))]] \right\},$$

we can transform it to the following equivalent definition in the normal form:

$$\left\{ \begin{array}{l} \forall s[H_1(s) \leftarrow A(s) \wedge \neg B(s) \wedge C(s) \wedge H_2(s)] \\ \forall s[H_2(s) \leftarrow \exists s'[s' = do(act,s) \wedge H_3(s')]] \\ \forall s[H_3(s) \leftarrow \neg D(s)] \\ \forall s[H_3(s) \leftarrow E(s) \wedge \neg F(s)] \end{array} \right\}.$$

Moreover, given the following definition in the τ-language of QL':

$$\left\{ \forall s[H(s) \leftarrow \forall s'[s' = do(act,s) \supset (A(s') \supset B(s'))]] \right\},$$

we can transform it to the following equivalent definition in the normal form:

$$\left\{ \begin{array}{l} \forall s[H_1(s) \leftarrow \forall s'[s' = do(act,s) \supset H_2(s')] \\ \forall s[H_2(s) \leftarrow \neg A(s)] \\ \forall s[H_2(s) \leftarrow B(s)] \end{array} \right\}.$$

3.1.3 Queries as QL' Sentences

Suppose that we have a robot that cleans a floor by sweeping. Suppose that a vocabulary τ of the propositional situation calculus includes an action constant symbol *sweep_floor* and two unary relational fluent symbols *Open* and *Clean*. The robot cleans the floor by performing the action *sweep_floor*. The relational fluent *Open* is true if the floor is open to the public, and the relational fluent *Clean* is true if the floor is clean. The following examples show how we can formalize queries about the robot's environment as formulae in the τ-language of QL.

Example 3.1.6. "The robot will keep sweeping the floor until the floor becomes clean."

$$\left\{ \begin{array}{c} \forall s_1[H(s_1) \leftarrow Clean(s_1)], \\ \forall s_1[H(s_1) \leftarrow \forall s_2[s_2 = do(sweep\_floor, s_1) \supset H(s_2)]] \end{array} \right\} \wedge H(S_0).$$

The robot stops sweeping the floor as soon as the floor becomes clean; otherwise, it keeps sweeping the floor.

Example 3.1.7. "The floor will remain closed to the public until the robot finishes cleaning the floor."

$$\left\{ \begin{array}{c} \forall s_1[H(s_1) \leftarrow Clean(s_1)], \\ \forall s_1[H(s_1) \leftarrow \neg Open(s_1) \wedge \forall s_2[s_2 = do(sweep\_floor, s_1) \supset H(s_2)]] \end{array} \right\} \wedge H(S_0).$$

The floor is either open or closed to the public after the robot finishes cleaning the floor; otherwise, the floor remains closed to the public and the robot keeps sweeping the floor.

3.2 The μ-Calculus

In the preceding section, we have presented our query language QL'. In this section, we present the μ-calculus, a well-known specification language, and compare to it the expressiveness of QL'. We shall first present the syntax of the μ-calculus and the syntactic characterization of its alternation-free fragment. Secondly, we shall present the semantics of the μ-calculus via a labelled transition system, and then, via a structure over a vocabulary of the propositional situation calculus. Finally, we shall present examples of the alternation-free μ-calculus sentences.

3.2.1 Syntax

The μ-calculus is the extension of the propositional modal logic with the least fixed-point operator and the greatest fixed-point operator, μ and ν, respectively. It is a well-known specification language, the expressiveness of which subsumes that of other well-known specification languages such as LTL, CTL, and CTL*.

Given a set ρ of action symbols, a set σ of proposition symbols, and a set *Vars* of proposition variables, the following grammar defines the well-formed formulae of the (σ, ρ)-language of the μ-calculus:

$$\phi := P|Z|\neg\phi|\phi \wedge \phi|\langle act\rangle\phi|\mu X.\phi|\nu Z.\phi,$$

where act is a symbol in ρ, P is a symbol in σ, and Z is a symbol in $Vars$ that occurs only positively in ϕ of $\mu Z.\phi$ and $\nu Z.\phi$. Moreover, $\phi \vee \phi$ and $[act]\phi$ are the abbreviations for $\neg(\neg\phi \wedge \neg\psi)$ and $\neg\langle act\rangle\neg\phi$, respectively.

We shall eliminate formulae with greatest fixed-points by using the following logical equivalence:

$$\nu Z.\phi \iff \neg\mu Z'.\neg\phi(Z/\neg Z')$$

where $Z/\neg Z'$ denotes that we replace each occurrence of the propositional symbol Z in ϕ with the propositional symbol Z'.

The alternation-free fragment of the μ-calculus does not allow any nesting of least and greatest fixed-points; that is, in formulae $\mu Z.\phi$ and $\nu Z.\phi$, there are no such sub-formulae $\nu Y.\psi$ and $\mu Y.\psi$, respectively, that the outer fixed-point variable Z occurs inside ψ.

We illustrate the difference between alternation-free and alternating fixed-point formulae by examples.

Example 3.2.1. Let act_1 and act_2 be action symbols, P a propositional constant, and Z and Y propositional variables. Consider the following fixed-point formula:

$$\phi := \nu Z.(\mu Y.P \vee \langle act_1\rangle Y) \wedge [act_1]Z.$$

It specifies the property: "It is always possible that the proposition P will hold." We observe that $\mu Y.P \wedge \langle act\rangle Y$ is a sub-formula of ϕ. However, the outer fixed-point variable Z does not occur inside the sub-formula of $\mu Y.P \wedge \langle act\rangle Y$, namely, $P \wedge \langle act\rangle Y$, and hence, ϕ is a formula of the alternation-free μ-calculus. In contrast, consider the following fixed-point formula:

$$\phi := \mu Y.\nu Z.[act_1]Y \wedge [act_2]Z.$$

It specifies the property: "A process performs action act only finitely many times." We observe that $\nu Z.[act_1]Y$ is a sub-formula of ϕ, and that the outer fixed-point variable Y indeed occurs inside the sub-formula of $\nu Z.[act_1]Y$, namely, $[act_1]Y$. Therefore, ϕ is not a formula of the alternation-free μ-calculus.

3.2.2 Semantics

For a labelled transition system $\mathfrak{M} := (\mathrm{St}, \{R_a\}_{a\in\rho}, \mathcal{L})$ (Definition 2.2.1) and a function $v : Vars \to \wp(\mathrm{St})$, the following induction defines the set $\|\phi\|_v^{\mathfrak{M}}$ of states where a formula ϕ

is true:

$$\|P\|_v^{\mathfrak{M}} = \{st : P \in \mathcal{L}(st)\},$$
$$\|X\|_v^{\mathfrak{M}} = v(X),$$
$$\|\neg\psi\|_v^{\mathfrak{M}} = \mathrm{St} - \|\psi\|_v^{\mathfrak{M}},$$
$$\|\psi \wedge \chi\|_v^{\mathfrak{M}} = \|\psi\|_v^{\mathfrak{M}} \cap \|\chi\|_v^{\mathfrak{M}},$$
$$\|\langle act\rangle\psi\|_v^{\mathfrak{M}} = \{st : \exists st' \text{ with } \langle st, st'\rangle \in R_{act} \text{and } st' \in \|\psi\|_v^{\mathfrak{M}}\},$$
$$\|\mu X.\psi(X)\|_v^{\mathfrak{M}} = \bigcap\{A \subseteq \mathrm{St} : \|\psi\|_{v(\mathrm{St}/X)}^{\mathfrak{M}} \subseteq A\}.$$

If ϕ is a sentence, we omit v in the notation and write $\mathfrak{M}, st \models \phi$, instead of $st \in \|\phi\|^{\mathfrak{M}}$.

Semantics—The Structural Version

We need to address one problem before we compare the expressiveness of QL' to that of the alternation-free μ-calculus. On one hand, structures over a vocabulary of the propositional situation calculus define the semantics of QL', and, on the other hand, labelled transition systems define the semantics of the μ-calculus.

We overcome this problem by defining the semantics of the μ-calculus via structures over a vocabulary of the propositional situation calculus as follows:

$$\|F\|_v^{\mathfrak{A}} = F^{\mathfrak{A}},$$
$$\|X\|_v^{\mathfrak{A}} = v(X),$$
$$\|\neg\psi\|_v^{\mathfrak{A}} = \mathrm{Sit}^{\mathfrak{A}} \backslash \|\psi\|_v^{\mathfrak{A}},$$
$$\|\psi \wedge \chi\|_v^{\mathfrak{A}} = \|\psi\|_v^{\mathfrak{A}} \cap \|\chi\|_v^{\mathfrak{A}},$$
$$\|\langle act\rangle\psi\|_v^{\mathfrak{A}} = \{s_1 \mid \exists s_2 \text{ with } s_2 = do^{\mathfrak{A}}(act^{\mathfrak{A}}, s_1) \text{ and } s_2 \in \|\psi\|_v^{\mathfrak{A}}\},$$
$$\|\mu X.\psi(X)\|_v^{\mathfrak{A}} = \bigcap\{A \subseteq \mathrm{Sit}^{\mathfrak{A}} \mid \|\psi\|_{v(A/X)}^{\mathfrak{A}} \subseteq A\}.$$

3.2.3 Examples of Alternation-Free μ-Calculus Sentences

Given a set ρ of an action constant symbol *sweep_floor* and a set σ of two proposition symbols *Open* and *Clean*, the following examples show how we can formalize the queries about the robot's environment in Examples 3.1.6 and 3.1.7 as formulae in the (ρ, σ)-language of the alternation-free μ-calculus.

Example 3.2.2. "The robot will keep sweeping the floor until the floor becomes clean."

$$\mu Z.[Clean \vee [sweep\_floor]Z].$$

Example 3.2.3. "The floor will remain closed until the robot cleans the floor."

$$\mu Z.[Clean \vee (Closed \wedge [sweep\_floor]Z)].$$

We can illustrate the trade-off between succinctness and readability by comparing the examples of the alternation-free μ-calculus formulae and those of QL' formulae. A formula of the alternation-free μ-calculus tends to be shorter than a formula of QL' that describes the same phenomenon. However, The formula of the alternation-free μ-calculus is not as easy to decode as the formula of QL' where inductive definitions describe causality relations explicitly.

3.3 Expressiveness of QL'

In the preceding section, we have presented the μ-calculus and its alternation-free fragment. In this section, we show that QL is equivalently as expressive as the alternation-free μ-calculus. Moreover, we show that QL', the extension of QL and our query language, is equivalently as expressive as QL with respect to a certain set of structures.

We have informally defined a logic's expressive equivalence in Chapter 1. We define expressiveness in a concrete way.

Definition 3.3.1. A logic L_1 is *less expressive* than a logic L_2 if every set of structures definable in a language of L_1 is definable in a language of L_2. Moreover, L_1 is *equivalently as expressive* as L_2 if L_1 is less expressive than L_1 and L_2 is less expressive than L_1.

We shall let $L_1 \leq L_2$ denote that L_1 is less expressive than L_2, and $L_1 = L_2$ that L_1 is equivalently as expressive as L_2.

Moreover, we present expressive equivalence in a narrower scope.

Definition 3.3.2. Given a set K of structures, a logic L_1 is *less expressive* than a logic L_2 *with respect to* K if every set of structures that is definable in a language of L_1 and is a subset of K is definable in a language of L_2. Moreover, L_1 is *equivalently as expressive* as L_2 *with respect to* K if L_1 is less expressive than L_1 with respect to K and L_2 is less expressive than L_1 with respect to K.

We shall let $L_1 \leq_K L_2$ denote that L_1 is less expressive than L_2 with respect to K, and $L_1 =_K L_2$ that L_1 is equivalently as expressive as L_2 with respect to K.

We consider Definition 3.3.1 as the general case of Definition 3.3.2. In Definition 3.3.1, we measure a logic's expressiveness with respect to the universe of structures whereas, in Definition 3.3.2, we do so with respect to a certain subset of the universe.

We present a theorem about the expressiveness of QL.

Theorem 3.3.3. *QL is equivalently as expressive as the alternation-free μ-calculus.*

Proof. See Appendix B. □

The μ-calculus is *invariant for bisimulation* [27]; that is, a sentence of the μ-calculus cannot distinguish bisimilar structures. Because QL is equivalently as expressive as the alternation-free μ-calculus by Theorem 3.3.3, QL' is invariant for bisimulation.

Corollary 3.3.4. *QL is invariant for bisimulation.*

We present a theorem about the expressiveness of QL', and, by using Theorem 3.3.3, prove the theorem.

Theorem 3.3.5. *Given a basic action theory $\mathcal{D}$ in a language of the propositional situation calculus, QL' is equivalently as expressive as QL with respect to the set of structures that are bisimilar to $\mathfrak{F}^{\mathcal{D}}$.*

Proof. See Appendix B. □

Corollary 3.3.6. *Given a basic action theory $\mathcal{D}$ in a language of the propositional situation calculus, QL' is invariant for bisimulation with respect to the set of structures that are bisimilar to $\mathfrak{F}^{\mathcal{D}}$.*

Chapter 4

Reasoning Algorithm

We have presented our representation language, the propositional situation calculus, in Chapter 2, and our query language, QL', in Chapter 3. In this chapter, we shall show that, if formalized in the propositional situation calculus and QL', the problem of logical entailment is reducible to the problem of model-checking. We shall state this result as the *reducibility* theorem. Moreover, we shall present our reasoning algorithm *AlgoR*, the correctness of which follows from the reducibility theorem.

We shall first prove the reducibility theorem by using the results of Chapters 2 and 3. We shall then present the reasoning algorithm *AlgoR* and prove that, given a knowledge base $\mathcal{D}$ in our representation language and a query ϕ in our query language, *AlgoR* answers whether $\mathcal{D}$ logically entails ϕ in time linear with respect to both the size of $\mathcal{D}$ and the size of ϕ.

4.1 Reducibility Theorem

In this section, we shall show that, if formalized in the propositional situation calculus and QL', the problem of logical entailment is reducible to the problem of model-checking.

Theorem 4.1.1 (Reducibility Theorem). *Given a vocabulary τ of the propositional situation calculus, a basic action theory $\mathcal{D}$ in the τ-language of the propositional situation calculus, and a query ϕ in the τ-language of QL', if $\mathcal{D}$ is satisfiable, then*

$$\mathcal{D} \models \phi \iff \models_{\mathfrak{F}^{\mathcal{D}}} \phi$$

where $\mathfrak{F}^{\mathcal{D}}$ is the filtration of the set of models of $\mathcal{D}$.

