UPPSALA THESES IN COMPUTING SCIENCE
34

Term-Modal Logic and Quantifier-free Dynamic Assignment Logic

Lars Thalmann

Uppsala University
Information Technology
Computing Science Department

Thesis for the Degree of
Doctor of Philosophy

UPPSALA 2000

Term-Modal Logic and Quantifier-free Dynamic Assignment Logic

Lars Thalmann

A Dissertation submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at Computing Science Department, Information Technology, Uppsala University.

Computing Science Department
Information Technology
Uppsala University
Box 337, SE–751 05 Uppsala, Sweden

Uppsala Theses in Computing Science 34
ISSN 0283–359X
ISBN 91–506–1443–6

Dissertation for the Degree of Doctor of Philosophy in Computing Science presented at Uppsala University in 2001

Abstract

Thalmann, Lars 2000: Term-Modal Logic and Quantifier-free Dynamic Assignment Logic. *Uppsala Theses in Computing Science 34*. 140 pp. Uppsala. ISSN 0283–359X, ISBN 91–506–1443–6.

In this dissertation, we present two new sorts of computer science logics.

Many powerful logics exist today for reasoning about multi-agent systems, but in most of these it is hard to reason about an infinite or indeterminate number of agents. Also the naming schemes used in the logics often lack expressiveness to name agents in an intuitive way.

To obtain a more expressive language for multi-agent reasoning and a better naming scheme for agents, we introduce in the first part of the dissertation a family of logics called *term-modal logics*. A main feature of our logics is the use of modal operators indexed by the terms of the logics. Thus, one can *quantify over variables occurring in modal operators*. In term-modal logics agents can be represented by terms, and knowledge of agents is expressed with formulas within the scope of modal operators.

This gives us a flexible and uniform language for reasoning about the agents themselves and their knowledge. We give examples of the expressiveness of the languages and provide sequent-style and tableau-based proof systems for the logics. Furthermore, we give proofs of soundness and completeness with respect to the possible world semantics.

In the second part of the dissertation, we treat another problem in reasoning about multi-agent systems, namely the problem of information updating. We develop a dynamic logic of assignments with a scoping operator instead of quantifiers. Function, relation symbols and logic variables are all rigidly interpreted in our semantics, while program variables are non-rigid. The scoping operator is used to distinguish between the value of a program variable before and after the execution of a program.

We provide a tableau proof system for the logic. First, the system is proved complete without the star operator, and then with the star operator using an omega rule. The full logic is shown to be undecidable, while some interesting fragments are decidable.

Lars Thalmann, Computing Science Department, Information Technology, Uppsala University, Box 337, SE–751 05 Uppsala, Sweden.

`www.LarsThalmann.com`

ISSN 0283–359X
ISBN 91–506–1443–6

Printed by Nina Tryckeri HB, Uppsala 2000

To my parents and my sister

Acknowledgments

I would first of all like to thank my advisor, Prof. Andrei Voronkov, for his advise and guidance, first during the years at Uppsala University, and later during my visits to Manchester University. His suggestions and comments have been of utmost importance for completing the research made in this dissertation and for getting to know what characterize good computer science.

From the summer of 1998, until late 1999, I spent a year visiting Prof. Melvin Fitting at the City University of New York. His suggestions and encouragement were key ingredients in formulating and carrying out the work especially behind the second part of the thesis. The many discussions we have had, have been both fun and interesting, much due to Mel's ability to discuss complicated things in a simple way.

During the work at Uppsala University, Faron Moller has shown me ways to prove things rigorously as well as structured. Our many discussions about computer science, as well as teaching, has taught me a lot.

My appreciation also goes to the present and past members of the Computing Science Department and other departments at Uppsala University: Anatoli, Anders, Arne, Cons, Evgeny, Greger, Göran, Happi, Håkan, Helena, Hessmo, Jan, Joel, Kostis, Margus, Marianne, Marko, Mikael, Monika, Sergei, Sven-Olof, Thomas, Peder, Per, Pierangelo, Plopp, Rafal, Richard, Roland, and all other I have forgotten to mention.

Finally, I would especially like to thank my family and all my friends, for all the great times, at work as well as off work. Special thanks also to Andrei, Carin, Maria and Rafal for proofreading the dissertation.

Contents

List of Figures

Introduction

This dissertation shows two things. Firstly, it shows how modal logic can be extended with modal operators indexed by terms. Variables can occur in the term indexes, which makes it possible to quantify over modal operators. Secondly, it presents a dynamic logic whose expressive power lies between propositional and first-order dynamic logic. Predicates have the same interpretation in every state, but program variables change interpretation between states. The two logics introduced, the *term-modal logic* and the *quantifier-free dynamic assignment logic*, are presented separately in the two main parts of the dissertation.

We have different motivation for the different logics. For the first logic, by introducing term-indexed modal operators, we get an expressive logic, in which we can quantify over an infinite set of modalities and name these modalities in an intuitive way. The main motivation is to describe epistemic systems of dynamic societies with infinite sets of agents. Since the logic is very general, several other applications of the logic are possible, e.g. for planning systems.

For the second logic, our goal is to treat complex actions for state change. This motivates us to look into dynamic first-order logic. In any first-order logic, the quantifiers are a main reason for the undecidability. The system presented here, uses a scoping operator, instead of quantifiers, which preserves much of the expressibility, while providing a more effective proof system. In the logic, the predicate and function symbols are interpreted rigidly, while program variables are non-rigid. The logic can be used in several systems replacing first-order dynamic logic.

Our treatment of the logics proceed in roughly the same way in the two parts of the dissertation. Firstly, we describe the *syntax* or what formulas are well-formed in the logic. Secondly, we introduce the *semantics* or meaning of formulas. Thirdly, we define what constitutes a *proof* in the logic. For the term-modal logics, we present three different proof systems — a sequent calculus, a tableau calculus and a free-variable tableau calculus. For

the quantifier-free dynamic assignment logic, we present a tableau calculus. Lastly, we show that all calculi are sound and complete.

We summarize the main results for each part of the dissertation.

- Part I: Term-Modal Logic
  1. A family of expressive first-order logics with term-indexed modal operators.
  2. Some examples showing that properties of multi-agent systems can be expressed in this logic.
  3. Sequent and tableau calculi.
  4. Free-variable tableau calculus.
  5. Proof of soundness and completeness of all calculi.
- Part II: Quantifier-free Dynamic Assignment Logic
  6. A quantifier-free dynamic logic of assignments.
  7. Proof that the validity problem for the logic is undecidable.
  8. A sound tableau calculus for the logic.
  9. Proof of completeness of the calculus. First for the star-free fragment of the logic, and then for the system resulting from adding an omega-rule for the star operator.

1.1 BACKGROUND

Logic is a formalization of language and reasoning. Traditionally, logic was used to formalize natural languages. Today, logics are used as abstractions of computer languages, and computer programs are formalized and analyzed using logic. The research in this field is vast, and various logics have been developed for specifying, verifying and implementing numerous computer systems.

Propositional logic is the logic of sentences; true or false. *First-order logic* reasons about objects and properties of objects. *Modal logic* is the logic of truth-variance. Something can be true in one context, but false in another. The context might be time-dependent, dependent on the beliefs or opinions of people, or something else.

The study of logic started with Aristotle, continued with Leibniz, and first-order logic was first formulated by Frege (1879). For the rest of the dissertation, we will assume that the reader is familiar with propositional, first-order and modal logic. There are a lot of books on the topic, e.g. (Shoenfield 1967), (Gallier 1986), (van Orman Quine 1974), and (Fitting 1996a) for an introduction on propositional and first-order logic. (Hughes and Cresswell 1984),

(Hughes and Cresswell 1996), (Hughes and Cresswell 1968), (Chellas 1980), and (Fitting 1983) are all good for an introduction on propositional modal logic. And, for first-order modal logic, see (Fitting and Mendelsohn 1998).

There are many versions of modal logic. *Temporal* logic deals with time, *epistemic* with knowledge, *dynamic* logic is the logic of programs, *alethic* deals with necessity and possibility, *deontic* with obligations and permissions, and *doxastic* with belief. Modal logic can in a general sense be interpreted as the logic of adverbials, see e.g. (Fitting and Mendelsohn 1998). The term-modal logics could be used as epistemic, deontic, or doxastic logics, while the quantifier-free dynamic assignment logic is a dynamic logic of programs.

Using modal logic instead of classical logic has many advantages. In many situations, it is easier and more intuitive to express properties using modal operators, compared to using first-order logic, in which one often has to augment predicates with extra arguments for the context, i.e. time, agents, etc.

1.2 OVERVIEW

After examining some related work in Chapter 2, the dissertation continues with the main two parts. Part I introduces the *term-modal logics*, and Part II, the *quantifier-free dynamic assignment logic*. The parts are independent and can be read separately. No definitions or propositions from Part I are used in Part II, and vice versa.

A short version of the term-modal logic part of this dissertation was presented at Tableaux 2000 (Fitting, Thalmann and Voronkov 2000), and a longer version has been accepted to Studia Logica (Fitting, Thalmann and Voronkov 2001). The part on quantifier-free dynamic assignment logic awaits submission.

1.3 TERM-MODAL LOGIC

We describe a new family of modal logics, namely the first-order term-modal logics, where we by *term-modal* mean that any term can be used as a modality. The specific logics we discuss are the term-modal versions of the modal logics $\mathsf{K}, \mathsf{D}, \mathsf{T}, \mathsf{K4}, \mathsf{D4}$, and $\mathsf{S4}$. Sequent-style and tableau-style proof systems for the logics are given, and their soundness and completeness are shown.

Many researchers have been interested in the use of multi-modal logics for knowledge representation (see e.g. Halpern 1993, Fagin, Halpern, Moses and Vardi 1995, Meyer and van der Hoek 1995), although most of them have investigated the use of a finite set of modalities, indexed by the first

n natural numbers, usually denoted either $[1], [2], \ldots, [n]$ or $\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n$. Each number is here naming some agent. By agent we mean any system, e.g. a human or a computer program, to which we can ascribe knowledge. When we instead use an infinite set of modalities we can reason about a dynamic society of agents, where some agents might vanish and new agents may appear.