Proof. Suppose that $\mathcal{D}$ is satisfiable. To prove $\mathcal{D} \models \phi \iff \models_{\mathfrak{F}^{\mathcal{D}}} \phi$, we shall first prove that

$$\mathcal{D} \models \phi \implies \models_{\mathfrak{F}^{\mathcal{D}}} \phi,$$

and then, that

$$\mathcal{D} \models \phi \impliedby \models_{\mathfrak{F}^{\mathcal{D}}} \phi.$$

($\Longrightarrow$) Suppose that $\mathcal{D}$ logically entails ϕ (i.e., $\mathcal{D} \models \phi$) but $\mathfrak{F}^{\mathcal{D}}$ does not satisfy ϕ (i.e., $\not\models_{\mathfrak{F}^{\mathcal{D}}} \phi$). Since $\mathcal{D}$ is satisfiable, there must exist at least one model of $\mathcal{D}$. Now suppose that $\mathfrak{A}$ is a model of $\mathcal{D}$ (i.e., $\models_{\mathfrak{A}} \mathcal{D}$). By Theorem 2.4.3, $\mathfrak{F}^{\mathcal{D}}$ is bisimilar to every model of $\mathcal{D}$, and hence, $\mathfrak{F}^{\mathcal{D}}$ is bisimilar to $\mathfrak{A}$ (i.e., $\mathfrak{F}^{\mathcal{D}} \underline{\leftrightarrow} \mathfrak{A}$). By Corollary 3.3.6, ϕ is invariant for bisimulation with respect to the set of structures that are bisimilar to $\mathfrak{F}^{\mathcal{D}}$, and hence, because $\mathfrak{F}^{\mathcal{D}} \underline{\leftrightarrow} \mathfrak{A}$,

$$\models_{\mathfrak{F}^{\mathcal{D}}} \phi \iff \models_{\mathfrak{A}} \phi.$$

Since $\not\models_{\mathfrak{F}^{\mathcal{D}}} \phi$ and $\models_{\mathfrak{F}^{\mathcal{D}}} \phi \iff \models_{\mathfrak{A}} \phi$, we have $\not\models_{\mathfrak{A}} \phi$. However, this is a contradiction because we derived earlier that $\models_{\mathfrak{A}} \mathcal{D}$. Therefore, our initial assumption that $\mathcal{D} \models \phi$ and $\not\models_{\mathfrak{F}^{\mathcal{D}}} \phi$ is false, and hence, $\mathcal{D} \models \phi \implies \models_{\mathfrak{F}^{\mathcal{D}}} \phi$.
($\Longleftarrow$) Suppose that $\mathfrak{F}^{\mathcal{D}}$ satisfies ϕ (i.e., $\models_{\mathfrak{F}^{\mathcal{D}}} \phi$). By Corollary 3.3.6, ϕ is invariant for bisimulation with respect to the set of structures that are bisimilar to $\mathcal{D}$, and hence, because $\models_{\mathfrak{F}^{\mathcal{D}}} \phi$, every structure bisimilar to $\mathfrak{F}^{\mathcal{D}}$ satisfies ϕ. Since every structure bisimilar to $\mathfrak{F}^{\mathcal{D}}$ satisfies ϕ and every model of $\mathcal{D}$ is bisimilar to $\mathfrak{F}^{\mathcal{D}}$ by Theorem 2.4.3, every model of $\mathcal{D}$ satisfies ϕ. Furthermore, because $\mathcal{D}$ is satisfiable, there must exist at least one model of $\mathcal{D}$. Therefore, $\mathcal{D} \models \phi$, and hence, $\mathcal{D} \models \phi \impliedby \models_{\mathfrak{F}^{\mathcal{D}}} \phi$. □

To answer whether $\mathcal{D}$ logically entails ϕ, instead of model-checking ϕ and each (possibly infinite) structure in the (possibly infinite) set of dynamic systems that $\mathcal{D}$ defines, we model-check ϕ and one structure $\mathfrak{F}^{\mathcal{D}}$. Moreover, if τ is finite, then $\mathfrak{F}^{\mathcal{D}}$ is a finite structure.

4.2 Reasoning Algorithm *AlgoR*

In the preceding section, we have proved the reducibility theorem, which says that if formalized in our representation and query languages, the problem of logical entailment is reducible to that of model-checking. In this section, we shall present our reasoning algorithm *AlgoR*, the correctness of which follows from the reducibility theorem.

Given a vocabulary τ of the propositional situation calculus, a basic action theory $\mathcal{D}$ in the τ-language of the propositional situation calculus, and a query $\Delta \wedge H(S_0)$ of the τ-language of QL', *AlgoR* returns "yes" if $\mathcal{D} \models \Delta \wedge H(S_0)$; otherwise, it returns "no".

The reasoning algorithm *AlgoR* uses three other algorithms as sub-routines: *AlgoF*, *AlgoG*, and *AlgoIS*. First, via *AlgoF*, *AlgoR* constructs the filtration of the set of models of $\mathcal{D}$ and returns it in the array representation $\mathfrak{F}$ on a unit-cost Random Access Machine (RAM). The array representation stores the characteristic membership function of each relation in an array whose dimension equals the arity of the relation. Secondly, *AlgoR* expands $\mathfrak{F}$ over a set of situation constant symbols and normalizes the rules of Δ. This is to prepare for the application of *AlgoG*. Thirdly, via *AlgoG*, *AlgoR* computes relations that Δ defines: *AlgoR* applies *AlgoG* to each stratum of Δ from the lowest to the highest. Fourthly, via *AlgoIS*, *AlgoR* constructs a set A_{S_0} such that each element of A_{S_0} can possibly be the initial situation of the filtration of the set of models of $\mathcal{D}$. Finally, *AlgoR* answers whether $\mathcal{D} \models \Delta \wedge H(S_0)$ by comparing A_{S_0} and $H^{\mathfrak{F}^\Delta}$: if $A_{S_0} \subseteq H^{\mathfrak{F}^\Delta}$, *AlgoR* returns "yes"; otherwise, it returns "no".

In the following pseudocode, we shall let n denote the number of unary fluent relation symbols in τ, and $sit_1, sit_2, \ldots, sit_{2^n}$ situation constant symbols. Moreover, we shall view the situation domain $\mathrm{Sit}^{\mathfrak{F}}$, the vector space $\{0,1\}^n$, as a lexicographically (totally) ordered set. We shall denote the smallest element in $\mathrm{Sit}^{\mathfrak{F}}$ as s_1, the second smallest element in $\mathrm{Sit}^{\mathfrak{F}}$ as s_2, and so on. Furthermore, we shall assume that a series $\Delta_1, \Delta_2, \ldots, \Delta_l$ is a stratification of Δ.

AlgoR$(\tau, \mathcal{D}, \Delta \wedge H(S_0))$
  $\mathfrak{F} :=$ *AlgoF*$(\tau, \mathcal{D})$
  Expand $\mathfrak{F}$ over $\{sit_1, sit_2, \ldots, sit_{2^n}\}$ as follows:
    for each integer i from 1 to 2^n
      $sit_i{}^{\mathfrak{F}} := s_i$
  Normalize the rules of Δ.
  for each integer i from 1 to l
    AlgoG$(\{sit_1, sit_2, \ldots, sit_{2^n}\}, \Delta_i, \mathfrak{F})$
  $A_{S_0} :=$ *AlgoIS*$(\mathcal{D}, \mathfrak{F})$
  if $A_{S_0} \subseteq H^{\mathfrak{F}^\Delta}$
    return "yes"
  else
    return "no"

To specify the complexity of each algorithm, we shall define the size of input data structures as follows:

- The size of τ, $|\tau|$, is the number of unary fluent relation symbols in τ plus the number of action constant symbols in τ.
- The size of $\mathcal{D}$, $|\mathcal{D}|$, is the sum of the lengths of axioms in $\mathcal{D}$.
- Given that the size of Δ, $|\Delta|$, is the sum of the lengths of the rules in Δ, the size of a query $\Delta \wedge H(S_0)$ is the size of Δ.
- Given that the size of a relation is the number of its tuples and the size of a domain is the number of its elements, the size of $\mathfrak{F}$, $|\mathfrak{F}|$, is the sum of the size of each domain of $\mathfrak{F}$ and the size of each relation of $\mathfrak{F}$. We convert the *do* function to an equivalent ternary relation.

In the rest of this section, we shall describe each sub-routine of *AlgoR* in detail and show its correctness and complexity. We shall then show the correctness and complexity of *AlgoR*.

4.2.1 Construct The Filtration—*AlgoF*

Given a vocabulary τ of the propositional situation calculus and a basic action theory $\mathcal{D}$ in the τ-language of the propositional situation calculus, the algorithm *AlgoF* constructs the filtration of the set of models of $\mathcal{D}$ and returns it in the array representation on a RAM.

In the following pseudocode, we shall let $\mathfrak{F}$ denote a $(\tau \backslash S_0)$-structure, m the number of action constant symbols in τ, and n the number of unary fluent relation symbols in τ. Moreover, *AlgoF* constructs the *do* function as a ternary relation R_{do} on a set $\mathrm{Act} \times \mathrm{Sit} \times \mathrm{Sit}$ such that, if $(c, u, u') \in R_{do}$, then $do^{\mathfrak{F}}(c, u) = u'$.

Theorem 4.2.1. *Given a vocabulary τ of the propositional situation calculus and a basic action theory $\mathcal{D}$ in the τ-language of the propositional situation calculus, the algorithm AlgoF returns the filtration of the set of models of $\mathcal{D}$ in time $O(2^{|\tau|} \times |\mathcal{D}|)$. The size of the filtration is $O(2^{|\tau|})$.*

Proof. Essentially, *AlgoF* constructs a $(\tau \backslash S_0)$-structure $\mathfrak{F}$ that satisfies the five conditions in Definition 2.3.1. Straight-forward, $\mathfrak{F}$ satisfies the first four conditions. We shall show that $\mathfrak{F}$, or rather, $do^{\mathfrak{F}}$ satisfies the fifth condition.

AlgoF$(\tau, \mathcal{D})$
  $\text{Sit}^{\mathfrak{F}} := \{0,1\}^n$
  $\text{Act}^{\mathfrak{F}} := \{c : c \in \{0,1\}^m, \|c\| = 0 \text{ or } \|c\| = 1\}$
  for each integer i from 1 n
    ${F_i}^{\mathfrak{F}} := \emptyset$
  for each integer i from 1 n
    for each element u in $\text{Sit}^{\mathfrak{F}}$
      if $(u)_i == 1$
        ${F_i}^{\mathfrak{F}} := {F_i}^{\mathfrak{F}} \cup \{u\}$
  for each integer i from 1 m
    for each element c in $\text{Act}^{\mathfrak{F}}$
      if $(c)_i == 1$
        ${act_i}^{\mathfrak{F}} := c$
  ${R_{do}}^{\mathfrak{F}} := \emptyset$
  for each $u \in \text{Sit}^{\mathfrak{F}}$
    ${R_{do}}^{\mathfrak{F}} := {do}^{\mathfrak{F}} \cup \{((0,0,\ldots,0), u, u)\}$
    for each integer i from 1 m
      $u' := (0,0,\ldots,0)$.
      for each integer j from 1 n
        if $a = act_i$ appears in $\gamma^+_{F_j}(a,s)$
          $(u')_j := 1$
        else if $(u)_j == 1$ AND $a = act_i$ does NOT appear in $\gamma^-_{F_j}(a,s)$
          $(u')_j := 1$
      ${R_{do}}^{\mathfrak{F}} := {R_{do}}^{\mathfrak{F}} \cup \{({act_i}^{\mathfrak{F}}, u, u')\}$
  Convert $\mathfrak{F}$ from the list representation to the array representation.
  return $\mathfrak{F}$

AlgoF constructs the ternary relation R_{do} such that $(c, u, u') \in R_{do}$ if and only if, for each integer i from 1 to n,

$$\left\{ \bigwedge_{j=1}^{m} L^c_j(x_a), \bigwedge_{j=1}^{n} L^u_j(x_s), \forall a \forall s \left[F_i(do(a,s)) \equiv \gamma^+_{F_i}(a,s) \wedge F_i(s) \vee \neg\gamma^-_{F_i}(a,s) \right] \right\} \models L^{u'}_i(do(x_a, x_s));$$

that is,

$$\left\{ \bigwedge_{j=1}^{m} L^c_j(x_a), \bigwedge_{j=1}^{n} L^u_j(x_s) \right\} \cup \mathcal{D}_{\text{ss}} \models \bigwedge_{j=1}^{n} L^{u'}_j(do(x_a, x_s)).$$

Thus, because $(c, u, u') \in R_{do}$ if and only if ${do}^{\mathfrak{F}}(c,u) = u'$, ${do}^{\mathfrak{F}}(c,u) = u'$ if and only if

$$\left\{ \bigwedge_{j=1}^{m} L^c_j(x_a), \bigwedge_{j=1}^{n} L^u_j(x_s) \right\} \cup \mathcal{D}_{\text{ss}} \models \bigwedge_{j=1}^{n} L^{u'}_j(do(x_a, x_s)),$$

and hence, $do^{\mathfrak{F}}$ satisfies the fifth condition.

AlgoF constructs the situation domain $\mathrm{Sit}^{\mathfrak{F}}$ in time $O(2^{|\tau|})$, and the action domain $\mathrm{Act}^{\mathfrak{F}}$ in time $O(|\tau|)$. Secondly, it constructs the fluent relations of $\mathfrak{F}$ in time $O(2^{|\tau|} \times |\tau|)$, that is, $O(2^{|\tau|})$, and the action constants in time $O(|\tau|^2)$. Thirdly, it constructs the ternary relation R_{do} (consequently, $do^{\mathfrak{F}}$) in time $O(2^{|\tau|} \times |\tau| \times |\mathcal{D}|)$, that is, $O(2^{|\tau|} \times |\mathcal{D}|)$. This time complexity contains the factor $|\mathcal{D}|$ because *AlgoF* traverses the successor state axioms for each element of $\mathrm{Sit}^{\mathfrak{F}} \times \mathrm{Act}^{\mathfrak{F}}$. Fourthly, it converts $\mathfrak{F}$ from the list representation to its array representation on a RAM in time linear with respect to $|\mathfrak{F}|$ [20], that is, $O(2^{|\tau|})$. (The size of $\mathrm{Sit}^{\mathfrak{F}}$ is $O(2^{|\tau|})$, the size of $\mathrm{Act}^{\mathfrak{F}}$ is $O(|\tau|)$, and the sum of the size of each relation including R_{do} is in $O(2^{|\tau|} \times |\tau|)$, that is, $O(2^{|\tau|})$.) Finally, the time complexity of *AlgoF* is $O(2^{|\tau|} \times |\mathcal{D}|)$, and the output size of *AlgoF* is $O(2^{|\tau|})$. □

4.2.2 Compute the Defined Relations—*AlgoG*

Suppose that τ is a vocabulary of the propositional situation calculus. Given a set of situation constant symbols $\{sit_1, \ldots, sit_{2^m}\}$, a definition Δ of the τ-language of QL' in the normal form, and a structure $\mathfrak{F}$ that interprets the symbols in $(\tau \backslash S_0) \cup \{sit_1, \ldots, sit_{2^m}\}$, by the *grounding* method, *AlgoG* computes relations that Δ defines. Informally, grounding (propositionalizing) is the process of eliminating variable symbols by replacing them with constant symbols. We have stated that, before applying *AlgoG*, *AlgoR* expands the vocabulary τ with a new set of situation constant symbols and associates each new constant symbol with an element of $\mathrm{Sit}^{\mathfrak{F}}$. This is to bring the elements of $\mathrm{Sit}^{\mathfrak{F}}$ into the syntax of our query language.

AlgoG grounds a rule of Δ by instantiating constant symbols for the variable symbols in the rule. After grounding, we can view the atomic sentences in the rule as a propositional symbol. We shall call a definition of QL' that consists of grounded rules a propositional definition of QL'.