In the family of multi-modal logics presented in the first part of the dissertation, any term can denote an agent. This makes naming of agents easy and the logics expressive. The use of complex names for agents, possibly involving variables, makes it easy to model a society of agents, and give names to new agents by their relationship to already existing agents. For example, to express that the agent $\mathit{mother}(x)$ thinks (or knows, or believes) that the agent x is good, we can write $[\mathit{mother}(x)]\mathit{good}(x)$.

The standard multi-modal logics allow us to reason about beliefs of particular agents, but provide very limited facilities to reason about beliefs of groups of agents or the agents themselves. In our language, we can distinguish a group of agents by specifying their properties. For example, to express that every Christian believes in the existence of God, we can write $\forall x(\mathit{christian}(x) \supset [x]\exists y\, \mathit{God}(y))$.

An example of a society of agents is the collection of computer processes on some system. Here the logic with its infinite complex naming mechanism can be used to specify requirements of the system as a whole, and the proof system can be used to check that these requirements are satisfied.

When the computer processes spawn new processes, the society of agents (i.e. the number of processes) grows, and the naming mechanism can be used to refer to the newly created processes. As the number of processes spawned by the program need not be known beforehand, it is convenient to have an unlimited set of names for these new agents.

Many researchers have investigated multi-modal logic with a finite set of modalities, and many have discussed naming. We here combine these two aspects into one general family of logics, the family of term-modal logics.

1.4 QUANTIFIER-FREE DYNAMIC ASSIGNMENT LOGIC

Propositional dynamic logic (or propositional modal logic of programs, as it was called then) was introduced by Fischer and Ladner (1977), (1979), following ideas of Pratt (1976).

In this dissertation, we present a variant of dynamic logic, *quantifier-free dynamic assignment logic*, which contrary to propositional dynamic logic

has variables,[1] and contrary to first-order dynamic logic, is quantifier-free, but has a scoping operator.

In the logic we can reason about the values of program variables before and after a program has been executed. An example formula, which we can express in the logic, is

$$\langle a := t \rangle P(a).$$

The intended meaning is that after the assignment $a := t$ has been executed, $P(a)$ is true. Without specifying which a we are referring to in $P(a)$, it is ambiguous what is actually meant by that. Do we mean the value of a before the program $a := t$ has been executed or the value of a after the execution?

In our semantics, we have made the choice that a in $P(a)$ refers to the value of a *after* execution. If we instead want to talk about the value of a *before* execution of $a := t$, we use a scoping operator λ and write

$$\{\lambda x \leftarrow a.\ \langle a := t \rangle P(x)\}.$$

The position of λx in the formula determines which value of a we are referring to. In the formula above, λx stands before the modal operator $\langle a := t \rangle$, so by x in the subformula $P(x)$, we mean the value of a before the program $a := t$ has been executed.

Since our logic use standard dynamic logic constructs, we may express many standard programs. As a simple example, swapping two program variables a and b, can be expressed by the program $(a' := a; a := b; b := a')$. Here semicolon is the composition of programs. Similarly, we have dynamic constructs for choice, tests and iteration.

As an example of iteration and the use of equality in the logic, we express that there exists an execution of $(a := f(a))*$, such that the final value of a equal the value of $f(f(x))$, where x is the starting value of a:

$$\{\lambda x \leftarrow a.\ \langle (a := f(a))* \rangle a = f(f(x))\}.$$

We present a tableau proof system for the logic and show that it is sound and complete. We also prove that the logic with the star operator is undecidable.

[1] Actually of two sorts — program and logic variables.

Related Work

The current trend in modal and description logics is to define expressive, but still decidable, logics. The term-modal logics are undecidable since they contain first-order classical logic. Moreover, the expressiveness of the term-modal logics is, in a way, higher than that of the standard first-order modal logics, since first-order modal logics can be interpreted in term-modal logic, by using a single constant in modal operators (at least for cumulative domains, but our results can be extended to the constant domain versions of the logics as well). Logics with modalities indexed by terms were studied by Grove and Halpern (1991), (1995). These logics are more expressive in some aspects and less expressive in other aspects than the term-modal logics, since these logics can handle equality and agents with special properties. However, there are restrictions on how formulas can be built in these logics, so some well-formed formulas of our logics can not be used as formulas in (Grove and Halpern 1991, Grove 1995). For example, the formula $[x]P(x)$ is not a valid formula in (Grove 1995), since in Grove's framework x in $[x]$ must be of the agent sort, but the formula $P(x)$ in the scope of $[x]$ must not have free variables of the agent sort.

Another related framework is the modal logics with names, see (Passy and Tinchev 1985, Passay and Tinchev 1991, Gargov and Goranko 1993, Blackburn 1993). In these logics a second sort of atomic formula is introduced (these are called names or nominals) and it is stipulated that such a formula is satisfied at a specific world of the model. Intuitively, a nominal names the world in which it is satisfied. Hybrid logics take this a step further. In these logics, nominals are treated as variables open to binding, see e.g. the work by Blackburn and Seligman (1993), Blackburn and Tzakova (1998) and Areces, Blackburn and Marx (2000). A tableau calculus for hybrid logics was presented by Tzakova (1999). The main difference between the logics of our paper and hybrids logics is that in hybrid logic one quantifies over state variables naming worlds, while in this dissertation, we quantify over variables naming accessibility relations.

Propositional dynamic logic (or, as it was called, propositional modal logic of programs) was introduced in (Fischer and Ladner 1977, Fischer and Ladner 1979), following ideas of (Pratt 1976). Completeness of propositional dynamic logic is proven in (Parikh 1978) and (Kozen and Parikh 1981). An interesting thing to notice is that the proof in (Parikh 1978) includes tests, while (Kozen and Parikh 1981) does not. Tableau procedures for propositional dynamic logic are given in (Pratt 1980) and (Pratt 1978). These are extended by Massacci (1998) to include the converse operator.

First-order dynamic logic appears in e.g. (Harel 1979) and (Kozen and Tiuryn 1989). Dynamic logic is a logic with complex modalities just as the term-modal logics. The main difference between term-modal and dynamic logic is the structure of the indexes. In dynamic logic modal operators are indexed by programs, either atomic or composed of subprograms or subformulas joined by modal connectives $;, \cup, *$, or $?$. In term-modal logic the modal operators are indexed with terms. There is a close syntactic connection between the logics. Term-modal logic can be translated into dynamic logic, by translating every modal operator $[t]$ into $[x := t]$ where x is a dummy programming variable. (It is not possible to translate it into quantifier-free dynamic assignment logic, since logic variables can not occur in assignment programs in this logic.) The semantics of the logics are different though. In dynamic logic, the program $x := t$ is usually deterministic, while in the term-modal logic framework, we are interested in having several worlds reachable through by term t. The simplified semantics of the term-modal logic also let us develop the proof theory further. In this dissertation, we provide a free-variable tableau calculus for the term-modal logics.

Modal action logics, see e.g. (Ryan, Fiadeiro and Maibaum 1991), is an example application for the term-modal logics. In (Ryan et al. 1991) a logic for actions is given, but without any proof procedure.

In general, the term-modal logic approach contrasts to other approaches, by considering a simple logic with terms in modal operators and develop a complete free-variable proof system, while other authors either limit the expressibility or fail to provide a complete proof procedure.

Fitting (1983) proves soundness and completeness of (single-)modal logics. In this dissertation we introduce some new definitions, and extend the proofs for term-modal logics.

Fagin et al. (1995), van der Hoek and Meyer (1997) all use modal logics to describe multi-agent systems. Their approaches are based on a finite set of agents, and they also discuss the use of common and distributed knowledge. By using the logic presented in this dissertation, their work might be extended to handle dynamic agent societies with a simple naming mechanism, where quantification over agents is possible.

The basic idea of predicate abstraction was introduced into modal logic by Stalnaker and Thomason (1968), and continued in (Thomason and Stalnaker 1968). (Bressan 1972) gave an extensive development. Other modal applications of predicate abstraction appear in e.g. (Fitting 1972, Fitting 1973, Fitting 1975), and more recently in (Fitting 1991, Fitting 1996b, Fitting and Mendelsohn 1998).

Part I

Term-Modal Logic

Introduction

In this part, we describe a new family of modal logics, namely the first-order term-modal logics, where we by *term-modal* mean that any term can be used as a modality. The specific logics we discuss are the term-modal versions of the modal logics K, D, T, $\mathsf{K4}$, $\mathsf{D4}$, and $\mathsf{S4}$. Sequent-style and tableau-style proof systems for the logics are given, and their soundness and completeness are shown.

3.1 OVERVIEW

Part I is structured as follows: Chapter 4 defines the syntax of term-modal logics, and Chapter 5 covers their semantics. In Chapter 6 we introduce two different types of proof systems, in Section 6.1, sequent calculi and in Section 6.2, tableau calculi for these logics. In Chapter 7 we establish soundness of the sequent calculi. In Chapter 8 we give the completeness proof. The proof is rather lengthy and split between two sections. In Section 8.1 we define the main technical tool used for the completeness proof, the so-called *consistency property*, and prove a Model Existence Theorem for consistency properties. Using this theorem, we establish completeness of the sequent calculi in Section 8.2. Soundness and completeness of the sequent calculi implies soundness and completeness of the tableau calculi for term-modal logics. As a step toward automated reasoning in term-modal logic in Chapter 9 we introduce free-variable versions of tableau calculi for term-modal logics. In Section 9.1 we give an example refutation in such a calculus, in order to illustrate some distinctive features of free-variable calculi for term-modal logics.

Syntax

The term-modal logics are obtained from the standard predicate modal logics by adding modal operators indexed by terms. In this chapter we give a formal definition of the syntax of term-modal logics.