AlgoG closely follows the model-checking algorithm for Datalog LITE [20] as it utilizes a Horn clause base (HCB), the Gelfond-Lifschitz transform algorithm, and the linear-time evaluation algorithm for Horn clauses [10, 26, 34]. A Horn clause is a propositional formulae, or equivalently, a propositionalized first-order formula that has at most one positive literal. A HCB is a data structure that represents Horn clauses in such a way that we can store, delete, and read literals in constant time.

The Gelfond-Lifschitz (GL) transform algorithm evaluates in linear time a logic program,

and also a stratifiable logic program, in which case it evaluates one stratum at a time. The GL transform algorithm is readily extendible to a propositional definition of QL' because the rules of a propositional definition of QL' are very similar to the rules of a logic program. Given a grounded rule r and a structure $\mathfrak{A}$, if r contains any false literals, the GL transform algorithm discards r completely; otherwise, it returns a Horn clause that is a Horn clause equivalent to r *minus* the literals true over $\mathfrak{A}$. Consequently, if the GL transform algorithm returns a Horn clause at all, then it should contain only defined symbols in the current stratum.

First, *AlgoG* grounds each rule of Δ, transforms it into a Horn clause via the GL transform algorithm, and stores the Horn clause in a HCB. Then, *AlgoG* evaluates the Horn clauses in the HCB via the linear-time evaluation algorithm for Horn clauses.

In the following pseudocode, we shall let *HCB* denote a HCB, *AlgoGL* the GL transform algorithm, and *AlgoEvalHorn* the linear-time evaluation algorithm for Horn clauses. Moreover, we shall let A denote the set of defined relations that *AlgoG* has computed so far. We shall view A as a persistent data structure: A remains after the current execution of *AlgoG* ends such that *AlgoG* can access A in the next execution and stores in A relations that it computes during each iteration. We shall assume that A is in the array representation on a RAM so that *AlgoG* can check the membership function of each tuple in a constant time. Moreover, we shall let $\mathfrak{F}^A$ denote the expansion of $\mathfrak{F}$ with the relations in A.

Theorem 4.2.2. *Suppose that τ is a vocabulary of the propositional situation calculus, and that Δ is a definition of the τ-language of QL' in the normal form that has no negated defined symbol. Given a set $\{sit_1, \ldots, sit_{2^m}\}$ of situation constant symbols, Δ, and a structure $\mathfrak{F}$ that interprets the symbols in $(\tau \backslash S_0) \cup \{sit_1, \ldots, sit_{2^m}\}$, AlgoG computes relations that Δ defines in time $O(|\Delta| \times |\mathfrak{F}|)$.*

Proof. The algorithm *AlgoG* is a straight-forward application of the grounding method to a definition of QL'. It instantiates each rule of Δ, transforms each instantiation into a Horn clause, and evaluates the Horn clause to compute relations that Δ defines.

To instantiate a guarded rule, *AlgoG* checks the tuples of the rule's guard. Given the array representation $\mathfrak{F}$ on a RAM, *AlgoG* can do so in time linear with respect to the size of the guard, which is $O(|\mathfrak{F}|)$. Thus, *AlgoG* instantiates all the rules of Δ in time $O(|\Delta| \times |\mathfrak{F}|)$. Moreover, for each instantiation, *AlgoG* calls *AlgoGL*, and *AlgoGL* runs in time linear with respect to the length of the instantiation. Thus, *AlgoG* transforms all the instantiations

$AlgoG(\{sit_1, \ldots, sit_{2^m}\}, \Delta, \mathfrak{F})$
  $HCB := \emptyset$
  for each rule r in Δ
    if r is a monadic rule $\forall s[H(s) \leftarrow \varphi(s)]$
      for each integer j from 1 to 2^m
        $HCB := HCB \cup AlgoGL(\{H(sit_j) \leftarrow \varphi(sit_j)\}, \mathfrak{F})$
    else if r is a existentially guarded rule
      if the guard is $s' = do(act, s)$
        $R := \{(u, u') : do^{\mathfrak{F}}(act^{\mathfrak{F}}, u) = u'\}$
      else if the guard is $s' = do(a, s)$
        $R := \{(u, u') : \exists c$ with $do^{\mathfrak{F}}(c, u) = u'\}$
      else if the guard is $\phi(a) \wedge s' = do(a, s)$
        $R := \{(u, u') : \exists c$ with $do^{\mathfrak{F}}(c, u) = u'$ and $\models_{\mathfrak{F}^A(a:c)} \phi(a)\}$
      for each tuple (u, u') in R
        $HCB := HCB \cup AlgoGL(\{H(sit_j) \leftarrow X(sit_k)\}, \mathfrak{F}^A)$ where ${sit_j}^{\mathfrak{F}} = u$, ${sit_k}^{\mathfrak{F}} = u'$, and H and X denote the head predicate of r and the predicate of the situation sort that appears in the body of r, respectively.
    else if r is a universally guarded rule
      for each integer j from 1 to 2^m
        if the guard is $s' = do(act, s)$
          $R := \{u : do^{\mathfrak{F}}(act^{\mathfrak{F}}, {sit_j}^{\mathfrak{F}}) = u\}$
        else if the guard is $s' = do(a, s)$
          $R := \{u : \exists c$ with $do^{\mathfrak{F}}(c, {sit_j}^{\mathfrak{F}}) = u\}$
        else if the guard is $\phi(a) \wedge s' = do(a, s)$
          $R := \{u : \exists c$ with $do^{\mathfrak{F}}(c, {sit_j}^{\mathfrak{F}}) = u$ and $\models_{\mathfrak{F}^A(a:c)} \phi(a)\}$
      $HCB := HCB \cup AlgoGL(\{H(sit_j) \leftarrow \bigwedge_{{sit_k}^{\mathfrak{F}} \in R} X(sit_k)\}, \mathfrak{F}^A)$ where H and X are the head predicate of r and the predicate of the situation sort that appears in the body of r, respectively.
  $A := A \cup AlgoEvalHorn(HCB)$

(i.e., grounded rules) in time $O(|\Delta| \times |\mathfrak{F}|)$.

Given a rule r of Δ, the number of grounded rules that are instantiations of r depends on whether r is a) monadic, b) existentially guarded, or c) universally guarded:

- If r is a monadic rule, then the number of grounded rules that are instantiations of r equals to the size of $\mathrm{Sit}^{\mathfrak{F}}$, which is $O(|\mathfrak{F}|)$. The length of each grounded rule differs from that of r only by a constant factor.

- If r is an existentially guarded rule, then the number of grounded rules that are instantiations of r equals to the size of the guard of r, which is $O(|\mathfrak{F}|)$. The length of each grounded rule differs from that of r only by a constant factor.

- If r is a universally guarded rule, then r gives only one instantiation, and hence, only

one grounded rule, but the length of the grounded rule differs from that of r at most by the size of the guard of r, which is in $O(|\mathfrak{F}|)$.

After *AlgoG* finishes grounding each rule in Δ, the sum of the lengths of the grounded rules, that is, the size of *HCB* is $O(|\Delta| \times |\mathfrak{F}|)$. *AlgoEvalHorn* evaluates the Horn clauses in *HCB* in time linear with respect to the size of *HCB*, which is $O(|\Delta| \times |\mathfrak{F}|)$.

Therefore, the total running time of *AlgoG* is $O(|\Delta| \times |\mathfrak{F}|)$. □

4.2.3 Find the Initial Situation(s)—*AlgoIS*

Suppose that τ is a vocabulary of the propositional situation calculus. Given a basic action theory $\mathcal{D}$ in the τ-language of the propositional situation calculus and a $(\tau \backslash S_0)$-structure $\mathfrak{F}$, the algorithm *AlgoIS* computes a subset A_{S_0} of $\mathrm{Sit}^{\mathfrak{F}}$ such that the elements of A_{S_0} are possible initial situations.

In the following pseudocode, we shall let m denote the number of unary fluent relation symbols in τ, and $\mathcal{D}_{S_0}$ the description of the initial situation in $\mathcal{D}$.

AlgoIS$(\mathcal{D}, \mathfrak{F})$
  $A_{S_0} := \emptyset$
  for each element u in $\mathrm{Sit}^{\mathfrak{F}}$
    $Flag := 1$
    $i := 1$
    while $Flag == 1$ OR $i == m$
      if $u \in F_i^{\mathfrak{F}}$ AND $\neg F_i(S_0)$ appears in $\mathcal{D}_{S_0}$
        $Flag := 0$
      else if $u \notin F_i^{\mathfrak{F}}$ AND $F_i(S_0)$ appears in $\mathcal{D}_{S_0}$
        $Flag := 0$
      else
        $i := i + 1$
    if $Flag == 1$
      $A_{S_0} := A_{S_0} \cup \{u\}$

Theorem 4.2.3. *Suppose that τ is a vocabulary of the propositional situation calculus. Given a basic action theory $\mathcal{D}$ in the τ-language of the propositional situation calculus and a structure $\mathfrak{F}$ that interprets the symbols in $\tau \backslash S_0$, AlgoIS computes a subset A_{S_0} of $\mathrm{Sit}^{\mathfrak{F}}$ such that the elements of A_{S_0} are possible initial situations of $\mathfrak{F}$ in time $O(|\mathfrak{F}| \times |\tau| \times |\mathcal{D}|)$. The size (i.e., cardinality) of A_{S_0} is $O(|\mathfrak{F}|)$.*

Proof. For each element u of $sit^{\mathfrak{F}}$, *AlgoIS* checks whether u satisfies the description of the initial situation $\mathcal{D}_{S_0}$ in time $O(|\tau| \times |\mathcal{D}|)$ because each time *AlgoIS* checks the membership

of u in each fluent symbol, it traverses $\mathcal{D}_{S_0}$. If u satisfies $\mathcal{D}_{S_0}$, then *AlgoIS* adds u to the set A_{S_0}. Thus, after checking each element of $sit^{\mathfrak{F}}$, A_{S_0} contains the elements of $sit^{\mathfrak{F}}$ that satisfy $\mathcal{D}_{S_0}$ and are possible initial situations. The total running time of *AlgoIS* is $O(|\mathfrak{F}| \times |\tau| \times |\mathcal{D}|)$. Since A_{S_0} is a subset of $\mathrm{Sit}^{\mathfrak{F}}$, the size of A_{S_0} is $O(|\mathfrak{F}|)$. □

4.2.4 Correctness and Complexity of *AlgoR*

We have showed the correctness and complexity of each sub-routine of *AlgoR*. We shall show the correctness and complexity of *AlgoR*, which follows from the correctness and complexity of each of its sub-routines and the reducibility theorem (i.e., Theorem 4.1.1).

Theorem 4.2.4. *Given a vocabulary τ of the propositional situation calculus, a basic action theory $\mathcal{D}$ in the τ-language of the propositional situation calculus, and a query $\Delta \wedge X(S_0)$ in the τ-language of QL', AlgoR answers whether $\mathcal{D} \models \Delta \wedge X(S_0)$ in time $O(2^{|\tau|} \times (|\mathcal{D}| + |\Delta|))$.*

Proof. The reasoning algorithm *AlgoR* uses three other algorithms as sub-routines: *AlgoF*, *AlgoG*, and *AlgoIS*. First, *AlgoF* constructs the filtration of the set of models of $\mathcal{D}$ and returns it in the array representation $\mathfrak{F}$ on a Random Access Machine (RAM). By Theorem 4.2.1, *AlgoF* does so in time $O(2^{|\tau|} \times |\mathcal{D}|)$ and the size of the filtration is $O(2^{|\tau|})$.

Secondly, given that τ contains n fluent relation symbols, *AlgoR* expands $\mathfrak{F}$ over a set of 2^n situation constant symbols, and then, normalizes the rules of Δ. This is done in $O(2^{|\tau|} + |\Delta|)$. Thirdly, *AlgoG* computes relations that Δ defines with the filtration: *AlgoG* evaluates each stratum of Δ from the lowest to the highest. By Theorem 4.2.2, *AlgoG* evaluates each stratum Δ_i of Δ in time $O(|\Delta_i| \times 2^{|\tau|})$, and hence, *AlgoR* evaluates the whole definition Δ in time $O(|\Delta| \times 2^{|\tau|})$.

Fourthly, *AlgoIS* constructs a set of elements that are possible initial situations in the filtration of the set of models of $\mathcal{D}$ and returns it in the array representation A_{S_0}. By Theorem 4.2.3, *AlgoIS* does so in time $O(2^{|\tau|} \times |\tau| \times |\mathcal{D}|)$, that is, $O(2^{|\tau|} \times |\mathcal{D}|)$, and the size of A_{S_0} is $O(2^{|\tau|})$. Finally, *AlgoR* answers whether $\mathcal{D} \models \Delta \wedge H(S_0)$ by comparing A_{S_0} and $H^{\mathfrak{F}^\Delta}$: if $A_{S_0} \subseteq H^{\mathfrak{F}^\Delta}$, *AlgoR* returns "yes"; otherwise, it returns "no". *AlgoR* does so by checking whether each element of A_{S_0} is an element of $H^{\mathfrak{F}^\Delta}$ in time $O(2^{|\tau|})$.

Therefore, the time complexity of *AlgoR* is

$$O(2^{|\tau|} \times |\mathcal{D}|) + O(2^{|\tau|} + |\Delta|) + O(2^{|\tau|} \times |\Delta|) + O(2^{|\tau|} \times |\mathcal{D}|) + O(2^{|\tau|});$$

that is, $O(2^{|\tau|} \times (|\mathcal{D}| + |\Delta|))$.

By Theorem 4.1.1, *AlgoR* answers the problem of logical entailment by reducing it to the problem of model-checking, which *AlgoR* answers by using *AlgoF*, *AlgoG*, and *AlgoIS*. Moreover, the correctness of *AlgoR* follows from the correctness of each of its sub-routines. □

If we restrict τ to be constant, then the time complexity of *AlgoR* will be $O(|\mathcal{D}| + |\Delta|)$. Such a problem is called *fixed-parameter tractable* [11]. Fixed-parameter tractability is a highly desirable feature, for it helps us to understand the computational nature of our problem and develop better parameterized algorithms for solving problem[21].

Chapter 5

Possible Applications

We suggest to apply the results of this thesis to the following two areas: planning under uncertainty and verification of robotics programs.

5.1 Planning under Uncertainty

Classical planning relies on several restrictive assumptions such as:

- Actions have only deterministic effects.
- The objective is to build a plan that leads to one of the goal states.
- Complete knowledge about the current state is available.

Consequently, plans in classical planning are sequences of actions. We want to relax the preceding assumptions and allow uncertainty such as non-determinism and possible failure of actions in our planning problem.

Determinism is a simplistic view in which the world evolves along one fully predictable path. It is a rather unrealistic assumption, for we know that not every event in the world is predictable. Non-determinism is a more realistic view. By non-determinism, we model a case where a system fails and plan for emergency and recovery. In some planning domains, we can model frequent, nominal cases by determinism and ignore non-nominal cases. In safety-critical and mission-critical planning domains, however, non-nominal cases are so important that we must model them at our planning time. Moreover, in other planning domains, nominal cases simply do not exist.

In non-determinism planning domains, we need to devise a plan that contains conditional behaviours and trial-and-error strategies. Such a plan results in many different executions, and hence, it is a little too strict to require that every possible execution of the plan meets the goal. We combine reachability goals (i.e., conditions about goal states) with extended goals (i.e., conditions on plan executions) to address a less strict notion of a satisfactory plan.

We can formalize and solve the problem of planning with extended goals via the results of this thesis as follows:

1. Axiomatize the planning domain (e.g., the action primitives, the description of the initial state) as a basic action theory $\mathcal{D}$ in the language of the propositional situation calculus.

2. Specify goal conditions (either reachability or extended) as a sentence ϕ in the language of QL'.

3. Solve the problem of logical entailment, $\mathcal{D} \models \phi$, by our reasoning algorithm *AlgoR*.

If *AlgoR* returns the negative answer, then there exists no plan that meets the specified goal conditions in the given planning domain. If *AlgoR* returns the affirmative answer, then we will generate a plan by applying a state-space search algorithm on the filtration $\mathfrak{F}^{\mathcal{D}}$ and progressing on the sentence ϕ of QL' to prune the search space.

Our reasoning algorithm *AlgoR* runs time only linear with respect to both the size of $\mathcal{D}$ and the size of ϕ. The time complexity of this planing algorithm depends on how fast we can generate a plan once receiving the affirmative answer from *AlgoR*. Moreover, *AlgoR* can deal with partial observability in the initial state; that is, it does not require complete knowledge about the initial state.

5.2 Verification of Robotics Programs

We want to prove the reliability of robotics programs when they are part of a safety-critical or mission-critical system. Especially when they employ concurrency, bugs are difficult to find just by testing (e.g., running several simulations of important scenarios) because bugs that concurrency induces tend to be non-reproducible. The verification of robotics programs is hence a topic worth exploring.

Golog [29] is a high-level programming language for robot control. It is based on the situation calculus. In Golog, the programmer can define primitive actions and fluents by writing action precondition axioms and successor state axioms. She can hence specify primitives according to the program domain she wants to model, regardless of the actual implementation of the system's primitive architecture. She can also choose the level of abstraction at which she wants to write her program. ConGolog [19] is the extension of Golog with constructs for concurrency.

The verification of Golog and ConGolog programs is quite feasible due to the foundation of Golog and ConGolog in the situation calculus. Liu [30] demonstrated the embedding of Hoare-style proof systems into the Golog context. Shapiro, Lesperance, and Levesque [44] developed a verification environment for ConGolog programs. However, the verification of Golog and ConGolog programs has not been largely explored beyond these results.

We can verify Golog and ConGolog programs by using the results of this thesis as follows: Given that a program Π consists of a theory of actions $\Pi_{\mathcal{D}}$ and a robot control Π_{ctrl}, and that $\Pi = \Pi_{\mathcal{D}} \cup \Pi_{\text{ctrl}}$ and $\Pi_{\mathcal{D}} \cap \Pi_{\text{ctrl}} = \emptyset$,

1. Transform $\Pi_{\mathcal{D}}$ into a set of domain-dependent axioms A (i.e., successor state axioms plus the description of the initial situation) in the language of the propositional situation calculus.

2. Transform Π_{ctrl} into a definition Δ_1 in the language of QL'.

3. Formalize the program specification for Π as a sentence $\Delta_2 \wedge X(S_0)$ in the language of QL' such that $\tau^{\text{d}}_{\Delta_1} \cap \tau^{\text{d}}_{\Delta_2} = \emptyset$.

4. Let $\mathcal{D}$ be a basic action theory in the language of the propositional situation calculus such that $\mathcal{D}_{\text{pss}} \cup \mathcal{D}_{\text{p}S_0} = A$. Solve the problem $\mathcal{D} \models (\Delta_1 \cup \Delta_2) \wedge X(S_0)$ of logical entailment via *AlgoR*.

If *AlgoR* returns the affirmative answer, then Π meets the program specification. If *AlgoR* returns the negative answer, then, as a counter-example, we will return a plan that we generate by applying a state-space search algorithm on the filtration $\mathfrak{F}^{\mathcal{D}}$ and progressing on the following sentence ϕ of QL':

$$\phi := (\Delta_1 \cup \Delta_2 \cup \{\forall s[X'(s) \leftarrow \neg X(s)]\}) \wedge X'(S_0)$$

where $X' \notin \tau^{\mathrm{d}}_{\Delta_1 \cup \Delta_2}$. We have already verified that $\mathcal{D} \not\models (\Delta_1 \cup \Delta_2) \wedge X(S_0)$, and hence,

$$\begin{aligned}
\mathcal{D} \models \neg((\Delta_1 \cup \Delta_2) \wedge X(S_0)) &\iff \mathcal{D} \models (\Delta_1 \cup \Delta_2) \wedge \neg X(S_0) \\
&\iff \mathcal{D} \models (\Delta_1 \cup \Delta_2 \cup \{\forall s[X'(s) \leftarrow \neg X(s)]\}) \wedge X'(S_0) \\
&\iff \mathcal{D} \models \phi.
\end{aligned}$$

Thus, we are ensured to find a plan that meets the goal condition ϕ. This plan generates executions of Π that do not meet the specification $\Delta_2 \wedge X(S_0)$.

Not every program in Golog or ConGolog is verifiable in the framework. Programs verifiable in the framework are those that we can convert into a basic action theory in the propositional situation calculus and a query in QL'. However, programs do not need to have the complete knowledge of the initial situation to be verifiable.

This framework for the verification of Golog and ConGolog programs is model-based and automated. Moreover, it is declarative, and hence, suitable for the verification of reactive programs, which result in infinite execution paths. Furthermore, the program specification ϕ together with the basic action theory $\mathcal{D}$ can serve as a system specification, which is an important component of a system documentation. A formally specified and verified system takes less resources to implement because we eliminate many errors in the designing phase. It is also easier to reuse because it comes with a documentation that clearly states what it is supposed to do.

Chapter 6

Conclusion

In this thesis, we rest our theoretical foundations on symbolic logic, and conduct the analysis of a knowledge-based agent through *logical entailment*: given a set T of sentences and a sentence ϕ in a language of symbolic logic, does every structure that satisfies each sentence in T satisfy the sentence ϕ (i.e., $T \models \phi$)? We formalize an agent's reasoning process as the following problem of logical entailment: given a knowledge base T in a representation language and a query ϕ in a query language, does T logically entail ϕ?

We choose the propositional situation calculus as our representation language, and QL' as our query language. We formalize the agent's knowledge as a set $\mathcal{D}$ of sentences in a language of the propositional situation calculus, a query as a sentence ϕ of a language of QL', and the agent's reasoning process as the problem of logical entailment (i.e., $\mathcal{D} \models \phi$). Moreover, we solve the problem of logical entailment by a model-theoretic approach: we construct from $\mathcal{D}$ a finite canonical model $\mathfrak{F}^{\mathcal{D}}$ of the agent's knowledge through the *filtration* operation, a model-theoretic operation of collapsing a possibly infinite structure into a quotient structure. We prove that $\mathcal{D}$ logically entails ϕ if and only if $\mathfrak{F}^{\mathcal{D}}$ satisfies ϕ; that is,

$$\mathcal{D} \models \phi \iff \models_{\mathfrak{F}^{\mathcal{D}}} \phi.$$

This results says that, if formalized in the propositional situation calculus and QL', the problem of logical entailment is reducible to the problem of *model-checking*: does a model $\mathfrak{A}$ of a computing system satisfy a formal specification ψ (i.e., $\models_{\mathfrak{A}} \psi$)? In other words, to answer whether $\mathcal{D}$ logically entailments ϕ, instead of model-checking ϕ and each (possibly infinite) structure in the (possibly infinite) set of models of $\mathcal{D}$, we model-check ϕ and $\mathfrak{F}^{\mathcal{D}}$. We call this result the *reducibility theorem*. Furthermore, we present a reasoning algorithm that

solves the problem of logical entailment by reducing it to the problem of model-checking. This algorithm is based on the linear-time model-checking algorithm for Datalog LITE [20], and runs in time linear with respect to both the size of $\mathcal{D}$ and the size of ϕ. Moreover, this algorithm does not assume that the agent has the complete knowledge of its initial environment.

This thesis makes two major contributions to the research of knowledge representation and reasoning. The first contribution is that this thesis formalizes a model-theoretic approach to reasoning about actions, which we can apply to representation and query languages more expressive than the propositional situation calculus and QL'. The second contribution is an expressive query language with a good complexity result, QL', which is more expressive than well-known specification languages such as LTL and CTL.

Future Work

In this thesis, we limit the vocabulary of the propositional situation calculus to include only unary relational symbols and constant symbols, so that a set of structures definable in the propositional situation calculus is bisimulation-closed; that is, if a structure is in the set, then every structure bisimilar to the structure is in the set. In other words, we choose to model dynamic systems at a certain level of abstraction, that is, into a bisimulation-closed set. We justify this limitation by the fact that QL', our query language, allows only queries about bisimulation-invariant properties of dynamic systems. We have stated in Chapter 2 that this level of abstraction is adequate when we study the computational behaviour of a dynamic system.

If we choose a query language more expressive than QL' to ask queries beyond bisimulation invariance, then we will need to choose a representation language more expressive than the propositional situation calculus, so that we can model dynamic systems at a lower level of abstraction (or at a higher level of granularity). Depending on the kind of queries we want to ask about dynamic systems, we change a level of abstraction (or granularity) in modelling the dynamic systems. As the next level of abstraction beyond bisimulation invariance, we suggest a set of structures that is closed under *guarded bisimulation* [1].

Guarded bisimulation is a straight-forward generalization of bisimulation. We can describe guarded bisimulation as the following guarded version of the infinite Ehrenfeucht-Fraïssé game. We have two players I and II and two sets of labelled pebbles. Let $\mathfrak{A}$ and $\mathfrak{B}$ be structures over the same vocabulary. Initially, neither $\mathfrak{A}$ nor $\mathfrak{B}$ carries pebbles. Player

I tries to show that $\mathfrak{A}$ is different from $\mathfrak{B}$, and Player II does otherwise. Players I and II mark elements of $\mathfrak{A}$ and $\mathfrak{B}$ by placing pebbles on the elements, such that the currently marked elements form *guarded sets* in $\mathfrak{A}$ and $\mathfrak{B}$. A subset X of the domain of a structure is a guarded set if, given $X = \{a_1, a_2, \ldots, a_r\}$, there exists some r-ary relation R in the structure such that $(a_1, a_2, \ldots, a_r) \in R$.

After each round of the game, the pebble placement in $\mathfrak{A}$ and $\mathfrak{B}$ induces a mapping $\bar{a} \mapsto \bar{b}$, such that an element a in the domain of $\mathfrak{A}$ is mapped to an element b in the domain of $\mathfrak{B}$ if and only if a and b are marked by pebbles with the same label. Player I makes moves in one structure, and Player II mimics Player I's moves by placing pebbles such that the new mapping $\bar{a} \mapsto \bar{b}$ is a partial isomorphism between $\mathfrak{A}$ and $\mathfrak{B}$. Consequently, the elements that Player II marks forms a guarded set. We say that Player II has a winning strategy if she can always respond to Player I's move. Moreover, Player II has a winning strategy if and only if there exists some guarded bisimulation between $\mathfrak{A}$ and $\mathfrak{B}$ [22].

To ask queries beyond bisimulation invariance, we need a query language more expressive than QL', so we extend QL' to the alternation-free fragment of the *guarded fixed-point logic* μGF. μGF is the fixed-point extension of the guarded fragment GF of the first-order logic. GF is the generalization of the modal fragment of the first-order logic[1], and retains nice properties of the modal logic such as decidability and finite model property. Moreover, μGF is characterized as the guarded-bisimulation-invariant fragment of the guarded second-order logic (GSO) [22]. GSO is the second-order logic with semantics that allows second-order quantifiers to range only over guarded sets.

We need a new query language that is guarded-bisimulation-invariant. One such language is a fragment of ID-logic that is equivalently as expressive as the alternation-free μGF. Moreover, we can extend the fragment of ID-logic with limited quantification over actions such that the extension is equivalently as expressive as the alternation-free μGF with respect to a set of structures closed under guarded bisimulation. (In this thesis, we have extended QL similarly to obtain QL'.) Our new query language allows vocabularies to include more than unary relational symbols and constant symbols. Furthermore, we can apply the grounding method to evaluate queries in our new query language to attain a good complexity result.

We need a new representation language to define a set of structures that is closed under

[1]The modal fragment of the first-order logic is the embedment of the modal logic into the first-order logic.

guarded bisimulation. To apply the model-theoretic approach that we have formalized in this thesis, we need to limit the vocabulary of our new representation language (and our new query language) so that we can construct a finite canonical model of a basic action theory that is axiomatized in the representation language. For example, we limit the vocabulary to include only a finite number of action and object symbols so that we can divide the elements of the (infinite) situation domain in a model of the basic action theory into a finite number of equivalence classes.

In this thesis, we have presented the propositional situation calculus, a fragment of the Reiter-style situation calculus as our representation language. It is not easy to describe ramifications (i.e., indirect effects of actions), in the Reiter-style situation calculus. To address the ramification problem, Denecker and Ternovska presented the *inductive situation calculus* [8]. The inductive situation calculus is a variant of the Reiter-style situation calculus (and also a fragment of ID-logic) where simultaneous inductive definitions define fluent symbols on the well-ordered set of situations. The solution to the ramification problem in the inductive situation calculus is among the most general solutions. We can alternatively obtain a representation language for the (guarded-)bisimulation-invariant level of abstract by identifying a (guarded-)bisimulation-invariant fragment of the inductive situation calculus.

Appendix A

Proofs of Chapter 2

We shall first prove two lemmas, and then, by using these lemmas, prove Theorem 2.4.3.

A.1 Lemmas for Theorem 2.4.3

Lemma A.1.1. *A structure over a vocabulary of the propositional situation calculus is bisimilar to the filtration of itself.*

Proof. A non-empty binary relation $\{(s, |s|) : s \in \mathrm{Sit}^{\mathfrak{A}}, |s| \in \mathrm{Sit}^{\mathfrak{A}^f}\}$ satisfies the conditions to be a bisimulation between $\mathfrak{A}$ and $\mathfrak{A}^f$, and hence, a structure over a vocabulary of the propositional situation calculus is bisimilar to the filtration of itself. □

Lemma A.1.2. *Given a basic action theory $\mathcal{D}$ in a language of the propositional situation calculus, the filtration of the set of models of $\mathcal{D}$ is isomorphic to the filtration of each model of $\mathcal{D}$.*

Proof. Suppose that τ is a vocabulary of the propositional situation calculus, and that $\mathcal{D}$ is a basic action theory in the τ-language of the propositional situation calculus. To prove that the filtration of the set of models of $\mathcal{D}$ is isomorphic to the filtration of each model of $\mathcal{D}$, we shall show that, for each model $\mathfrak{A}$ of $\mathcal{D}$, there exists a one-to-one and onto homomorphism from $\mathrm{Sit}^{\mathfrak{A}^f}$ to $\mathrm{Sit}^{\mathfrak{F}^{\mathcal{D}}}$ as well as a one-to-one and onto homomorphism from $\mathrm{Act}^{\mathfrak{A}^f}$ to $\mathrm{Act}^{\mathfrak{F}^{\mathcal{D}}}$.