We assume a *signature* Σ consisting of three disjoint sets of constants, function symbols and relation symbols. Usually, the signature is assumed to be fixed, but in some situations we will vary it for technical convenience. In addition to the symbols of Σ, we will use various infinite *sets P of parameters* disjoint from the symbols in Σ. In some situations parameters will behave as new constants, in others as elements of a domain on which formulas are evaluated. The signature is not necessarily finite or countable. For every signature Σ, we denote by Σ^- the signature obtained from Σ by omitting all constants and function symbols.

Definition 1 (Term) Suppose P is a set of parameters and V a set of *variables* disjoint from the set of parameters. The set of *terms of the signature Σ with parameters in P* and variables in V, denoted $\mathcal{T}(\Sigma \cup P, V)$ is defined inductively as follows.

1. Each constant in Σ is a term.
2. Each variable in V is a term.
3. Each parameter in P is a term.
4. If $t_1, \ldots, t_n$ are terms and f is an n-ary function symbol, then $f(t_1, \ldots, t_n)$ is a term.

In this part, we can restrict ourselves to a fixed set of variables, however the set of parameters (and sometimes the signature) will vary. So we will use a simpler notation $\mathcal{T}(\Sigma \cup P)$. A term is called *ground* if it has no occurrences of variables.

Definition 2 (Formula) Let P be a set of parameters. The set of *formulas of the signature* Σ *with parameters in* P, denoted $\mathcal{F}(\Sigma \cup P)$, is defined inductively as follows.

1. If R is a relation symbol of arity n and $t_1, \ldots, t_n$ are terms in $\mathcal{T}(\Sigma \cup P)$, then $R(t_1, \ldots, t_n)$ is an *atomic formula*. Any atomic formula is a formula.
2. If A and B are formulas, then so are $(A \wedge B)$, $(A \vee B)$, $(A \supset B)$ and $\neg A$.
3. If A is a formula and t is a term in $\mathcal{T}(\Sigma \cup P)$, then $[t]A$ and $\langle t \rangle A$ are formulas.
4. If A is a formula and x is a variable, then $\forall x A$ and $\exists x A$ are formulas.

The notions of *free and bound occurrences of variables* are defined as usual, with the exception of the following item:

- The free occurrences of variables in $[t]A$ and $\langle t \rangle A$ are all occurrences of variables in t plus all free occurrences of variables in A.

A formula is called *closed*, or a *sentence* if it has no free occurrences of variables (but note that it may contain parameters). An $\exists$*-formula* is any formula $\exists x A$. A *literal* is either an atomic formula A or its negation $\neg A$. Literals A and $\neg A$ are said to be *complementary* to each other.

Intuitively, when interpreting the formulas in a multi-agent context, the meaning of the formula $[t]A$ is that the agent denoted by the term t knows (believes etc.) the information represented by the formula A. The formula $\langle t \rangle A$ intuitively means that the agent denoted by t considers it possible that A holds, i.e. it is not the case that the agent knows the contrary (which can also be expressed by $\neg[t]\neg A$).

In the proof systems introduced later we make no assumptions about the kind of knowledge expressed. The knowledge could in fact be only beliefs, i.e. an agent might believe something which is false.

It is easy to add axioms of knowledge, if one is interested in describing a specific kind of knowledge. An example of this is the knowledge axiom, $[x]A \supset A$, which intuitively means that if an agent x knows something, then it is true. More about different epistemic interpretations of modal logic can be found in e.g. the books by Hintikka (1962) or Lenzen (1978).

The distinctive feature of our logics is the possibility to express knowledge of agents and properties of agents themselves in one language. For example, we can write

$$\forall x(human(x) \supset [x]good(x))$$

to express that everyone knows that he/she is good. Note the use of quantification over agents.

Notation 3 For the rest of Part I, we will denote

- variables by x, y, z, u, v;
- parameters by p;
- terms by s, t;
- domain elements by d (i.e. elements in the set $\mathbf{D}$, which is defined later);
- formulas by A, B, C;
- sets of formulas by S, Ψ;
- sets of parameters of the language by P;
- literals by L;
- logics $\mathsf{K}, \mathsf{D}, \mathsf{T}, \mathsf{K4}, \mathsf{D4}, \mathsf{S4}$ by the generic symbol $\mathcal{L}$.

We write $A(x)$ to denote a formula A with zero or more free occurrences of the variable x and write $A(t)$ to denote the replacement of all free occurrences of x by a term t. Before the replacement, we rename in $A(x)$ all bound occurrences of variables that have free occurrences in t.

SEMANTICS

In this chapter, we describe a possible world semantics for the logics. The semantics is defined through the notions of *frames* and *structures*. It differs from the standard semantics of first-order modal logics by the treatment of the reachability relation on worlds: the reachability relation is indexed by elements of the domain. We assume all definitions be given w.r.t. a nonempty set **D**, called the *domain*.

5.1 FRAMES

Definition 4 (Frame) A *frame* over **D** is a triple $\langle \mathcal{W}, \mathcal{D}, \longrightarrow \rangle$, where

1. $\mathcal{W}$ is a non-empty set, called the *set of possible worlds*.
2. $\mathcal{D}$ is a mapping from $\mathcal{W}$ to the set of subsets of **D**. The set $\mathcal{D}(w)$ is denoted by $\mathcal{D}_w$ and called the *domain of* w.
3. $\longrightarrow$ is a relation on $\mathcal{W} \times \mathbf{D} \times \mathcal{W}$, called the *accessibility relation*. If $\longrightarrow (w_1, d, w_2)$, then we say that w_2 is *d-reachable* from w_1 and write $w_1 \stackrel{d}{\longrightarrow} w_2$.

We require the *monotonicity condition* to be satisfied in all frames:

$$\text{If } w_1 \stackrel{d}{\longrightarrow} w_2 \text{ then } \mathcal{D}_{w_1} \subseteq \mathcal{D}_{w_2}.^{1}$$

The monotonicity condition corresponds to *cumulative domains* (e.g. Wallen 1990) and *nested domains* (e.g. Garson 1984).

[1] The monotonicity condition can be replaced by a weaker condition:

$$\text{If } w_1 \stackrel{d}{\longrightarrow} w_2 \text{ and } d \in w_1 \text{ then } \mathcal{D}_{w_1} \subseteq \mathcal{D}_{w_2}.$$

The reason is that the first-order language can not express properties of worlds d-reachable from w, when $d \notin \mathcal{D}_w$.

Definition 5 ($\mathcal{L}$-frame) We specialize the concept of frames to six different classes:

- K. All frames are K-*frames*.
- D. If for all d and w there exists w' such that $w \xrightarrow{d} w'$ (i.e. the accessibility relation is *serial* in its 1st and 3rd arguments) then the frame is a D-*frame*.
- T. If $w \xrightarrow{d} w$ holds for all w and d (i.e. the accessibility relation is *reflexive* in its 1st and 3rd arguments), then the frame is a T-*frame*.
- K4. If, from $w \xrightarrow{d} w'$ and $w' \xrightarrow{d} w''$ it follows that $w \xrightarrow{d} w''$ for all d and $w, w', w'' \in \mathcal{W}$, (i.e. the accessibility relation is *transitive* in its 1st and 3rd arguments) then it is a K4-*frame*.
- D4. If the accessibility relation is both serial and transitive in its 1st and 3rd arguments, then the frame is a D4-*frame*.
- S4. If the accessibility relation is both reflexive and transitive in its 1st and 3rd arguments, then the frame is an S4-*frame*.

5.2 FIRST-ORDER MODAL STRUCTURES

The first-order modal (Kripke) structures are introduced in the standard way, except for the case of modal operators.

Definition 6 (Structure for $\mathcal{L}$) Let $\mathcal{L}$ be one of K, D, T, K4, D4, S4. A *first-order modal structure* for $\mathcal{L}$, or simply $\mathcal{L}$-*structure* over a domain $\mathbf{D}$ is a tuple $\mathfrak{S} = \langle \mathcal{W}, \mathcal{D}, \longrightarrow, I, \Vdash \rangle$, where

1. $\langle \mathcal{W}, \mathcal{D}, \longrightarrow \rangle$ is a $\mathcal{L}$-frame over $\mathbf{D}$.
2. $\Vdash$ is a binary relation between worlds and atomic sentences in $\mathcal{F}(\Sigma^- \cup \mathbf{D})$. (Note that elements of $\mathbf{D}$ are treated as parameters in $\mathcal{F}(\Sigma^- \cup \mathbf{D})$.)
3. I, called the *interpretation function*, is a mapping that maps every constant c of Σ to an element of $\mathbf{D}$ and every function symbol f of Σ of arity n to an n-place function on $\mathbf{D}$. The corresponding element of $\mathbf{D}$ and function on $\mathbf{D}$ are called the *interpretations* of c and f respectively. We require the interpretation of any constant and function symbol to be totally defined in every world: this means that $I(c)$ belongs to $\mathcal{D}_w$ for every world $w \in \mathcal{W}$ and for every $d_1, \ldots, d_n \in \mathcal{D}_w$ we have $I(f)(d_1, \ldots, d_n) \in \mathcal{D}_w$.

Note that $\Vdash$ is only defined for formulas without function symbols or constants, but with parameters in $\mathbf{D}$.

We call a *valuation* V *in a structure* $\mathfrak{S}$ any mapping $V : P \to \mathbf{D}$ from a set of parameters to the domain $\mathbf{D}$ of $\mathfrak{S}$. Any valuation V can be extended to the set of all ground terms by defining

$$\begin{array}{rcl} V(c) & = & I(c); \\ V(f(t_1, \ldots, t_n)) & = & I(f)(V(t_1), \ldots, V(t_n)). \end{array}$$

Now we can give the central notion of satisfiability of formulas in structures.

Given a first-order modal structure $\langle \mathcal{W}, \mathcal{D}, \longrightarrow, I, \Vdash \rangle$, we change the relation $\Vdash$ into a ternary relation between worlds in $\mathcal{W}$, valuations, and sentences in $\mathcal{F}(\Sigma \cup \mathbf{D})$ as given below. We write $\mathfrak{S}, w, V \Vdash A$ when this relation holds on w, V, A and denote by $\nVdash$ the complement of $\Vdash$. When we use this notation, we can omit one or both of $\mathfrak{S}, V$, when they are clear from the context.

Definition 7 (Relation $\Vdash$) Given $\mathfrak{S}$ and V, we define the relation $\Vdash$ as follows.

1. $w, V \Vdash R(t_1, \ldots, t_n)$ if $w \Vdash R(V(t_1), \ldots, V(t_n))$.
2. $w, V \Vdash A \wedge B$ if $w, V \Vdash A$ and $w, V \Vdash B$.
3. $w, V \Vdash A \vee B$ if $w, V \Vdash A$ or $w, V \Vdash B$.
4. $w, V \Vdash A \supset B$ if $w, V \nVdash A$ or $w, V \Vdash B$.
5. $w, V \Vdash \neg A$ if $w, V \nVdash A$.
6. $w, V \Vdash [t]A$ if for all w' such that $w \stackrel{V(t)}{\longrightarrow} w'$ we have $w', V \Vdash A$.
7. $w, V \Vdash \langle t \rangle A$ if there exists w' such that $w \stackrel{V(t)}{\longrightarrow} w'$ and $w', V \Vdash A$.
8. $w, V \Vdash \forall x A(x)$ if $w, V \Vdash A(d)$, for all $d \in \mathcal{D}_w$.
9. $w, V \Vdash \exists x A(x)$ if $w, V \Vdash A(d)$, for some $d \in \mathcal{D}_w$.

Definition 8 (Truth, satisfiability) Let $\mathfrak{S} = \langle \mathcal{W}, \mathcal{D}, \longrightarrow, I, \Vdash \rangle$ be a structure. We say a formula A is *true*, or *holds*, or is *locally satisfied* in $\mathfrak{S}$ at a world $w \in \mathcal{W}$ *under a valuation* V if $\mathfrak{S}, w, V \Vdash A$. A formula A is *globally satisfied* in a structure $\mathfrak{S}$ under a valuation V if it is locally satisfied at every world of $\mathfrak{S}$ under V. A formula A is called *locally (respectively, globally) satisfiable in* $\mathfrak{S}$ if it is locally (respectively, globally) satisfied in $\mathfrak{S}$ under some valuation. If A is locally satisfiable in $\mathfrak{S}$ we also say that $\mathfrak{S}$ is a *model* of A.

Note that the truth of a formula A under a valuation V only depends on the value of V on the parameters occurring in A. Thus, if A is a sentence in $\mathcal{F}(\Sigma)$, its truth does not depend on the valuation at all.

Definition 9 (Model, validity) Let $\mathcal{L}$ be one of K, D, T, K4, D4, S4. We call a model $\langle \mathcal{W}, \mathcal{D}, \longrightarrow, I, \Vdash \rangle$ of a formula A an *$\mathcal{L}$-model* if its frame $\langle \mathcal{W}, \mathcal{D}, \longrightarrow \rangle$ is an $\mathcal{L}$-frame. A formula A is called *$\mathcal{L}$-satisfiable* if it has an $\mathcal{L}$-model. A formula A is called *$\mathcal{L}$-valid* if it is true in every world of every $\mathcal{L}$-structure under every valuation.

It is not hard to argue that satisfiability and validity are dual notions in the following sense: a formula A is unsatisfiable if and only if $\neg A$ is valid. In view of this duality we will formulate our results in terms of (un)satisfiability only.

When we speak of a *logic* $\mathcal{L}$ in this part, we understand the set of $\mathcal{L}$-valid formulas. So, we will speak of *logics* K, D, T, K4, D4, S4. Another standard way of introducing a logic is to define a suitable *calculus* deriving valid formulas in this logic. In the next section we introduce such calculi for all these logics.

Formulas A and B are called *$\mathcal{L}$-equivalent* if the formulas $A \supset B$ and $B \supset A$ are $\mathcal{L}$-valid. It is evident that in any context when we speak about worlds, structures, valuations, and satisfiability, we can replace formulas by equivalent ones.

We will now introduce the negation normal form of formulas, which will simplify our proofs considerably.

Definition 10 (Negation normal form) A formula A is said to be in *negation normal form* if it is constructed from literals using $\wedge$, $\vee$, $\forall$, $\exists$, $[t]$ and $\langle t \rangle$. A formula B is called a *negation normal form of a formula* A, if B is in negation normal form and B is equivalent to A.

Lemma 11 Every formula A has a negation normal form.

PROOF. It is not hard to argue that one can reduce A to its negation normal form by means of the transformations shown in Figure 5.1. These transformations replace, in any order, subformulas of A on the left of $\Rightarrow$ by the corresponding subformulas on the right, until no transformation is applicable. □

$$
\begin{array}{rcl}
B \supset C & \Rightarrow & \neg B \vee C \\
\neg\neg B & \Rightarrow & B \\
\neg(B \vee C) & \Rightarrow & \neg B \wedge \neg C \\
\neg(B \wedge C) & \Rightarrow & \neg B \vee \neg C \\
\neg\forall x B & \Rightarrow & \exists x \neg B \\
\neg\exists x B & \Rightarrow & \forall x \neg B \\
\neg[t]B & \Rightarrow & \langle t\rangle \neg B \\
\neg\langle t\rangle B & \Rightarrow & [t]\neg B
\end{array}
$$

Figure 5.1: Negation normal form transformation

Proof Systems

6.1 SEQUENT CALCULI

In this section we define sequent calculi for the family of term-modal logics. There are several essentially equivalent notions of sequent giving rise to different calculi. The original definition of Gentzen (1934) defines sequents as expressions $A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$, where $A_1, \ldots, A_n$, $B_1, \ldots, B_m$ are formulas. Smullyan (1963) represents such a sequent as a collection of formulas $A_1, \ldots, A_n, \neg B_1, \ldots, \neg B_m$ or a collection $T\ A_1, \ldots, T\ A_n, F\ B_1, \ldots, F\ B_m$ of *signed formulas* with the intended meaning that all formulas A_i are true and all formulas B_j are false and introduces a *uniform notation* to group together inference rules with similar behavior. We will use the approach formulated by Schütte (1960). Instead of using arbitrary formulas we will use only formulas in *negation normal form*. Every rule in the uniform notation corresponds to an inference rule introducing a particular connective on formulas in negation normal form. Then we do not need the unifying notation anymore, since we can label inference rules by the corresponding connectives. We will use parameters instead of free variables, and therefore only deal with sentences.

Definition 12 (Sequent) A *sequent* is a set of sentences. Let S be a sequent and $\mathfrak{S}$ be a structure. We say that a sequent S is *locally satisfied* in $\mathfrak{S}$ at a world $w \in \mathcal{W}$ *under a valuation* V if $\mathfrak{S}, w, V \Vdash A$ for all $A \in S$. A sequent S is *globally satisfied* in a structure $\mathfrak{S}$ under a valuation V if S is locally satisfied at every world of $\mathfrak{S}$ under V. A sequent S is called *locally (respectively, globally) satisfiable in* $\mathfrak{S}$ if it is locally (respectively, globally) satisfied in $\mathfrak{S}$ under some valuation. If S is locally satisfiable in $\mathfrak{S}$ we also say that $\mathfrak{S}$ is a *model* of S.

Thus, a sequent is understood as a (possibly infinite) conjunction of its members.

For a formula A and a set of formulas S we use A, S or S, A to denote the set $S \cup \{A\}$. Likewise, we write S_1, S_2 to denote the union of two sequents $S_1 \cup S_2$.

Sequent calculi for logics K, D, T, K4, D4, S4 are shown in Figure 6.1. $S^{[t]}$ is a generalization of the notation used in (Fitting 1983). Semantically, $S^{[t]}$ denotes the set of formulas which must hold in every world that is $V(t)$-reachable from the world in which S holds. Depending on the logic, semantic restrictions on the frame make the definition of $S^{[t]}$ vary between logics.

In the rule (ax), A is atomic. We can generalize the calculus for non-atomic axioms in the standard way, but the calculus is complete with atomic axioms.

Definition 13 (Inference, derivation, refutation) The *inference rules* of the sequent calculi are shown in Figure 6.1. We call an *inference* any particular instance of an inference rule. The *premises* of any inference or inference rule are the sequents above the bar; its *conclusion* is the sequent below the bar. An *axiom* is any conclusion of (ax). A *derivation* of a sequent S is a tree made of inferences and having S as the root. A derivation is called a *refutation* if all leaves in it are axioms.

We use the term *refutation* instead of the term proof because the sequent calculi used in this part establish unsatisfiability rather than validity.

Example 14 Suppose that we wish to establish K-validity of the sentence

$$\forall z([z]\forall x R(x) \supset \forall y[z]R(y)).$$

We turn this formula into its negation

$$\neg\forall z([z]\forall x R(x) \supset \forall y[z]R(y))$$

and establish the unsatisfiability of the latter. To this end, we transform this formula into its negation normal form

$$\exists z([z]\forall x R(x) \wedge \exists y\langle z\rangle\neg R(y))$$

and try to find a refutation in the sequent calculus for K. An example refutation is as follows.