Suppose that ρ and σ are the set of action constant symbols in τ and the set of unary relational fluent symbols in τ, respectively. We shall view ρ and σ as lexicographically (totally) ordered sets. We denote the smallest element in ρ as act_1, the second smallest

element in ρ as act_2, and so on. Similarly, we denote the smallest element in σ as F_1, the second smallest element in σ as F_2, and so on. Moreover, we denote the cardinality of ρ and the cardinality of σ by m and n, respectively.

For each model $\mathfrak{A}$ of $\mathcal{D}$, we shall define a function $g : \mathrm{Sit}^{\mathfrak{A}^f} \to \mathrm{Sit}^{\mathfrak{F}^{\mathcal{D}}}$ as well as a function $h : \mathrm{Act}^{\mathfrak{A}^f} \to \mathrm{Act}^{\mathfrak{F}^{\mathcal{D}}}$ as follows:

- For each element t of $\mathrm{Sit}^{\mathfrak{A}^f}$, $g(t) = u$ for some element u of $\mathrm{Sit}^{\mathfrak{F}^{\mathcal{D}}}$ where, for each integer i from 1 to n,
$$t \in {F_i}^{\mathfrak{A}^f} \iff u_i = 1.$$
- For each integer i from 1 to m, $h({act_i}^{\mathfrak{A}^f}) = c$ for some element c of $\mathrm{Act}^{\mathfrak{F}^{\mathcal{D}}}$ with $c_i = 1$.

Both g and h are one-to-one and onto. Moreover, to show that g and h are a homomorphism from $\mathrm{Sit}^{\mathfrak{A}^f}$ to $\mathrm{Sit}^{\mathfrak{F}^{\mathcal{D}}}$ and a homomorphism from $\mathrm{Act}^{\mathfrak{A}^f}$ to $\mathrm{Act}^{\mathfrak{F}^{\mathcal{D}}}$, respectively, we shall show that g and h satisfy the following conditions:

- For each element t of $\mathrm{Sit}^{\mathfrak{A}^f}$ and each integer i from 1 to n,
$$t \in {F_i}^{\mathfrak{A}^f} \iff g(t) \in {F_i}^{\mathfrak{F}^{\mathcal{D}}}.$$
- For each integer i from 1 to m,
$$h({act_i}^{\mathfrak{A}^f}) = {act_i}^{\mathfrak{F}^{\mathcal{D}}}.$$
- For each element (b, t) of $\mathrm{Sit}^{\mathfrak{A}^f} \times \mathrm{Act}^{\mathfrak{A}^f}$,
$$g(do^{\mathfrak{A}^f}(b, t)) = do^{\mathfrak{F}^{\mathcal{D}}}(h(b), g(t)).$$

Given an element t of $\mathrm{Sit}^{\mathfrak{A}^f}$, let $g(t) = u$ for some element u of $\mathrm{Sit}^{\mathfrak{F}^{\mathcal{D}}}$. By the definition of g, $t \in {F_i}^{\mathfrak{A}^f} \iff u_i = 1$ for each integer i from 1 to n. Moreover, by the definition of $\mathfrak{F}^{\mathcal{D}}$, $u \in {F_i}^{\mathfrak{F}^{\mathcal{D}}} \iff u_i = 1$ for each integer i from 1 to n. Thus, for each integer i from 1 to n, $t \in {F_i}^{\mathfrak{A}^f} \iff u \in {F_i}^{\mathfrak{F}^{\mathcal{D}}}$, and hence, because $u = g(t)$, $t \in {F_i}^{\mathfrak{A}^f} \iff g(t) \in {F_i}^{\mathfrak{F}^{\mathcal{D}}}$. We have proved that g satisfies the first condition.

Given an integer i between 1 and n, let $h({act_i}^{\mathfrak{A}^f}) = c$ for some element c of $\mathrm{Act}^{\mathfrak{F}^{\mathcal{D}}}$. By the definition of h, $c_i = 1$. Moreover, by the definition of $\mathfrak{F}^{\mathcal{D}}$, ${act_i}^{\mathfrak{F}^{\mathcal{D}}} = c$. Thus, because $h({act_i}^{\mathfrak{A}^f}) = c$, $h({act_i}^{\mathfrak{A}^f}) = {act_i}^{\mathfrak{F}^{\mathcal{D}}}$. We have proved that h satisfies the second condition.

Given an element (b,t) of $\mathrm{Act}^{\mathfrak{A}^f} \times \mathrm{Sit}^{\mathfrak{A}^f}$, let $g(do^{\mathfrak{A}^f}(b,t)) = u'$ for some element u' of $\mathrm{Sit}^{\mathfrak{F}^{\mathcal{D}}}$. Suppose that $do^{\mathfrak{A}^f}(b,t) = t'$ for some element t' of $\mathrm{Sit}^{\mathfrak{A}^f}$, and that $b = |a|$, $t = |s|$, and $t' = |s'|$ for some element (a,s,s') of $\mathrm{Act}^{\mathfrak{A}} \times \mathrm{Sit}^{\mathfrak{A}} \times \mathrm{Sit}^{\mathfrak{A}}$. Because $do^{\mathfrak{A}^f}(|a|,|s|) = |s'|$, by the definition of $\mathfrak{A}^f$, $do(a,s) = s$, and hence, for some action variable x_a and some situation variable x_s,

$$\left\{\bigwedge_{i=1}^{m} L_i^a(x_a), \bigwedge_{i=1}^{n} L_i^s(x_s)\right\} \cup \mathcal{D}_{\mathrm{ss}} \models \bigwedge_{i=1}^{n} L_i^{s'}(do(x_a,x_s)).$$

Suppose that $(\vec{a},\vec{s},\vec{s'})$ is an element of $\mathrm{Act}^{\mathfrak{F}^{\mathcal{D}}} \times \mathrm{Sit}^{\mathfrak{F}^{\mathcal{D}}} \times \mathrm{Sit}^{\mathfrak{F}^{\mathcal{D}}}$ such that, for each integer i from 1 to m, $(\vec{a})_i = 1 \iff act_i^{\mathfrak{A}} = a$; moreover, for each integer i from 1 to n, $(\vec{s})_i = 1 \iff s \in F_i^{\mathfrak{A}}$ and $(\vec{s'})_i = 1 \iff s' \in F_i^{\mathfrak{A}}$. Because $\bigwedge_{i=1}^m L_i^a(x_a) = \bigwedge_{i=1}^m L_i^{\vec{a}}(x_a)$, $\bigwedge_{i=1}^n L_i^s(x_s) = \bigwedge_{i=1}^n L_i^{\vec{s}}(x_s)$, and $\bigwedge_{i=1}^n L_i^{s'}(x_s) = \bigwedge_{i=1}^n L_i^{\vec{s'}}(x_s)$,

$$\left\{\bigwedge_{i=1}^{m} L_i^{\vec{a}}(x_a), \bigwedge_{i=1}^{n} L_i^{\vec{s}}(x_s)\right\} \cup \mathcal{D}_{\mathrm{ss}} \models \bigwedge_{i=1}^{n} L_i^{\vec{s'}}(do(x_a,x_s)),$$

and hence, by the definition of $\mathfrak{F}^{\mathcal{D}}$, $do^{\mathfrak{F}^{\mathcal{D}}}(\vec{a},\vec{s}) = \vec{s'}$. Moreover, by the definition of g, $g(|s|) = \vec{s}$ and $g(|s'|) = \vec{s'}$. Similarly, by the definition of h, $h(|a|) = \vec{a}$. Thus, $do^{\mathfrak{F}^{\mathcal{D}}}(h(|a|),g(|s|)) = g(|s'|)$, and hence, because $|a| = b$, $|s| = t$, $|s'| = t'$, and $t' = do^{\mathfrak{A}^f}(b,t)$, $do^{\mathfrak{F}^{\mathcal{D}}}(h(b),g(t)) = g(do^{\mathfrak{A}^f}(b,t))$. We have proved that h and g satisfy the third condition.

We have showed that, given a model $\mathfrak{A}$ of $\mathcal{D}$, g is a one-to-one and onto homomorphism from $\mathrm{Sit}^{\mathfrak{A}^f}$ to $\mathrm{Sit}^{\mathfrak{F}^{\mathcal{D}}}$, and that h is a one-to-one and onto homomorphism from $\mathrm{Act}^{\mathfrak{A}^f}$ to $\mathrm{Act}^{\mathfrak{F}^{\mathcal{D}}}$. Thus, we have showed that, for each model $\mathfrak{A}$ of $\mathcal{D}$, there exists a one-to-one and onto homomorphism from $\mathrm{Sit}^{\mathfrak{A}^f}$ to $\mathrm{Sit}^{\mathfrak{F}^{\mathcal{D}}}$ as well as a one-to-one and onto homomorphism from $\mathrm{Act}^{\mathfrak{A}^f}$ to $\mathrm{Act}^{\mathfrak{F}^{\mathcal{D}}}$. □

A.2 Proof of Theorem 2.4.3

Theorem 2.4.3 *Given a basic action theory $\mathcal{D}$ in a language of the propositional situation calculus, the filtration of the set of models of $\mathcal{D}$ is bisimilar to each model of $\mathcal{D}$.*

Proof. Suppose that $\mathfrak{A}$ is a model of $\mathcal{D}$. By Lemma A.1.1, $\mathfrak{A}$ is bisimilar to $\mathfrak{A}^f$ (i.e., $\mathfrak{A} \leftrightarrows \mathfrak{A}^f$).

By Lemma A.1.2, $\mathfrak{A}^f$ is isomorphic to $\mathfrak{F}^{\mathcal{D}}$, and hence, there exists a one-to-one and onto homomorphism g from $\mathfrak{A}^f$ to $\mathfrak{F}^{\mathcal{D}}$. Let Z be a non-empty binary relation on a set $\mathrm{Sit}^{\mathfrak{A}^f} \times \mathrm{Sit}^{\mathfrak{F}^{\mathcal{D}}}$ such that $Z := (t, g(t))$ for each element t in $\mathrm{Sit}^{\mathfrak{A}^f}$; indeed, Z is a bisimulation between $\mathfrak{A}^f$ and $\mathfrak{F}^{\mathcal{D}}$ (cf. Definition 2.4.2). Thus, $\mathfrak{A}^f$ is bisimilar to $\mathfrak{F}^{\mathcal{D}}$ (i.e., $\mathfrak{A}^f \leftrightarrows \mathfrak{F}^{\mathcal{D}}$).

Because $\mathfrak{A} \underline{\leftrightarrow} \mathfrak{A}^f$ and $\mathfrak{A}^f \underline{\leftrightarrow} \mathfrak{F}^{\mathcal{D}}$, by the transitivity of bisimulations, $\mathfrak{A} \underline{\leftrightarrow} \mathfrak{F}^{\mathcal{D}}$. Thus, we have proved that the filtration of the set of models of $\mathcal{D}$ is bisimilar to every model of $\mathcal{D}$. □

Appendix B

Proofs of Chapter 3

We shall first present the preliminaries of fixed-point operations, and then, by using the preliminaries, prove Theorems 3.3.3 and 3.3.5.

B.1 Preliminaries of Fixed-Point Operations

Given a finite set A, a function or operator $F : \wp(A) \to \wp(A)$ gives rise to the following sequence of sets:

$$\emptyset, F(\emptyset), F(F(\emptyset)), \ldots.$$

We denote the members of the sequence as follows:

$$F_0 := \emptyset, F_{n+1} := F(F_n),$$

and call F_n the n-th *stage* of F. If there exists a natural number n_0 such that $F_{n_0+1} = F_{n_0}$ (i.e., $F(F_{n_0}) = F_{n_0}$), then, we say that the *fixed-point* of F exists. We denote F_{n_0}, the fixed-point of F, by F_∞. Moreover, $F(F_\infty) = F_\infty$. If the fixed-point of F does not exist, then we set F_∞ to $\emptyset$. We say that F is *monotone* if, for arbitrary subsets X and Y of A,

$$X \subseteq Y \Longrightarrow F(X) \subseteq F(Y).$$

Lemma B.1.1. *[12] If a unary operator F is monotone, then F_∞ is the least fixed-point of F; that is, $F(F_\infty) = F_\infty$ and $F(X) = X$ implies $F_\infty \subseteq X$. (Even $F(X) \subseteq X$ implies $F_\infty \subseteq X$.)*

We shall define the fixed-point of a function or operator whose arity is greater than one. Given r finite sets $A_1, A_2, \ldots, A_r$ and r functions or operators

$$\begin{aligned} F^1 &: \wp(A_1) \times \wp(A_2) \times \cdots \times \wp(A_r) \to \wp(A_1) \\ F^2 &: \wp(A_1) \times \wp(A_2) \times \cdots \times \wp(A_r) \to \wp(A_2) \\ &\vdots \\ F^r &: \wp(A_1) \times \wp(A_2) \times \cdots \times \wp(A_r) \to \wp(A_r), \end{aligned}$$

we define a sequence $(F^1_{(i)}, F^2_{(i)}, \ldots, F^r_{(i)})_{i \geq 0}$ as follows: for each integer j from 1 to r,

$$F^j_{(0)} := \emptyset, F^j_{(n+1)} := F^j(F^1_{(n)}, F^2_{(n)}, \ldots, F^r_{(n)}).$$

If there exists a natural number n_0 such that

$$(F^1_{(n_0+1)}, F^2_{(n_0+1)}, \ldots, F^r_{(n_0+1)}) = (F^1_{(n_0)}, F^2_{(n_0)}, \ldots, F^r_{(n_0)}),$$

then we say that the *simultaneous fixed-point* of a tuple $(F^1, F^2, \ldots, F^r)$ of r operators exists. We denote the fixed-point of $(F^1, F^2, \ldots, F^r)$ by $(F^1_{(\infty)}, F^2_{(\infty)}, \ldots, F^r_{(\infty)})$. Moreover, for each integer i from 1 to r,

$$F^i(F^1_{(\infty)}, F^2_{(\infty)}, \ldots, F^r_{(\infty)}) = F^i_{(\infty)}.$$

If the fixed-point of $(F^1, F^2, \ldots, F^r)$ does not exist, then, for each integer i from 1 to r, we set $F^i_{(\infty)}$ to $\emptyset$. We say that $(F^1, F^2, \ldots, F^r)$ is monotone if, for each integer i from 1 to r,

$$X_1 \subseteq Y_1, X_2 \subseteq Y_2, \ldots, X_r \subseteq Y_r \Longrightarrow F^i(X_1, X_2, \ldots, X_r) \subseteq F^i(Y_1, Y_2, \ldots, Y_r)$$

where, for each integer j from 1 to r, X_j and Y_j are arbitrary subsets of A_j.

Lemma B.1.2. *[12] If a tuple $(F^1, F^2, \ldots, F^r)$ of r operators is monotone, then*

$$(F^1_{(\infty)}, F^2_{(\infty)}, \ldots, F^r_{(\infty)})$$

is the simultaneous least fixed-point of $(F^1, F^2, \ldots, F^r)$.

We can express a simultaneous fixed-point by the fixed-point of nested monotone operators.