For all logics (K, D, T, K4, D4, S4):

$$\frac{}{S, A, \neg A}\ (\text{ax})$$

$$\frac{S, A \quad S, B}{S, A \vee B}\ (\vee) \qquad \frac{S, A, B}{S, A \wedge B}\ (\wedge)$$

$$\frac{S, A(p)}{S, \exists x A(x)}\ (\exists)^* \qquad \frac{S, \forall x A(x), A(t)}{S, \forall x A(x)}\ (\forall)$$

$$\frac{S^{[t]}, A}{S, \langle t \rangle A}\ (\langle t \rangle)$$

For serial logics (D, D4):

$$\frac{S^{[t]}}{S}\ ([t])$$

For reflexive logics (T, S4):

$$\frac{S, A}{S, [t]A}\ ([t])$$

Logic $\mathcal{L}$	**Definition of $S^{[t]}$**
K, D, T	$S^{[t]} = \{A \mid [t]A \in S\}$
K4, D4	$S^{[t]} = \{A \mid [t]A \in S\} \cup \{[t]A \mid [t]A \in S\}$
S4	$S^{[t]} = \{[t]A \mid [t]A \in S\}$

* The rule ($\exists$) satisfies the *parameter condition*: p is a parameter having no occurrences in the conclusion of the rule.

Figure 6.1: Sequent calculi

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall xR(x), R(q), \neg R(q)}\,(\text{ax})}{\forall xR(x), \neg R(q)}\,(\forall)}{[p]\forall xR(x), \langle p\rangle\neg R(q)}\,(\langle p\rangle)}{[p]\forall xR(x), \exists y\langle p\rangle\neg R(y)}\,(\exists)}{[p]\forall xR(x) \wedge \exists y\langle p\rangle\neg R(y)}\,(\wedge)}{\exists z([z]\forall xR(x) \wedge \exists y\langle z\rangle\neg R(y))}\,(\exists)$$

This refutation is also a valid refutation in D and T.

To obtain a refutation in K4 and D4, we have to modify the top part of this refutation because of the difference in the definition of $S^{[t]}$:

$$\dfrac{\dfrac{\dfrac{}{[p]\forall xR(x), \forall xR(x), R(q), \neg R(q)}\,(\text{ax})}{[p]\forall xR(x), \forall xR(x), \neg R(q)}\,(\forall)}{[p]\forall xR(x), \langle p\rangle\neg R(q)}\,(\langle p\rangle)$$

A refutation in S4 follows a different strategy because of the difference in the $([t])$ rule:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\forall xR(x), R(q), \neg R(q)}\,(\text{ax})}{\forall xR(x), \neg R(q)}\,(\forall)}{[p]\forall xR(x), \neg R(q)}\,([p])}{[p]\forall xR(x), \langle p\rangle\neg R(q)}\,(\langle p\rangle)$$

Since every formula has a negation normal form, we can restrict ourselves to negation normal forms. Moreover, using the fact that a formula and its negation normal form obtained by the transformation of Figure 5.1 have, in a sense, *similar structure*, it is not hard to change sequent calculi introduced below for formulas in negation normal forms, into calculi for arbitrary formulas or signed formulas. For example, consider the transformation

$$B \supset C \Rightarrow \neg B \vee C.$$

Using the similarity between $\supset$ and $\vee$, we can turn a sequent calculus inference rule for $\vee$

$$\dfrac{S, A \quad S, B}{S, A \vee B}\,(\vee)$$

into a structurally similar sequent calculus rule for $\supset$:

Uniform rule notation	Our notation
(α)	$(\wedge)$
(β)	$(\vee)$
(γ)	$(\forall)$
(δ)	$(\exists)$
(π)	$([t])$
(ν)	$(\langle t \rangle)$

Figure 6.2: The correspondence between the uniform notation and rules of sequent calculi for $\mathcal{L}$

$$\frac{S, \neg A \quad S, B}{S, A \supset B} \ (\supset).$$

or a structurally similar sequent calculus rule for $T \supset$ in the system for signed formulas:

$$\frac{S, F\ A \quad S, T\ B}{S, T\ A \supset B} \ (T \supset).$$

So, we will not concern ourselves with arbitrary formulas anymore, but rather deal with formulas in negation normal form.

The correspondence between our rules and the uniform notation of Smullyan (1963) used in (Fitting 1983, Fitting 1996a) and (Wallen 1990) is given in Figure 6.2.

We will augment the logics defined above with so-called *global assumptions*. Let Ψ be a set of sentences and $\mathcal{L}$ be one of the logics defined above. We call a *sequent calculus for $\mathcal{L}$ with global assumptions Ψ* the calculus obtained from $\mathcal{L}$ by adding the rule

$$\frac{S, A}{S} \ (\Psi),$$

where $A \in \Psi$.

6.2 TABLEAU SYSTEMS

Tableau systems formalize proof-search in sequent calculi. Tableaux are often introduced as trees of formulas, with inference rules on tableaux formulated in terms of *branches*. To simplify the presentation, we introduce tableaux as multisets of branches.

Definition 15 (Tableau, branch, empty branch) A *tableau* is a finite multiset $S_1, \ldots, S_n$ of sequents, denoted $S_1 \mid \cdots \mid S_n$. The *empty tableau* is denoted by $\#$. Every sequent S_i is called a *branch* of this tableau.

The tableau calculus for each logic studied in this paper can be obtained by a simple transformation of the corresponding sequent calculus. For every inference rule

$$\frac{S_1 \quad \cdots \quad S_n}{S}$$

of the sequent calculus, the corresponding tableau rule has the form

$$\frac{S \mid \mathcal{T}}{S_1 \mid \cdots \mid S_n \mid \mathcal{T}}$$

where $\mathcal{T}$ is any tableau. Note the reverse order of the sequents. The tableau calculus rules have the following intuitive meaning: suppose that we search for a refutation of S and all sequents in $\mathcal{T}$. Then, since there is a sequent calculus rule reducing S to the sequents $S_1, \ldots, S_n$, it is enough to find a refutation of $S_1, \ldots, S_n$ and all sequents in $\mathcal{T}$. To find a refutation for a formula A, we begin with a tableau consisting of one branch A and try to apply the tableau rules until no (unrefuted) branches remain.

Formally, the *tableau calculi* for $\mathcal{L}$ are shown in Figure 6.3.

Theorem 16 (Equivalence of tableau calculi and sequent calculi) A sequent S has a refutation in the sequent calculus for $\mathcal{L}$ (with global assumptions Ψ) iff a derivation of $\#$ from S exists in the tableau calculus for $\mathcal{L}$ (with the global assumptions Ψ).

For all logics (K, D, T, K4, D4, S4):

$$\frac{S, A, \neg A \mid \mathcal{T}}{\mathcal{T}} \; |\mathrm{ax}|$$

$$\frac{S, A \vee B \mid \mathcal{T}}{S, A \mid S, B \mid \mathcal{T}} \; |\vee| \qquad \frac{S, A \wedge B \mid \mathcal{T}}{S, A, B \mid \mathcal{T}} \; |\wedge|$$

$$\frac{S, \exists x A(x) \mid \mathcal{T}}{S, A(p) \mid \mathcal{T}} \; |\exists|^* \qquad \frac{S, \forall x A(x) \mid \mathcal{T}}{S, \forall x A(x), A(t) \mid \mathcal{T}} \; |\forall|$$

$$\frac{S, \langle t \rangle A \mid \mathcal{T}}{S^{[t]}, A \mid \mathcal{T}} \; |\langle t \rangle|$$

For serial logics (D, D4):

$$\frac{S \mid \mathcal{T}}{S^{[t]} \mid \mathcal{T}} \; |[t]|$$

For reflexive logics (T, S4):

$$\frac{S, [t] A \mid \mathcal{T}}{S, A \mid \mathcal{T}} \; |[t]|$$

For systems with global assumptions Ψ:

$$\frac{S \mid \mathcal{T}}{S, A \mid \mathcal{T}} \; |\Psi|^*$$

Here $S^{[t]}$ is defined in the same way as for the sequent calculi.

* The rule $|\exists|$ satisfies the *parameter condition*: p is a parameter having no occurrences in the premise of the rule. In the rule $|\Psi|$, $A \in \Psi$.

Figure 6.3: Tableau calculi

Soundness

The aim of this chapter is to prove soundness of the introduced sequent calculi. Soundness of the tableau calculi will immediately follow by Theorem 16.

Theorem 17 (Soundness of sequent calculi) If a sequent has a refutation in the sequent calculus for $\mathcal{L}$, then it is $\mathcal{L}$-unsatisfiable.

Proof. The proof is by induction on the number of inferences in the refutation. The smallest refutations are simply the axioms $S, A, \neg A$. Evidently, this sequent has no model. Take any longer refutation and consider the bottom inference of this refutation

$$\frac{S_1 \quad \cdots \quad S_n}{S} \ . \tag{7.1}$$

If we prove that any $\mathcal{L}$-model of S is also a $\mathcal{L}$-model for some S_i, then we are done, since all S_i have shorter refutations than S and by the induction hypothesis can not have $\mathcal{L}$-models.

So we now assume that $\mathfrak{S}$ is a $\mathcal{L}$-model of S and prove that it is also a model of some S_i. The proof is by the case analysis on the inference rule used in inference (7.1). The proof is standard for most cases, so we only consider two rules ($[t]$) and ($\langle t \rangle$). In the proof, let $[t]S$ denote the set $\{[t]A \mid A \in S\}$.

Case : rule ($\langle t \rangle$) for logics K, D and T. The rule has the form

$$\frac{S_2, A}{S_1, [t]S_2, \langle t \rangle A} \ (\langle t \rangle)$$

for some S_1, S_2. We assume that there exists a structure $\mathfrak{S}$ and valuation V under which for some world w we have $w \Vdash S_1, [t]S_2, \langle t \rangle A$ and show that the sequent S_2, A is satisfiable in $\mathfrak{S}$. Since $w \Vdash \langle t \rangle A$, there exists a world w' such that $w \stackrel{V(t)}{\longrightarrow} w'$ and $w' \Vdash A$. By $w \Vdash [t]S_2$ and $w \stackrel{V(t)}{\longrightarrow} w'$ we also have $w' \Vdash S_2$, and therefore $w' \Vdash S_2, A$.

CASE : rule ($[t]$) for logic D4. The rule has the form

$$\frac{S_2, [t]S_2}{S_1, [t]S_2}\ (\langle t\rangle)$$

We assume that there exists a D4-structure $\mathfrak{S}$, world w and valuation V such that under $\mathfrak{S}$ and V we have $w \Vdash S_1, [t]S_2$ and show that the sequent $S_2, [t]S_2$ is satisfiable in $\mathfrak{S}$.