Lemma B.1.3. *[12] Let F and G be monotone operators such that*

$$F : \wp(A^k) \times \wp(A^l) \to \wp(A^k)$$

and

$$G : \wp(A^k) \times \wp(A^l) \to \wp(A^l).$$

Moreover, let E be an operator such that

$$E : \wp(A^k) \to \wp(A^k)$$

and

$$E(X) = F(X, G(X, \_)_\infty)$$

where $G(X, \_) : \wp(A^l) \to \wp(A^l)$ denotes the monotone operator $Y \mapsto G(X, Y)$ and $G(X, \_)_\infty$ denotes its least fixed-point. Then, E is monotone and $E_\infty = F_{(\infty)}$.

The following lemma is the generalization of Lemma B.1.3, which we shall use to prove Theorems 3.3.3 and 3.3.5.

Lemma B.1.4. *For each integer n greater than or equal to 2, the simultaneous fixed-point of n monotone operators is computable by n nested fixed-points.*

Proof. We prove the lemma by mathematical induction on n. The base case is $n = 2$. Let F^1 and F^2 be monotone operators such that

$$F^1 : \wp(A^{k_1}) \times \wp(A^{k_2}) \to \wp(A^{k_1})$$

and

$$F^2 : \wp(A^{k_1}) \times \wp(A^{k_2}) \to \wp(A^{k_2}).$$

Moreover, let E^1 be an operator such that

$$E^1 : \wp(A^{k_1}) \to \wp(A^{k_1})$$

and

$$E^1(X) = F^1(X, F^2(X, \_)_\infty)$$

where $F^2(X, \_) : \wp(A^{k_2}) \to \wp(A^{k_2})$ denotes the monotone operator $Y \mapsto F^2(X, Y)$ and $F^2(X, \_)_\infty$ denotes its least fixed-point. Then, by Lemma B.1.3, E^1 is monotone and $E^1_\infty = F^1_{(\infty)}$.

Similarly, let E^2 be an operator such that

$$E^2 : \wp(A^{k_1}) \to \wp(A^{k_1})$$

and

$$E^2(Y) = F^2(F^1(\_, Y)_\infty, Y)$$

where $F^1(\_, Y) : \wp(A^{k_1}) \to \wp(A^{k_1})$ denotes the monotone operator $X \mapsto F^1(X, Y)$ and $F^1(\_, Y)_\infty$ denotes its least fixed-point. Furthermore, let I^1 and I^2 and monotone operators such that

$$I^1 : \wp(A^{k_2}) \times \wp(A^{k_1}) \to \wp(A^{k_2})$$

and

$$I^2 : \wp(A^{k_2}) \times \wp(A^{k_1}) \to \wp(A^{k_1})$$

with $I^1(Y, X) = F^2(X, Y)$ and $I^2(Y, X) = F^1(X, Y)$. Then, let H be an operator such that

$$H : \wp(A^{k_2}) \to \wp(A^{k_2})$$

and

$$H(Y) = I^1(Y, I^2(Y, \_)_\infty).$$

By Lemma B.1.3, H is monotone and $H_\infty = I^1_{(\infty)}$. Indeed, $H_\infty = E^2_\infty$ and $I^1_{(\infty)} = F^2_{(\infty)}$, and hence, E^2 is monotone and $E^2_\infty = F^2_{(\infty)}$. Therefore, E^1 and E^2 are monotone, and $(F^1_{(\infty)}, F^2_{(\infty)}) = (E^1_\infty, E^2_\infty)$ where E^1_∞ and E^2_∞ are nested fixed-points.

In the inductive step, we suppose that, for some $m \in Z^+ \backslash 1$, the simultaneous fixed-point of m monotone operations is computable by m nested fixed-points. For each $i \in \{1, 2, \ldots, m+1\}$, let F^i be a monotone operator such that

$$F^i : \wp(A^{k_1}) \times \wp(A^{k_2}) \times \cdots \times \wp(A^{k_{m+1}}) \to \wp(A^{k_i}).$$

and, for each $j \in \{1, 2, \ldots, m+1\} \backslash i$, suppose that

$$F^j(X/Z_i) : \wp(A^{k_1}) \times \cdots \times \wp(A^{k_{i-1}}) \times \wp(A^{k_{i+1}}) \times \cdots \times \wp(A^{k_{m+1}}) \to \wp(A^{k_j})$$

denotes the monotone operation $X \mapsto F^j(Z_1, \ldots, Z_{i-1}, X, Z_{i+1}, \ldots, Z_{m+1})$, and that

$$F^j(X/Z_i)_{(\infty)}$$

denotes its least fixed-point.

By the inductive hypothesis, for each $i \in \{1, 2, \ldots, m+1\}$, the simultaneous fixed-point

$$(F^j(X/Z_i)_{(\infty)})_{j \in \{1,2,\ldots,m+1\} \backslash i}$$

of m monotone operations is computable by m nested fixed-points; that is, for each monotone operator $F^j(X/Z_i)$, there exists a monotone operator whose nested fixed-point coincides with $F^j(X/Z_i)_{(\infty)}$. Let $E_i^j(X, \_) : \wp(A^{k_j}) \to \wp(A^{k_j})$ denote the monotone operator that maps X to the monotone operator whose nested fixed-point coincides with $F^j(X/Z_i)_{(\infty)}$. Consequently, $E_i^j(X, \_)_\infty = F^j(X/Z_i)_{(\infty)}$.

For each $i \in \{1, 2, \ldots, m+1\}$, let E^i be an operator such that

$$E^i : \wp(A^{k_i}) \to \wp(A^{k_i})$$

and

$$E^i(X) = F^i(E_i^1(X, \_)_\infty, \ldots, E_i^{i-1}(X, \_)_\infty, X, E_i^{i+1}(X, \_)_\infty, \ldots, E_i^{m+1}(X, \_)_\infty).$$

Since $E^i(X)$ is positive in X, E^i is monotone.

We are going to show that, for each $i \in \{1, 2, \ldots, m+1\}$, $E_\infty^i \subseteq F_{(\infty)}^i$. By the definition of the operator E^i,

$$\begin{aligned} & E^i(F_{(\infty)}^i) \\ &= F^i(E_i^1(F_{(\infty)}^i, \_)_\infty, \ldots, E_i^{i-1}(F_{(\infty)}^i, \_)_\infty, F_{(\infty)}^i, E_i^{i+1}(F_{(\infty)}^i, \_)_\infty, \ldots, E_i^{m+1}(F_{(\infty)}^i, \_)_\infty). \end{aligned}$$

For each $j \in \{1, 2, \ldots, m+1\} \backslash i$,

$$E_i^j(F_{(\infty)}^i, \_)_\infty = F^j(F_{(\infty)}^i/Z_i)_{(\infty)}$$

and

$$F^j(F_{(\infty)}^i/Z_i)_{(\infty)} \subseteq F^j(F_{(\infty)}^1, F_{(\infty)}^2, \ldots, F_{(\infty)}^{m+1}) = F_{(\infty)}^j$$

and hence

$$E_i^j(F_{(\infty)}^i, \_)_\infty \subseteq F_{(\infty)}^j.$$

Since F^i is monotone,

$$\begin{aligned} & F^i(E_i^1(F_{(\infty)}^i, \_)_\infty, \ldots, F_{(\infty)}^i, E_i^{i+1}(F_{(\infty)}^i, \_)_\infty, \ldots, E_i^{m+1}(F_{(\infty)}^i, \_)_\infty) \\ &\subseteq F^i(F_{(\infty)}^1, \ldots, F_{(\infty)}^{i-1}, F_{(\infty)}^i, F_{(\infty)}^{i+1}, \ldots, F_{(\infty)}^{l+1}) \\ &= F_{(\infty)}^i \end{aligned}$$

and hence, $E^i(F^i_{(\infty)}) \subseteq F^i_{(\infty)}$. By Lemma B.1.1, $E^i(F^i_{(\infty)}) \subseteq F^i_{(\infty)}$ implies $E^i_\infty \subseteq F^i_{(\infty)}$. Therefore, $E^i_\infty \subseteq F^i_{(\infty)}$.

Furthermore, we are going to show that, for each $i \in \{1, 2, \ldots, m+1\}$, $F^i_{(\infty)} \subseteq E^i_\infty$. First, we need to show by mathematical induction that, for each $n \in N$, $F^i_{(n)} \subseteq E^i_\infty$, and, for each $j \in \{1, 2, \ldots, m+1\} \backslash i$, $F^j_{(n)} \subseteq E^j_i(E^i_\infty, \_)_\infty$. The base case is $n = 0$. Since $F^i_{(0)} = \emptyset$, and, for $j \in \{1, 2, \ldots, m+1\} \backslash i$, $F^j_{(0)} = \emptyset$, $F^i_{(0)} \subseteq E^i_\infty$ and $F^j_{(0)} \subseteq E^j_i(E^i_\infty, \_)_\infty$.

In the inductive step, we suppose that, for some $k \in N$, $F^i_{(k)} \subseteq E^i_\infty$ and $j \in \{1, 2, \ldots, m+1\} \backslash i$, $F^j_{(k)} \subseteq E^j_i(E^i_\infty, \_)_\infty$. Then,

$$F^i_{(k+1)} = F^i(F^1_{(k)}, \ldots, F^{i-1}_{(k)}, F^i_{(k)}, F^{i+1}_{(k)}, \ldots, F^{l+1}_{(k)}).$$

Since F^i is monotone, by the inductive hypothesis,

$$\begin{aligned} & F^i(F^1_{(k)}, \ldots, F^{i-1}_{(k)}, F^i_{(k)}, F^{i+1}_{(k)}, \ldots, F^{l+1}_{(k)}) \\ \subseteq\ & F^i(E^1_i(E^i_\infty, \_)_\infty, \ldots, E^{i-1}_i(E^i_\infty, \_)_\infty, E^i_\infty, E^{i+1}_i(E^i_\infty, \_)_\infty, \ldots, E^{m+1}_i(E^i_\infty, \_)_\infty) \\ =\ & E^i(E^i_\infty) \\ =\ & E^i_\infty. \end{aligned}$$

Similarly, for each $j \in \{1, 2, \ldots, m+1\} \backslash i$,

$$\begin{aligned} F^j_{(k+1)} &= F^j(F^1_{(k)}, \ldots, F^{i-1}_{(k)}, F^i_{(k)}, F^{i+1}_{(k)}, \ldots, F^{l+1}_{(k)}) \\ &\subseteq F^j(E^1_i(E^i_\infty, \_)_\infty, \ldots, E^{i-1}_i(E^i_\infty, \_)_\infty, E^i_\infty, E^{i+1}_i(E^i_\infty, \_)_\infty, \ldots, E^{m+1}_i(E^i_\infty, \_)_\infty). \end{aligned}$$

For each $j \in \{1, 2, \ldots, m+1\} \backslash i$, $E^j_i(E^i_\infty, \_)_\infty = F^j(E^i_\infty / Z_i)_{(\infty)}$, and hence

$$\begin{aligned} & F^j(E^1_i(E^i_\infty, \_)_\infty, \ldots, E^{i-1}_i(E^i_\infty, \_)_\infty, E^i_\infty, E^{i+1}_i(E^i_\infty, \_)_\infty, \ldots, E^{m+1}_i(E^i_\infty, \_)_\infty) \\ =\ & F^j(F^1(E^i_\infty / Z_i)_{(\infty)}, \ldots, F^{i-1}(E^i_\infty / Z_i)_{(\infty)}, E^i_\infty, F^{i+1}(E^i_\infty / Z_i)_{(\infty)}, \ldots, F^{m+1}(E^i_\infty / Z_i)_{(\infty)}) \\ =\ & F^j(E^i_\infty / Z_i)_{(\infty)} \\ =\ & E^j_i(E^i_\infty, \_)_\infty. \end{aligned}$$

By the base case and the inductive step, we have shown that, for each $n \in N$, $F^i_{(n)} \subseteq E^i_\infty$, and, for each $j \in \{1, 2, \ldots, m+1\} \backslash i$, $F^j_{(n)} \subseteq E^j_i(E^i_\infty, \_)_\infty$, and hence, $F^i_{(\infty)} \subseteq E^i_\infty$.

Since $E^i_\infty \subseteq F^i_\infty$ and $F^i_\infty \subseteq E^i_\infty$, $E^i_\infty = F^i_\infty$. we have shown that $(F^1_{(\infty)}, \ldots, F^{m+1}_{(\infty)}) = (E^1_\infty, \ldots, E^{m+1}_\infty)$ where, for each $i \in \{1, 2, \ldots, m+1\}$, E^i_∞ is a nested fixed-point.

Finally, by the base case and the inductive step, we have proved that, for each $n \in Z^+ \backslash 1$, the simultaneous fixed-point of n operators is computable by n nested fixed-points. ■

B.2 Proof of Theorem 3.3.3

Theorem 3.3.3 *QL is equivalently as expressive as the alternation-free μ-calculus.*

Proof. Given a set ρ of action symbols and a set σ of proposition symbols, we shall let L_μ denote the (σ, ρ)-language of the alternation-free μ-calculus. We view ρ and σ as a set of action constant symbols and a set of unary relational fluent symbols, respectively, and hence, $\tau := \{S_0, do\} \cup \rho \cup \sigma$ is a vocabulary of the propositional situation. We shall let L_{QL} denote the τ-language of QL.

To show that QL is equivalently as expressive as the alternation-free μ-calculus, we shall first show that the alternation-free μ-calculus is less expressive than QL (i.e., $L_\mu \leq L_{QL}$), and then show that QL is less expressive than the alternation-free μ-calculus (i.e., $L_{QL} \leq L_\mu$). Concretely, given a sentence of the alternation-free μ-calculus, we shall show how to construct a sentence of QL that defines the same set of structures as the given sentence does, and vice versa.

(Proof of $L_\mu \leq L_{QL}$) Given a sentence ϕ of the alternation-free μ-calculus, we shall show how to construct a sentence ϕ' of QL such that $\phi' = \Delta \wedge H(S_0)$ with a definition Δ in L_{QL} and a predicate symbol H in τ^{d}_Δ, and, for every structure $\mathfrak{A}$ that interprets the symbols in τ,

$$\mathfrak{A}, S_0^{\mathfrak{A}} \models \phi \iff \mathfrak{A}^\Delta \models \Delta \wedge H(S_0);$$

that is,

$$\|\phi\|_v^{\mathfrak{A}} = H^{\mathfrak{A}^\Delta}.$$

First, we shall inductively define how to construct ϕ' from ϕ. We shall then prove by mathematical induction on the structure of ϕ that $\|\phi\|_v^{\mathfrak{A}} = H^{\mathfrak{A}^\Delta}$.

In the following definition of how to construct ϕ' from ϕ, we shall let F denote a symbol in σ, act a symbol in ρ, ϕ_1 and ϕ_2 formulae in L_μ, and Δ_1 and Δ_2 definitions in L_{QL} such that H does not occur in either Δ_1 or Δ_2. Moreover, we shall let H_1 and H_2 denote predicate symbols such that $H_1 \in \tau^{\mathrm{o}}_{\Delta_1}$, $H_2 \in \tau^{\mathrm{o}}_{\Delta_2}$, ${H_1}^{\mathfrak{A}^\Delta} = \|\phi_1\|_v^{\mathfrak{A}}$, and ${H_2}^{\mathfrak{A}^\Delta} = \|\phi_2\|_v^{\mathfrak{A}}$.