Since $\mathfrak{S}$ is a D4-structure, there exists a world w' such that $w \xrightarrow{V(t)} w'$. By $w \Vdash [t]S_2$ and $w \xrightarrow{V(t)} w'$ we have $w' \Vdash S_2$.

Consider any world w'' such that $w' \xrightarrow{V(t)} w''$. Since $\mathfrak{S}$ is a D4-structure, we have $w \xrightarrow{V(t)} w''$. This together with $w \Vdash [t]S_2$ gives us $w'' \Vdash S_2$. Since w'' was an arbitrary world satisfying $w' \xrightarrow{V(t)} w''$, we get $w' \Vdash [t]S_2$. Thus, $w' \Vdash S_2, [t]S_2$.

□

Completeness

Now our aim is to prove completeness of the sequent calculi.

Theorem 18 (Completeness of sequent calculi) Let S be a set of sentences. If S has no refutation in the sequent calculus for $\mathcal{L}$ with the global assumptions Ψ, then there exists an $\mathcal{L}$-structure $\mathfrak{S}$ and a valuation V under which S is locally satisfied and all formulas in Ψ are globally satisfied.

We will build a $\mathcal{L}$-model for a sequent with no refutation using the construction of Fitting (1983). The construction is roughly as follows. First we define an abstract property capturing the syntactic counterpart of satisfiability, called the *consistency property*. Then we show that the family of non-refutable sets of formulas is such a consistency property. The completeness is finally proved by showing that each set of formulas in this consistency property must be satisfiable. This means that every non-refutable sequent has a model. We will also establish a stronger form of completeness for calculi with global assumptions Ψ: in the constructed $\mathcal{L}$-model all formulas in Ψ will be satisfied *globally*, i.e. in *every* world. Our construction differs from that of Fitting (1983) in several respects. Firstly, the new modal operators require a special treatment. Secondly, our logic has function symbols which were not treated in (Fitting 1983). Thirdly, we simplified the construction of Fitting (1983) in several respects.

We will use the fact that valuations do not change in different worlds, and prove the existence of a special kind of model: a structure in which (i) the domain consists of all ground terms in $\mathcal{T}(\Sigma \cup P)$, where P is a set of parameters, and (ii) each term is evaluated to itself. We call such structures *Herbrand structures*.

Definition 19 (Herbrand structure) Let $\mathfrak{S} = \langle \mathcal{W}, \mathcal{D}, \longrightarrow, I, \Vdash \rangle$ be a structure over a domain $\mathbf{D}$ of a signature Σ. Then $\mathfrak{S}$ is called a *Herbrand structure* if (i) $\mathbf{D}$ is the set of ground terms of $\mathcal{T}(\Sigma \cup P)$ for some set of parameters P and (ii) for the interpretation function I the following holds

1. for every constant c, $I(c) = c$;
2. for every function symbol f and terms $t_1, \ldots, t_n$, $I(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$;

Note the following two properties of Herbrand structures.

1. The requirement on functions to be totally defined in a world has a consequence on the structure of the worlds for Herbrand structures: if the domain $\mathcal{D}_w$ of a world w contains parameters $p_1, \ldots, p_n$, it also contains all terms built using the function symbols of Σ and parameters $p_1, \ldots, p_n$.
2. If a valuation V in a Herbrand structure is the identity function on the set of parameters, i.e. $V(p) = p$ for all p, then also $V(t) = t$ for every ground term $t \in \mathcal{T}(\Sigma \cup P)$.

8.1 MODEL EXISTENCE

In the proof of the Model Existence Theorem below we will construct a Herbrand structure.

In the proofs below we will assume that we have a set of parameters P, which has the same cardinality as the set of closed formulas of Σ. If, for instance, the number of constants, function symbols and relation symbols are all countable, then we also assume the set P to be countable.

Definition 20 (Consistency property) A set of sequents $\mathbf{C}$ is called a (first-order) *$\mathcal{L}$-consistency property* if, for each $S \in \mathbf{C}$,

(A) S contains no atomic formula A and its negation $\neg A$.

($\wedge$) If $A \wedge B \in S$, then $S \cup \{A, B\} \in \mathbf{C}$.

($\vee$) If $A \vee B \in S$, then $S \cup \{A\} \in \mathbf{C}$ or $S \cup \{B\} \in \mathbf{C}$.

($\langle t \rangle$) If $\langle t \rangle A \in S$, then $S^{[t]} \cup \{A\} \in \mathbf{C}$.

($[t]$) (a) For logics K and $\mathsf{K4}$ no other conditions.
(b) For T and $\mathsf{S4}$, if $[t]A \in S$, then $S \cup \{A\} \in \mathbf{C}$.
(c) For D and $\mathsf{D4}$, if $S \in \mathbf{C}$, then $S^{[t]} \in \mathbf{C}$.

($\forall$) If $\forall x A(x) \in S$, then $S \cup \{A(t)\} \in \mathbf{C}$ for *every* ground term t.

($\exists$) If $\exists x A(x) \in S$, then $S \cup \{A(t)\} \in \mathbf{C}$ for *some* ground term t.

Let Ψ be a set of sentences of the signature Σ. A consistency property $\mathbf{C}$ is called *Ψ-compatible* if

(Ψ) For every $S \in \mathbf{C}$ and $A \in \Psi$ we have $S \cup \{A\} \in \mathbf{C}$.

We will simply say *consistency property* instead of $\mathcal{L}$-consistency property, when it causes no ambiguity. Note that we only speak of Ψ-compatible consistency properties when formulas in Ψ use no parameters. Also note that the notion of consistency property depends on the signature and parameters used in formulas, because the ($\forall$)-condition requires a property to be satisfied for *every ground term*. So if a set of sequents is a consistency property in a language with parameters P, it may violate the ($\forall$)-condition considered in a language with more parameters.

The main theorem of this section is the following.

Theorem 21 (Model existence) Let $\mathbf{C}$ be a Ψ-compatible $\mathcal{L}$-consistency property and $S \in \mathbf{C}$ be a set of sentences in the signature Σ. Then there exists a Herbrand structure $\mathfrak{S}$ and a valuation V in $\mathfrak{S}$ under which S is locally and Ψ is globally satisfied.

The proof will be given after a series of lemmas.

The first three lemmas (22, 25 and 27) are applied to the consistency property to close it under subsets, add all parameter variants to it, and add all sets constructed from finite subsets already in it. The three lemmas are summarized as Proposition 28.

The first step in our attempt to build a model is to close the consistency property under subsets.

Lemma 22 (Subsets closure) Let $\mathbf{C}$ be a Ψ-compatible $\mathcal{L}$-consistency property and $\mathbf{C}'$ consist of the subsets of all $S \in \mathbf{C}$. Then $\mathbf{C}'$ is also a Ψ-compatible $\mathcal{L}$-consistency property and $\mathbf{C}'$ is closed under subsets.

PROOF. We consider only one case, the other cases are similar.

($\wedge$) Suppose $S' \in \mathbf{C}'$ and $A \wedge B \in S'$. We have to show $S' \cup \{A, B\} \in \mathbf{C}'$. Since $\mathbf{C}'$ consists of the subsets of sets in $\mathbf{C}$, for some $S \in \mathbf{C}$ we have $S' \subseteq S$. Then $A \wedge B \in S$, and by the ($\wedge$)-condition on consistency properties $S \cup \{A, B\} \in \mathbf{C}$. Evidently, $S' \cup \{A, B\} \subseteq S \cup \{A, B\}$, hence $S' \cup \{A, B\} \in \mathbf{C}'$.

□

Now, the conditions on consistency properties reflect both the definition of truth of formulas in structures and the rules of the sequent calculi, but with

one exception. The $(\exists)$-condition is in the spirit of the definition of truth (if $\exists x A(x)$ is true, then $A(p)$ is true for some p). However, the corresponding sequent calculus rule is

$$\frac{S, A(p)}{S, \exists x A(x)} \ (\exists),$$

where p is a new parameter. We want to make the notion of consistency property reflect this rule, so we will change the $(\exists)$-condition of consistency properties.

Definition 23 (Alternate $\mathcal{L}$-consistency property) Let $\mathbf{C}$ be a set of sequents. We say that $\mathbf{C}$ meets the *new parameter condition* if for each $S \in \mathbf{C}$, if $\exists x A(x) \in S$, then $S \cup \{A(p)\} \in \mathbf{C}$ for *every parameter* p that does not occur in S. If $\mathbf{C}$ satisfies all conditions for a (Ψ-compatible) consistency property except that the $(\exists)$-condition is replaced by the new parameter condition, then $\mathbf{C}$ is called an *alternate $\mathcal{L}$-consistency property*.

The condition that $A(p) \in S$ for *every* parameter p that does not occur in S is not restrictive. Since p does not occur in S (and, being a parameter, does not occur in Ψ either), there is from the viewpoint of S no difference between p and any other new parameter.

An alternate consistency property is not necessarily a consistency property. S may already contain all parameters. We will overcome this problem by an iterative construction of consistency properties, adding more parameters into the language at every iteration step.

Definition 24 (Parameter substitution, parameter variant) Any function $\sigma : P \to \mathcal{T}(\Sigma \cup P)$ is called a *parameter substitution*. For a sentence A, $\sigma(A)$ denotes the result of replacing every parameter in A by its image under σ. Similarly, σ is extended to sets of sentences. The formula $\sigma(A)$ and the set $\sigma(S)$ are called the *parameter variants* of A and S, respectively.

Lemma 25 (Parameter variants extension) Suppose that $\mathbf{C}'$ is a Ψ-compatible $\mathcal{L}$-consistency property closed under subsets. Define $\mathbf{C}''$ by: $S \in \mathbf{C}''$ if $\sigma(S) \in \mathbf{C}'$ for some parameter substitution σ. Then $\mathbf{C}''$ extends $\mathbf{C}'$ and is a Ψ-compatible *alternate* $\mathcal{L}$-consistency property closed under subsets.

PROOF. We will only verify that $\mathbf{C}''$ satisfies the new parameter condition, all other conditions are not difficult to prove.

Suppose $S \in \mathbf{C}''$, $\exists x A(x) \in S$ and p is a parameter that does not occur in S. We have to show that $S \cup \{A(p)\} \in \mathbf{C}''$.

Since $S \in \mathbf{C}''$, there is a parameter substitution σ such that $\sigma(S) \in \mathbf{C}'$. Note that $\sigma(\exists x A(x)) \in \sigma(S)$. Denote $\sigma(A(x))$ by $B(x)$. Since $\mathbf{C}'$ is a $\mathcal{L}$-consistency property and $\exists x B(x) \in \sigma(S)$, there exists a term t such that $\sigma(S) \cup \{B(t)\} \in \mathbf{C}'$. Define σ' to behave exactly as σ except that $\sigma'(p) = t$. Using the fact that p does not occur in S, it is not hard to argue that $\sigma'(S \cup \{A(p)\}) = \sigma(S) \cup \{B(t)\}$. Thus $S \cup \{A(p)\}$ is a parameter variant of a sequent in $\mathbf{C}'$, and hence it is a member of $\mathbf{C}''$. □

Next, we would like the consistency property to satisfy the finite character property defined below.

Definition 26 (Finite character) A collection $\mathbf{C}$ of sets is said to be of *finite character* if for every set S, S belongs to $\mathbf{C}$ if and only if each finite subset of S belongs to $\mathbf{C}$.

Lemma 27 (Finite character extension) Suppose $\mathbf{C}''$ is a Ψ-compatible alternate $\mathcal{L}$-consistency property closed under subsets. Let $\mathbf{C}'''$ consist of those sequents S all whose finite subsets are in $\mathbf{C}''$. Then $\mathbf{C}'''$ is again a Ψ-compatible alternate $\mathcal{L}$-consistency property, which extends $\mathbf{C}''$ and is of finite character.

PROOF. As usual, we will only check some conditions on alternate consistency properties.

($\vee$) Let $S \in \mathbf{C}'''$ and $A \vee B \in S$. We have to prove that either $S \cup \{A\} \in \mathbf{C}'''$ or $S \cup \{B\} \in \mathbf{C}'''$. Suppose, by contradiction, $S \cup \{A\} \notin \mathbf{C}'''$ and $S \cup \{B\} \notin \mathbf{C}'''$. By the definition of $\mathbf{C}'''$, there are finite sets $F_1 \subseteq S \cup \{A\}$ and $F_2 \subseteq S \cup \{B\}$ such that $F_1, F_2 \notin \mathbf{C}''$. Consider the finite set $F = (F_1 - \{A\}) \cup (F_2 - \{B\}) \cup \{A \vee B\}$. Then F is a finite subset of S, hence $F \in \mathbf{C}''$. By the condition ($\vee$) on $\mathbf{C}''$, either $F \cup \{A\} \in \mathbf{C}''$ or $F \cup \{B\} \in \mathbf{C}''$. We show that in either case we obtain a contradiction. Suppose $F \cup \{A\} \in \mathbf{C}''$ (the second case is similar). It is not hard to argue that $F_1 \subseteq F \cup \{A\}$. Since $\mathbf{C}''$ is closed under subsets, $F_1 \in \mathbf{C}''$. Contradiction.

($\exists$) Let $S \in \mathbf{C}'''$ and $\exists x A(x) \in S$. We have to prove that for each parameter p not occurring in S we have $S \cup \{A(p)\} \in \mathbf{C}'''$. Suppose, by contradiction $S \cup \{A(p)\} \notin \mathbf{C}'''$. By the definition of $\mathbf{C}'''$, there is a finite set $F \subseteq S \cup \{A(p)\}$ such that $F \notin \mathbf{C}''$. Consider the finite set $F' = (F - \{A(p)\}) \cup \{\exists x A(x)\}$. Then F' is a finite subset of S, so $F' \in \mathbf{C}''$. Note that p does not occur in F', then by the condition ($\exists$) on $\mathbf{C}''$ we have $F' \cup \{A(p)\} \in \mathbf{C}''$. Evidently, $F \subseteq F' \cup \{A(p)\}$. Since $\mathbf{C}''$ is closed under subsets, $F \in \mathbf{C}''$. Contradiction.

It is easy to see that $\mathbf{C}'''$ is of finite character. □

Let us summarize the results obtained so far:

Proposition 28 Let $\mathbf{C}$ be a Ψ-compatible $\mathcal{L}$-consistency property. Then $\mathbf{C}$ can be extended to a set $\mathbf{C}^*$ that is a Ψ-compatible alternate $\mathcal{L}$-consistency property of finite character.

Let us now state two lemmas about alternate consistency properties of finite character. Lemma 30 says that a restriction of an alternate consistency property of finite character to certain sublanguages gives us an alternate consistency property of finite character. It will be helpful when we need to add new parameters in order to satisfy the $(\exists)$-condition on consistency properties. Lemma 33 asserts the existence of maximal elements in sets of finite character, which will be used as possible worlds in the model construction.

Definition 29 (Section) Let P be a set of parameters and $\mathbf{C}$ be a set of sequents. By the *P-section* of $\mathbf{C}$, denoted $\mathbf{C}{\restriction_P}$, we mean

$$\{S \in \mathbf{C} \mid \text{each parameter occurring in } S \text{ is a member of } P\}.$$

The following lemma is straightforward.

Lemma 30 (Section restriction) Suppose $\mathbf{C}$ is a Ψ-compatible alternate $\mathcal{L}$-consistency property of finite character in the language with parameters P and $P_0 \subseteq P$. Then the P_0-section of $\mathbf{C}$ is a Ψ-compatible alternate $\mathcal{L}$-consistency property of finite character in the language with parameters P_0.

Note that this lemma does not hold when alternate consistency properties are replaced by consistency properties. Consider e.g. that while $\mathbf{C} = \{\{\exists x A(x)\}, \{\exists x A(x), A(p)\}\}$ is a consistency property, $\mathbf{C}{\restriction_\emptyset}$ is not.

The following lemma has a straightforward proof by transfinite induction on ordinals.

Lemma 31 Let $\mathbf{C}$ be a collection of sets of finite character. Then

1. Each member of $\mathbf{C}$ is contained in a maximal member;
2. The union of any chain of members is a member.

Now we give a definition closely related to witness formulas $A(p)$ for $\forall x A(x)$, and then proceed to the proof of Model Existence Theorem 21.

Definition 32 (Downward saturated set) Suppose $\mathbf{C}^*$ is a Ψ-compatible alternate $\mathcal{L}$-consistency property of finite character and P is a set of parameters. Let S be a set of sentences in $\mathcal{F}(\Sigma \cup P)$. We say S is *downward saturated in* $\mathbf{C}^*\restriction_P$ if

1. S is maximal in the alternate $\mathcal{L}$-consistency property $\mathbf{C}^*\restriction_P$.
2. If $\exists x A(x) \in S$, then $A(p) \in S$ for some $p \in P$.

Lemma 33 ($\exists$-completion) Suppose $\mathbf{C}^*$ is a Ψ-compatible alternate $\mathcal{L}$-consistency property of finite character in the language with parameters P_0, where P_0 has the same cardinality as $\mathcal{F}(\Sigma)$. Suppose also that $P, Q \subseteq P_0$ are disjoint sets of parameters of the same cardinality as $\mathcal{F}(\Sigma)$. If $S \in \mathbf{C}^*\restriction_P$, then S may be extended to a set that is downward saturated in $\mathbf{C}^*\restriction_{P\cup Q}$.

PROOF. Since Q is infinite, it can be partitioned into countably many pairwise disjoint sets $Q_1, Q_2, \ldots$, all of the same cardinality as Q itself. Note that the sets of $\exists$-sentences in the sets $\mathcal{F}(\Sigma)$, $\mathcal{F}(\Sigma \cup P)$, $\mathcal{F}(\Sigma \cup P \cup Q)$, and $\mathcal{F}(\Sigma \cup P \cup Q_1 \cup Q_2 \cup \cdots \cup Q_n)$ are all of the same cardinality as P.

Now, suppose $S \in \mathbf{C}^*\restriction_P$. Then $S \in \mathbf{C}^*\restriction_{P\cup Q_1}$. Well-order the members of Q_1 as $q_0, q_1, \ldots, q_\alpha, \ldots$ and the $\exists$-sentences of S: $\exists x A_0(x)$, $\exists x A_1(x)$, ..., $\exists x A_\alpha(x)$, ... in such a way that for every ordinal α, the set of parameters occurring in $A_\alpha(x)$ is a subset of $\{q_\beta \mid \beta < \alpha\}$.

Consider the set $S \cup \{A_\beta(q_\beta) \mid \beta = 0, 1, \ldots, \alpha, \ldots\}$. We claim it is a member of $\mathbf{C}^*\restriction_{P\cup Q_1}$. Suppose the contrary, then there is the smallest ordinal α such that $S \cup \{A_\beta(q_\beta) \mid \beta < \alpha\} \in \mathbf{C}^*\restriction_{P\cup Q_1}$ but $S \cup \{A_\beta(q_\beta) \mid \beta \leq \alpha\} \notin \mathbf{C}^*\restriction_{P\cup Q_1}$. Note that q_α does not occur in any member of $S \cup \{A_\beta(q_\beta) \mid \beta < \alpha\} \in \mathbf{C}^*\restriction_{P\cup Q_1}$. By Lemma 30, $\mathbf{C}^*\restriction_{P\cup Q_1}$ is an alternate $\mathcal{L}$-consistency property, so by the ($\exists$)-condition of alternate consistency properties

$$S \cup \{A_\beta(q_\beta) \mid \beta < \alpha\} \cup \{A_\alpha(q_\alpha)\} \in \mathbf{C}^*\restriction_{P\cup Q_1}.$$

But this set is exactly $S \cup \{A_\beta(q_\beta) \mid \beta \leq \alpha\}$, so we have a contradiction.