- If ϕ is F, then ϕ' is $\{\forall s[H(s) \leftarrow F(s)]\} \wedge H(S_0)$.
- If ϕ is $\neg\phi_1$, then ϕ' is $(\Delta_1 \cup \{\forall s[H(s) \leftarrow \neg H_1(s)]\}) \wedge H(S_0)$.
- If ϕ is $\phi_1 \wedge \phi_2$, then ϕ' is $(\Delta_1 \cup \Delta_2 \cup \{\forall s[H(s) \leftarrow H_1(s) \wedge H_2(s)]\}) \wedge H(S_0)$.

- If ϕ is $\langle act\rangle\phi_1$, then ϕ' is

$$(\Delta_1 \cup \{\forall s[H(s) \leftarrow \exists s'[s' = do(act, s) \wedge H_1(s')]]\}) \wedge H(S_0).$$

- If ϕ is $\mu Z.\phi_1(Z)$, then ϕ' is

$$(\Delta_1(H/Z) \cup \{\forall s[H(s) \leftarrow H_1(s)]\}) \wedge H(S_0)$$

  where Z in Δ_1 is a predicate symbol that corresponds to the propositional variable Z in ϕ_1 and $\Delta_1(H/Z)$ denotes a definition that we obtain by replacing by H each occurrence of Z in Δ_1.

We shall prove by mathematical induction on the structure of ϕ that $\|\phi\|_v^{\mathfrak{A}} = H^{\mathfrak{A}^\Delta}$. The base case is when ϕ is F. Then, ϕ' is $\{\forall s[H(s) \leftarrow F(s)]\} \wedge H(S_0)$. By the structural semantics of the μ-calculus, $\|F\|_v^{\mathfrak{A}} = F^{\mathfrak{A}}$. Moreover, $H^{\mathfrak{A}^\Delta} = F^{\mathfrak{A}}$. Thus, $\|F\|_v^{\mathfrak{A}} = H^{\mathfrak{A}^\Delta}$. The inductive step consists of the following four cases:

- When ϕ is $\neg\phi_1$, ϕ' is $(\Delta_1 \cup \{\forall s[H(s) \leftarrow \neg H_1(s)]\}) \wedge H(S_0)$ where ${H_1}^{\mathfrak{A}^{\Delta_1}} = \|\phi_1\|_v^{\mathfrak{A}}$. By the structural semantics of the μ-calculus, $\|\neg\phi_1\|_v^{\mathfrak{A}} = Sit^{\mathfrak{A}} - \|\phi_1\|_v^{\mathfrak{A}}$, and hence, because $\|\phi_1\|_v^{\mathfrak{A}} = {H_1}^{\mathfrak{A}^{\Delta_1}}$, $\|\neg\phi_1\|_v^{\mathfrak{A}} = Sit^{\mathfrak{A}} - {H_1}^{\mathfrak{A}^{\Delta_1}}$. Moreover, $H^{\mathfrak{A}^\Delta} = Sit^{\mathfrak{A}} - {H_1}^{\mathfrak{A}^{\Delta_1}}$, and hence, because $\|\neg\phi_1\|_v^{\mathfrak{A}} = Sit^{\mathfrak{A}} - {H_1}^{\mathfrak{A}^{\Delta_1}}$, $\|\neg\phi_1\|_v^{\mathfrak{A}} = H^{\mathfrak{A}^\Delta}$.

- When ϕ is in $\phi_1 \wedge \phi_2$, ϕ' is $(\Delta_1 \cup \Delta_2 \cup \{\forall s[H(s) \leftarrow H_1(s) \wedge H_2(s)]\}) \wedge H(S_0)$ where ${H_1}^{\mathfrak{A}^{\Delta_1}} = \|\phi_1\|_v^{\mathfrak{A}}$ and ${H_2}^{\mathfrak{A}^{\Delta_2}} = \|\phi_2\|_v^{\mathfrak{A}}$. By the structural semantics of the μ-calculus, $\|\phi_1 \wedge \phi_2\|_v^{\mathfrak{A}} = \|\phi_1\|_v^{\mathfrak{A}} \cap \|\phi_2\|_v^{\mathfrak{A}}$, and hence, because $\|\phi_1\|_v^{\mathfrak{A}} = {H_1}^{\mathfrak{A}^{\Delta_1}}$ and $\|\phi_2\|_v^{\mathfrak{A}} = {H_2}^{\mathfrak{A}^{\Delta_2}}$, $\|\phi_1 \wedge \phi_2\|_v^{\mathfrak{A}} = {H_1}^{\mathfrak{A}^{\Delta_1}} \cap {H_2}^{\mathfrak{A}^{\Delta_2}}$. Moreover, $H^{\mathfrak{A}^\Delta} = {H_1}^{\mathfrak{A}^{\Delta_1}} \cap {H_2}^{\mathfrak{A}^{\Delta_2}}$, and hence, because $\|\phi_1 \wedge \phi_2\|_v^{\mathfrak{A}} = {H_1}^{\mathfrak{A}^{\Delta_1}} \cap {H_2}^{\mathfrak{A}^{\Delta_2}}$, $\|\phi_1 \wedge \phi_2\|_v^{\mathfrak{A}} = H^{\mathfrak{A}^\Delta}$.

- When ϕ is $\langle act\rangle\phi_1$, ϕ' is $(\Delta_1 \cup \{\forall s[H(s) \leftarrow \exists s'[s' = do(act, s) \wedge H_1(s')]]\}) \wedge H(S_0)$ where ${H_1}^{\mathfrak{A}^{\Delta_1}} = \|\phi_1\|_v^{\mathfrak{A}}$. By the structural semantics of the μ-calculus, $\|\langle act\rangle\phi_1\|_v^{\mathfrak{A}} = \{s_1 \mid \exists s_2 \text{ with } s_2 = do^{\mathfrak{A}}(act^{\mathfrak{A}}, s_1) \text{ and } s_2 \in \|\phi_1\|_v^{\mathfrak{A}}\}$, and hence, because $\|\phi_1\|_v^{\mathfrak{A}} = {H_1}^{\mathfrak{A}^{\Delta_1}}$, $\|\langle act\rangle\phi_1\|_v^{\mathfrak{A}} = \{s_1 \mid \exists s_2 \text{ with } s_2 = do^{\mathfrak{A}}(act^{\mathfrak{A}}, s_1) \text{ and } s_2 \in {H_1}^{\mathfrak{A}^{\Delta_1}}\}$. Moreover, $H^{\mathfrak{A}^\Delta} = \{s_1 \mid \exists s_2 \text{ with } s_2 = do^{\mathfrak{A}}(act^{\mathfrak{A}}, s_1) \text{ and } s_2 \in {H_1}^{\mathfrak{A}^{\Delta_1}}\}$, and hence, because $\|\langle act\rangle\phi_1\|_v^{\mathfrak{A}} = \{s_1 \mid \exists s_2 \text{ with } s_2 = do^{\mathfrak{A}}(act^{\mathfrak{A}}, s_1) \text{ and } s_2 \in {H_1}^{\mathfrak{A}^{\Delta_1}}\}$, $\|\langle act\rangle\phi_1\|_v^{\mathfrak{A}} = H^{\mathfrak{A}^\Delta}$.

- When ϕ is $\mu Z.\phi_1(Z)$, then ϕ' is $(\Delta_1(H/Z) \cup \{\forall s[H(s) \leftarrow H_1(s)]\}) \wedge H(S_0)$ where, for each subset A of $\mathrm{Sit}^{\mathfrak{A}}$, ${H_1}^{\mathfrak{A}^{\Delta_1(A/Z)}} = \|\phi_1\|_{v[A/Z]}^{\mathfrak{A}}$. By the structural semantics of

the μ-calculus, $\|\mu Z.\phi_1(Z)\|_v^{\mathfrak{A}} = \bigcap\{A \subseteq \mathrm{Sit}^{\mathfrak{A}} \mid \|\phi_1\|_{v[A/Z]}^{\mathfrak{A}} \subseteq A\}$, and hence, because $\|\phi_1\|_{v[A/Z]}^{\mathfrak{A}} = {H_1}^{\mathfrak{A}^{\Delta_1(A/Z)}}$ for any subset A of $\mathrm{Sit}^{\mathfrak{A}}$,

$$\|\mu Z.\phi_1(Z)\|_v^{\mathfrak{A}} = \bigcap\{A \subseteq \mathrm{Sit}^{\mathfrak{A}} \mid {H_1}^{\mathfrak{A}^{\Delta_1(A/Z)}} \subseteq A\}.$$

Suppose that, given a rule r, functions *head* and *body* return the head of r and the body of r, respectively. Moreover, suppose that, given a definition Δ and a predicate symbol H in τ_Δ°, R_Δ^H returns the set of rules in Δ such that, for each rule r in R_Δ^H, $head(r) = H$. We shall let a function $F^{H_1} : \wp(\mathrm{Sit}^{\mathfrak{A}} \times \mathrm{Sit}^{\mathfrak{A}}) \to \wp(\mathrm{Sit}^{\mathfrak{A}})$ denote an operator such that

$$F^{H_1}(X_1, X_2) = \left[\bigvee_{r \in R_{\Delta_1}^{H_1}} body(r)\right]^{\mathfrak{A}(H_1:X_1, Z:X_2)}.$$

Because ${H_1}^{\mathfrak{A}^{\Delta_1(A/Z)}} = F^{H_1}(\_, A)_\infty$ and

$$\|\mu Z.\phi_1(Z)\|_v^{\mathfrak{A}} = \bigcap\{A \subseteq \mathrm{Sit}^{\mathfrak{A}} \mid {H_1}^{\mathfrak{A}^{\Delta_1(A/Z)}} \subseteq A\},$$

$\|\mu Z.\phi_1(Z)\|_v^{\mathfrak{A}} = \bigcap\{A \subseteq \mathrm{Sit}^{\mathfrak{A}} \mid F^{H_1}(\_, A)_\infty \subseteq A\}$. We shall let a function $F^H : \wp(\mathrm{Sit}^{\mathfrak{A}} \times \mathrm{Sit}^{\mathfrak{A}} \times \mathrm{Sit}^{\mathfrak{A}}) \to \wp(\mathrm{Sit}^{\mathfrak{A}})$ denote an operator such that

$$F^H(X_1, X_2) = \left[\bigvee_{r \in R_\Delta^H} body(r)\right]^{\mathfrak{A}(H_1:X_1, H:X_2)};$$

that is,

$$F^H(X_1, X_2) = {H_1}^{\mathfrak{A}(H_1:X_1, H:X_2)} = X_1.$$

We shall then let a function $E^H : \wp(Sit^{\mathfrak{A}}) \to \wp(Sit^{\mathfrak{A}})$ denote an operator such that

$$E^H(X) = F^H(F^{H_1}(\_, X)_\infty, X) = F^{H_1}(\_, X)_\infty.$$

By Lemma B.1.3, ${E^H}_\infty = {F^H}_{(\infty)}$. Because the well-founded semantics coincides with the least fixed-point semantics for stratifiable definitions of ID-logic, $H^{\mathfrak{A}^\Delta} = {F^H}_{(\infty)}$. Thus, $H^{\mathfrak{A}^\Delta} = {E^H}_\infty$. Furthermore, because $F^{H_1}(\_, X)_\infty = E^H(X)$ and

$$\|\mu Z.\phi_1(Z)\|_v^{\mathfrak{A}} = \bigcap\{A \subseteq \mathrm{Sit}^{\mathfrak{A}} \mid F^{H_1}(\_, A)_\infty \subseteq A\},$$

$\|\mu Z.\phi_1(Z)\|_v^{\mathfrak{A}} = \bigcap\{A \subseteq \mathrm{Sit}^{\mathfrak{A}} \mid E^H(A) \subseteq A\}$, and hence, $\|\mu Z.\phi_1(Z)\|_v^{\mathfrak{A}} = E_\infty^H$. Thus, because $H^{\mathfrak{A}^\Delta} = {E^H}_\infty$ and $\|\mu Z.\phi_1(Z)\|_v^{\mathfrak{A}} = {E^H}_\infty$, $\|\mu Z.\phi_1(Z)\|_v^{\mathfrak{A}} = H^{\mathfrak{A}^\Delta} = {E^H}_\infty$.

We have proved by mathematical induction on the structure of ϕ that $\|\phi\|_v^{\mathfrak{A}} = H^{\mathfrak{A}^\Delta}$.
(Proof of $L_{QL} \leq L_\mu$) Given a sentence $\Delta \wedge H(S_0)$ of L_{QL}, we shall show how to construct a sentence ϕ of L_μ such that, for every structure $\mathfrak{A}$ that interprets the symbols in τ,

$$\mathfrak{A}^\Delta \models \Delta \wedge X(S_0) \iff \mathfrak{A}, S_0^{\mathfrak{A}} \models \phi;$$

that is, $H^{\mathfrak{A}^\Delta} = \|\phi\|^{\mathfrak{A}}$. First, we shall inductively define how to construct ϕ, and then prove that $H^{\mathfrak{A}^\Delta} = \|\phi\|^{\mathfrak{A}}$ by mathematical induction on a stratification of Δ.

For each relational fluent symbol F of σ, we construct a sentence F of L_μ. Moreover, given each stratum Δ_i of Δ from the lowest to the highest, for each defined symbol H of Δ_i, we construct a sentence ϕ^H of L_μ as follows:

1. Given a rule r of Δ_i, we construct a sentence ϕ^r of L_μ as follows: given that ϕ' is a conjunction of literals such that $\bigwedge_{X \in A}(\neg)X(s)$ for some subset A of $\sigma \cup \tau_\Delta^{\mathrm{d}}$, and H and X_ψ are predicate symbols, which are not necessarily distinct.

   - If r is $\forall s[H(s) \leftarrow \phi'(s)]$, then ϕ^r is $\bigwedge_{X \in A}(\neg)\alpha^X$ where α^X is X if $X \in \tau_{\Delta_i}^{\mathrm{d}}$; otherwise, α^X is ϕ^X.
   - If r is $\forall s[H(s) \leftarrow \exists s'[s' = do(act, s) \wedge X_\psi(s')]]$, then ϕ^r is $\langle act \rangle X$ if $X \in \tau_{\Delta_i}^{\mathrm{d}}$. Otherwise, ϕ^r is $\langle act \rangle \phi^X$.

2. For each defined symbol H of Δ_i, we construct a set Φ^H of sentences of L_μ such that

$$\Phi^H = \{\phi^r : head(r) = H\}.$$

3. For each defined symbol H of Δ_i, we construct a sentence ϕ'^H of L_μ such that $\phi'^H = \bigvee_{\phi^r \in \Phi^H} \phi^r$. If ϕ'^H contains no defined symbol of Δ_i, then ϕ^H is ϕ'^H. Otherwise, given that ϕ'^H contains l defined symbols $H_1, \ldots, H_l$ of Δ_i, we perform the following procedure:

   (a) For each pair (j, k) of integers from 1 to l, if $j \neq k$, $H_j \neq H$, and $H_k \neq H$, then we replace each appearance of H_k in ϕ'^{H_j} with $\mu H_k.\phi'^{H_k}$.

   (b) For each integer j from 1 to l, if $H_j \neq H$, then we replace each appearance of H_j in ϕ'^H with $\mu H_j.\phi'^{H_j}(H_j, H)$.

   Then, ϕ^H is $\mu X.\phi'^H(X)$.

We have inductively defined how to construct a sentence ϕ of L_μ for each defined symbol H of Δ. We shall prove that $H^{\mathfrak{A}^\Delta} = \|\phi\|^{\mathfrak{A}}$ by mathematical induction on a stratification of Δ.

For each relational fluent symbol F of σ, we construct a sentence F of L_μ such that $F^{\mathfrak{A}^\Delta} = F^{\mathfrak{A}} = \|F\|$. This is the base case. For the inductive step, we shall let ϕ^H denote a sentence of L_μ for H. Given each stratum Δ_i of Δ from the lowest to the highest, for each defined symbol H of Δ_i, we construct a set R^H such that $\{r : r \in \Delta_i, head(r) = H\}$, and then the following simultaneous fixed-point operator $F^H : \wp((\mathrm{Sit}^{\mathfrak{A}})^k) \to \wp(\mathrm{Sit}^{\mathfrak{A}})$:

$$F^H(A_1, \ldots, A_k) = \{t \in \mathrm{Sit}^{\mathfrak{A}} : \models_{\mathfrak{A}(s:t, H_1:A_1, \ldots, H_k:A_k)} \bigvee_{r \in R^H} body(r)(s)\}$$

where $k = |\tau^{\mathrm{d}}_{\Delta_i}|$ and $\tau^{\mathrm{d}}_{\Delta_i} = \{H_1, \ldots, H_k\}$. We shall show that $F^H_{(\infty)} = \|\phi^H\|$.

If $\bigvee_{r \in R^H} body(r)(s)$ contains no defined symbol of Δ_i, then ϕ^H is $\bigcup_{\phi^r \in \Phi^H} \phi^r$. Moreover,

$$\begin{aligned}
F^H_{(\infty)} &= F^H_{(1)} = \{t \in \mathrm{Sit}^{\mathfrak{A}} : \models_{\mathfrak{A}(s:t)} \bigvee_{r \in R^H} body(r)(s)\} \\
&= \bigcup_{r \in R^H} \{t \in \mathrm{Sit}^{\mathfrak{A}} : \models_{\mathfrak{A}(s:t)} body(r)(s)\} \\
&= \bigcup_{r \in R^H} \|\phi^r\|^{\mathfrak{A}} \\
&= \bigcup_{\phi^r \in \Phi^H} \|\phi^r\|^{\mathfrak{A}} \\
&= \| \bigcup_{\phi^r \in \Phi^H} \phi^r \|^{\mathfrak{A}} \\
&= \|\phi^H\|^{\mathfrak{A}}.
\end{aligned}$$

Otherwise, ϕ^H is $\mu H.\phi'^H(H)$. Given that $\bigvee_{r \in R^H} body(r)(s)$ contains l defined symbols $H_{i_1}, \ldots, H_{i_l}$ of Δ_i, then we construct the following least fixed-point operator $E^H : \wp(\mathrm{Sit}^{\mathfrak{A}}) \to \wp(\mathrm{Sit}^{\mathfrak{A}})$:

$$E^H(A) = F^H(A/H, G^{H_{i_1}}(A, \_)_\infty/H_{i_1}, \ldots, G^{H_{i_l}}(A, \_)_\infty/H_{i_l})$$

where, for each integer j from 1 to l, if $H_{i_j} \neq H$, then

$$G^{H_{i_j}}(A, \_)_\infty = F^{H_{i_j}}(A/H)_{(\infty)}.$$

By Lemma B.1.4, there exists a monotone operator whose nested fixed-point coincides with $F^H_{(\infty)}$, and, by the construction of such an operator in the proof of Lemma B.1.4, E^H is the

monotone operator with $E^H_\infty = F^{X_i}_{(\infty)}$. Consequently,

$$\begin{aligned} F^H_{(\infty)} = E^H_\infty = \bigcap\{A \subseteq \mathrm{Sit}^{\mathfrak{A}} : E^H(A) \subseteq A\} & \\ = \bigcap\{A \subseteq \mathrm{Sit}^{\mathfrak{A}} : \|\phi'^H\|^{\mathfrak{A}}_{v[A/H]} \subseteq A\} & \\ = \|\mu H.\phi'^H(H)\|^{\mathfrak{A}} & \\ = \|\phi^H\|^{\mathfrak{A}}. & \end{aligned}$$

Because the well-founded semantics coincides with the least fixed-point semantics for stratifiable definitions of ID-logic, $H^{\mathfrak{A}\Delta_i} = F^H{}_{(\infty)}$, and hence, $H^{\mathfrak{A}\Delta_i} = \|\phi^H\|^{\mathfrak{A}}$. □

B.3 Proof of Theorem 3.3.5

Theorem 3.3.5 *Given a basic action theory $\mathcal{D}$ in a language of the propositional situation calculus, QL' is equivalently as expressive as QL with respect to the set of structures that are bisimilar to $\mathfrak{F}^{\mathcal{D}}$.*

Proof. Suppose that τ is a vocabulary of the propositional situation calculus, and that ρ is the set of action constant symbols in τ. We shall let $L_{QL'}$ denote the τ-language of QL. Moreover, suppose that act_0 is an action constant symbol not in ρ. We shall let L_{QL} denote the $(\tau \cup \{act_0\})$-language of QL. Furthermore, we shall let K denote the set of structures that are bisimilar to $\mathfrak{F}^{\mathcal{D}}$.

Because QL is a fragment of QL', $L_{QL} \leq L_{QL'}$ and hence $L_{QL} \leq_K L_{QL'}$. To show that $L_{QL'} =_K L_{QL}$, we shall show that $L_{QL'} \leq_K L_{QL}$. Concretely, given a sentence of $L_{QL'}$, we shall show how to construct a sentence of L_{QL} that defines the same set of structures as the given sentence does with respect to K.

Because K is the set of structures that are bisimilar to $\mathfrak{F}^{\mathcal{D}}$, by Corollary 3.3.6, every sentence of L_{QL} can defines only two sets of structures with respect to K: $\emptyset$ or K. In other words, if $\mathfrak{F}^{\mathcal{D}}$ satisfies a sentence of L_{QL}, then every structure in K satisfies the sentence; otherwise, no structure in K satisfies the sentence. Therefore, to show $L_{QL'} \leq_K L_{QL}$, given a sentence ϕ of $L_{QL'}$, we shall show how to construct a sentence ϕ' of L_{QL} such that $\models_{\mathfrak{F}^{\mathcal{D}}} \phi \iff \models_{\mathfrak{F}^{\mathcal{D}}} \phi'$.

Suppose that $\rho = \{act_1, \ldots, act_m\}$, and that $\mathfrak{F}^{\mathcal{D}}$ interprets act_0 as the unnamed action (i.e., the zero vector) of $\mathrm{Act}^{\mathfrak{F}^{\mathcal{D}}}$ (cf. Definition 2.3.1). Given a sentence $\Delta \wedge X(S_0)$ of $L_{QL'}$, we shall construct a definition Δ' of L_{QL} by replacing each rule of Δ with equivalent rules

of QL. Given that H, A_χ, and X_ψ denote predicate symbols, which are not necessarily distinct, we replace a rule $\forall s[H(s) \leftarrow \exists a \exists s'[s' = do(a, s) \wedge X_\psi(s')]]$ in Δ with the following rules:

$$\begin{aligned}
&\forall s[H(s) \leftarrow \exists s'[s' = do(act_0, s) \wedge X_\psi(s')]] \\
&\forall s[H(s) \leftarrow \exists s'[s' = do(act_1, s) \wedge X_\psi(s')]] \\
&\vdots \\
&\forall s[H(s) \leftarrow \exists s'[s' = do(act_m, s) \wedge X_\psi(s')]].
\end{aligned}$$

Furthermore, given that $\chi_1, \ldots, \chi_l$ are conjunctions of literals, we replace the following rules in Δ:

$$\begin{aligned}
&\forall a[A_\chi(a) \leftarrow \chi_1(a)] \\
&\vdots \\
&\forall a[A_\chi(a) \leftarrow \chi_l(a)] \\
&\forall s[H(s) \leftarrow \exists a \exists s'[A_\chi(a) \wedge s' = do(a, s) \wedge X_\psi(s')]]
\end{aligned}$$

with the following rules:

$$\begin{aligned}
&\forall s[H(s) \leftarrow \exists s'[s' = do(act_{i_1}, s) \wedge X_\psi(s')]] \\
&\vdots \\
&\forall s[H(s) \leftarrow \exists s'[s' = do(act_{i_k}, s) \wedge X_\psi(s')]]
\end{aligned}$$

where, for each integer j from 1 to k, $\models_{\mathfrak{F}^{\mathcal{D}}(a:act_{i_j})} \chi_1 \vee \cdots \vee \chi_l$.

The definition Δ' of L_{QL} computes the same set of defined relations as Δ does, and hence, $X^{\Delta'^{\mathfrak{F}^{\mathcal{D}}}} = X^{\Delta^{\mathfrak{F}^{\mathcal{D}}}}$. Therefore, $\models_{\mathfrak{F}^{\mathcal{D}}} \Delta \wedge X(S_0) \iff \models_{\mathfrak{F}^{\mathcal{D}}} \Delta' \wedge X(S_0)$. □

Bibliography

[1] H. Andréka, I. Németi, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27(3):217–274, June 1998.

[2] K.R. Apt and E.-R. Olderog. *Verification of Sequential and Concurrent Programs.* Graduate Texts in Computer Science. Springer, second edition, 1997.

[3] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 2001.

[4] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *AAAI-00: Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 355–362, Menlo Park, California, July-August 2000. AAAI Press.

[5] M. Cadoli. *Tractable Reasoning in Artificial Intelligence*, volume 941 of *Lecture Notes in Computing Science; Lecture Notes in Artificial Intelligence.* Springer, 1995.

[6] C.C. Chang and H.J. Keisler. *Model Theory*, volume 73 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, third edition, 1990.

[7] M. Denecker, M. Bruynooghe, and V. Marek. Logic programming revisited: Logic programs as inductive definitions. *ACM Transactions on Computational Logic*, 2(4):623–654, October 2001.

[8] M. Denecker and E. Ternovska. Inductive situation calculus. In D. Dubois, C. Welty, and M.-A. Williams, editors, *KR-04: Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning*, pages 545–553, Menlo Park, California, June 2004. KR, Inc., AAAI Press.

[9] M. Denecker and E. Ternovska. A logic of non-monotone inductive definitions and its modularity properties. In V. Lifschitz and I. Niemelä, editors, *LPNMR-04: Proceedings of the Seventh International Conference on Logic Programming and Non-Monotonic Reasoning*, volume 2923 of *Lecture Notes in Computer Science*, pages 47–60. Springer, January 2004.

[10] W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, October 1984.

[11] R.G. Downey and M.R. Fellows. *Parameterized Complexity.* Springer, 1999.

[12] H. Ebbinghaus and J. Flum. *Finite Model Theory.* Perspectives in Mathematical Logic. Springer, second edition, 1999.

[13] H.B. Enderton. *A Mathematical Introduction to Logic.* Harcourt/Academic Press, second edition, 2001.

[14] M. Fitting. Fixpoint semantics for logic programming a survey. *Theoretical Computer Science*, 278(1–2):25–51, May 2002.

[15] A. Gabaldon. Non-markovian control in the situation calculus. In *AAAI-02: Proceedings of the Eighteenth National Conference on Artificial Intelligence*, Menlo Park, California, 2002. AAAI Press.

[16] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[17] M. Genesereth and N.J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, 1987.

[18] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practise.* Elsevier/Morgan Kaufmann, 2004.

[19] G.D. Giacomo, Y. Lespérance, and H.J. Levesque. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[20] G. Gottlob, E. Grädel, and H. Veith. Datalog LITE: A deductive query language with linear time model checking. *ACM Transactions on Computational Logic*, 3(1):42–79, January 2002.

[21] G. Gottlob, F. Scarcello, and M. Sideri. Fixed-parameter complexity in AI and nonmonotonic reasoning. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *LPNMR-99: Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1730 of *Lecture Notes in Computer Science.* Springer, December 1999.

[22] E. Grädel, C. Hirsch, and M. Otto. Back and forth between guarded and modal logics. *ACM Transactions on Computational Logic*, 3(3):418–463, July 2002.

[23] E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *LICS-99: Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, pages 45–54, Washington, D.C., July 1999. IEEE.

[24] W. Hodges. *Model Theory*, volume 42 of *Encyclopedia of Mathematics and its Application*. Cambridge University Press, 1993.

[25] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Texts in Theoretical Computer Science. Cambridge University Press, second edition, 2004.

[26] A. Itai and J.A. Makowsky. Unification as a complexity measure for logic programming. *Journal of Logic Programming*, 4(2):105–117, June 1987.

[27] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to the monadic second order logic. In U. Montanari and V. Sassone, editors, *CONCUR-96: Proceedings of the Seventh International Conference on Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277. Springer, August 1996.

[28] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, December 1983.

[29] H.J. Levesque, R. Reiter, Y. Lespérance, F. Line, and R.B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–84, 1997.

[30] Y. Liu. A hoare-style proof system for robot programs. In *AAAI-02: Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 74–79, Menlo Park, California, 2002. AAAI Press.

[31] Y. Liu and H.J. Levesque. Tractable reasoning in first-order knowledge bases with disjunctive information. In M.M. Veloso and S. Kambhampati, editors, *AAAI-05: Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 639–644. AAAI Press/MIT Press, July 2005.

[32] J. McCarthy. Situations, actions, and causal laws. Technical Report AIM-2, Artificial Intelligence Project, Stanford University, Stanford, California, 1963.

[33] J. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[34] M. Minoux. LTUR: A simplified linear-time unit resolution algorithm for horn formulae and computer implementation. *Information Processing Letters*, 29:1–12, September 1988.

[35] M. Minsky, editor. *Semantic Information Processing*. MIT Press, 1968.

[36] M. Otto. Bisimulation invariance and finite models. In *Logic Colloquium '02*, volume 27 of *Lecture Notes in Logic*. AK Peters, 2005.

[37] D. Poole, A. Mackworth, and R. Goebel. *Computational Intelligence: a Logical Approach.* Oxford University Press, 1998.

[38] S. Popkorn. *First Steps in Modal Logic.* Cambridge University Press, 1994.

[39] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, California, 1991.

[40] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* MIT Press, 2001.

[41] R. Reiter. On knowledge-based programming with sensing in the situation calculus. *ACM Transactions on Computational Logic*, 2(4):433–457, 2001.

[42] S. Russell and P. Norvig. *Artificial Intelligence: a Modern Approach.* Prentice Hall, second edition, 2003.

[43] K. Schneider. *Verification of Reactive Systems: Formal Methods and Algorithms.* Springer, 2004.

[44] S. Shapiro, Y. Lespérance, and H.J. Levesque. The cognitive agents specification language and verification environment for multi-agent systems. In *AAMAS-02: Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems*, pages 19–26, New York, New York, July 2002. ACM IGART, ACM Press.

[45] E. Ternovskaia. Automata theory for reasoning about actions. In T. Dean, editor, *IJCAI-99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 153–158, San Francisco, California, July-August 1999. IJCAI, Inc., Morgan Kaufman.

[46] A. van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, 47(1):185–221, August 1993.