We have proved that $S \cup \{A_\beta(q_\beta) \mid \beta = 0, 1, \ldots, \alpha, \ldots\}$ is a member of $\mathbf{C}^*\restriction_{P\cup Q_1}$. By Lemma 31, it can be extended to a maximal member S_1. Now S_1 has a "witness parameter" q for every sentence $\exists x A(x)$ that uses only parameters in P, i.e. such that $A(q) \in S_1$. However, our construction does not imply that S_1 contains a witness for $\exists$-formulas containing parameters in Q_1.

To overcome this problem, we iterate the construction again, this time using S_1 instead of S and Q_2 instead of Q_1. Further iterations give us a sequence of sets

$$S \subseteq S_1 \subseteq S_2 \subseteq \cdots$$

such that

(8.1) each S_n is maximal in $\mathbf{C}^*\restriction_{P\cup Q_1\cup\cdots\cup Q_n}$;

(8.2) if $\exists x A(x) \in S_n$, then for some $q \in Q_{n+1}$ we have $A(q) \in S_{n+1}$.

Define $S^* = \bigcup_n S_n$. By Lemma 31, $\mathbf{C}^*$ is closed under chains, so $S^* \in \mathbf{C}^*$. Note that all parameters occurring in S^* are in Q, thus $S^* \in \mathbf{C}^*\restriction_{P\cup Q}$. We claim that S^* is a maximal member of $\mathbf{C}^*\restriction_{P\cup Q}$. This amounts to showing that for every sentence $B \in \mathcal{F}(\Sigma \cup P \cup Q)$, if $S^* \cup \{B\} \in \mathbf{C}^*\restriction_{P\cup Q}$, we have $B \in S^*$. Since B can contain only a finite number of parameters, then $B \in \mathcal{F}(\Sigma \cup P \cup Q_1 \cup \cdots \cup Q_n)$ for some n. Since $S \cup \{B\} \in \mathbf{C}^*\restriction_{P\cup Q}$, which is of finite character, hence closed under subsets, $S_n \cup \{B\} \in \mathbf{C}^*\restriction_{P\cup Q}$. It follows that $S_n \cup \{B\} \in \mathbf{C}^*\restriction_{P\cup Q_1\cup\cdots\cup Q_n}$, hence by maximality of S_n we have $B \in S_n$, and since $S_n \subseteq S^*$ also $B \in S^*$.

Using (8.2), one may show that if $\exists x A(x) \in S^*$, then $A(q) \in S^*$ for some $q \in Q$. Thus S^* is downward saturated in $\mathbf{C}^*\restriction_{P\cup Q}$. □

Now we have the background to prove the Model Existence Theorem.

PROOF (of Theorem 21). Recall that, given a Ψ-compatible $\mathcal{L}$-consistency property $\mathbf{C}$ and a set of sentences $S \in \mathbf{C}$ in the signature Σ, we are going to build a Herbrand structure $\mathfrak{S}$ and a valuation V in $\mathfrak{S}$ under which S is locally and Ψ is globally satisfied.

Using Proposition 28 extend $\mathbf{C}$ to an alternate $\mathcal{L}$-consistency property $\mathbf{C}^*$ that is also Ψ-compatible. We take a set P of parameters of the same cardinality as $\mathcal{F}(\Sigma)$ and split it in countably many mutually disjoint sets $P_1, P_2, \ldots$, each of them of the same cardinality as P itself. We will define our structure $\mathfrak{S} = \langle \mathcal{W}, \mathcal{D}, \longrightarrow, I, \Vdash \rangle$ over a domain $\mathbf{D}$ as follows.

The domain $\mathbf{D}$ is the set of all ground terms of Σ with parameters in P. We put a set S in $\mathcal{W}$ if S is downward saturated in $\mathbf{C}^*\restriction_{P_1\cup\cdots\cup P_n}$ for some n. The domain of any such world S is the subset of $\mathbf{D}$ consisting of terms with parameters in $P_1 \cup \cdots \cup P_n$. Since we want $\mathfrak{S}$ to be a Herbrand structure, we define the interpretation function I as in Definition 19. Now we define the valuation V as the identity mapping. By our remarks after the definition of Herbrand structures we have $V(t) = t$ for every ground term t of the signature $\Sigma \cup P$.

Define the reachability relation $\longrightarrow$ on $\mathcal{W}$ as follows: $S \xrightarrow{t} S'$ if $S^{[t]} \subseteq S'$ and $\mathcal{D}_S \subseteq \mathcal{D}_{S'}$.

Now we define the relation $\Vdash$ on $\mathfrak{S}$ as follows. For any atomic sentence A we let $S \Vdash A$ if $A \in S$. Thus, the structure $\mathfrak{S}$ and valuation V are defined. We prove the following:

(8.3) $\mathfrak{S}$ is a $\mathcal{L}$-structure.

First, we note the following useful facts:

(8.4) For any choice of logic $\mathcal{L}$, if $S_1 \subseteq S_2$, then ${S_1}^{[t]} \subseteq {S_2}^{[t]}$.

(8.5) For $\mathsf{S4}$ we have $S^{[t]} = {S^{[t]}}^{[t]}$ and $S^{[t]} \subseteq S$.

(8.6) For $\mathsf{K4}$ and $\mathsf{D4}$ we have $S^{[t]} \subseteq {S^{[t]}}^{[t]}$.

Now we verify (8.3) for each particular logic $\mathcal{L}$.

CASE : $\mathcal{L} = \mathsf{K}$. There is nothing to verify since every structure is a K-structure.

CASE : $\mathcal{L} = \mathsf{T}$. We have to prove $S \xrightarrow{t} S$, i.e. $S^{[t]} \subseteq S$. Take any $A \in S^{[t]}$, then $[t]A \in S$. Since $\mathbf{C}^*$ is an alternate T-consistency property, $S \cup \{A\} \in \mathbf{C}^*$. But S is maximal in a section of $\mathbf{C}^*$ containing all parameters in A, hence $A \in S$. Since A was arbitrary, $S^{[t]} \subseteq S$.

The condition on the domains $\mathcal{D}_S \subseteq \mathcal{D}_S$ is obvious. This condition will be obvious for all other cases, so we do not verify it henceforth.

CASE : $\mathcal{L} = \mathsf{D}$. Suppose $S \in \mathcal{W}$, we have to find $S' \in \mathcal{W}$ such that $S \xrightarrow{t} S'$, i.e. $S^{[t]} \subseteq S'$. Since $\mathbf{C}^*$ is an alternate D-consistency property, $S^{[t]} \in \mathbf{C}^*$. Take S' to be a maximal member extending $S^{[t]}$ in any language containing all parameters in S, then $S^{[t]} \subseteq S'$.

CASE : $\mathcal{L} = \mathsf{K4}$. Suppose ${S_1}^{[t]} \subseteq S_2$ and ${S_2}^{[t]} \subseteq S_3$. We have to prove ${S_1}^{[t]} \subseteq S_3$. By (8.4) above, ${S_1}^{[t]^{[t]}} \subseteq {S_2}^{[t]}$. Then by (8.6) above, ${S_1}^{[t]} \subseteq {S_2}^{[t]}$, this and ${S_2}^{[t]} \subseteq S_3$ implies ${S_1}^{[t]} \subseteq S_3$.

CASE : $\mathcal{L} = \mathsf{D4}$. Seriality is proved by the same argument as for D above and transitivity by the same argument as for $\mathsf{K4}$.

CASE : $\mathcal{L} = \mathsf{S4}$. Showing reflexivity reduces to $S^{[t]} \subseteq S$, which follows from (8.5) above. Transitivity is proved by the same argument as for $\mathsf{K4}$.

We proved (8.3), i.e. that $\mathfrak{S}$ is a $\mathcal{L}$-structure. Next, we prove:

(8.7) Let A be a sentence and $S \in \mathbf{C}^*$. If $A \in S$, then $S \Vdash A$.

The proof is by induction on the structure of A. We take any $S \in \mathbf{C}^*$. Suppose S is downward saturated in $\mathbf{C}^*\restriction_{P_1 \cup \ldots \cup P_n}$ and $A \in S$.

CASE : A is atomic. Then $S \Vdash A$ by the definition of $\Vdash$ in $\mathfrak{S}$.

CASE : A is a negative literal $\neg B$. By definition of consistency property we have $B \notin S$. Then by the definition of $\Vdash$ in $\mathfrak{S}$ we have $S \nVdash B$, and hence $S \Vdash A$.

CASE : A is $A_1 \wedge A_2$. By the definition of consistency property we have $S \cup \{A_1, A_2\} \in \mathbf{C}^*$. Since S is maximal, this implies $A_1, A_2 \in S$. By the induction hypothesis, $S \Vdash A_1$ and $S \Vdash A_2$, hence $S \Vdash A_1 \wedge A_2$.

CASE : A is $A_1 \vee A_2$. By the definition of consistency property, either $S \cup \{A_1\} \in \mathbf{C}^*$ or $S \cup \{A_2\} \in \mathbf{C}^*$. Since S is maximal, this implies that either $A_1 \in S$ or $A_2 \in S$. By the induction hypothesis, either $S \Vdash A_1$ or $S \Vdash A_2$, hence $S \Vdash A_1 \vee A_2$.

CASE : A is $\forall x B(x)$. By Lemma 30, $\mathbf{C}^*\restriction_{P_1 \cup \ldots \cup P_n}$ is an alternate consistency property in the signature $\Sigma \cup P_1 \cup \cdots \cup P_n$. Then for every term $t \in \mathcal{T}(\Sigma \cup P_1 \cup \cdots \cup P_n)$ we have $S \cup \{B(t)\} \in \mathbf{C}^*\restriction_{P_1 \cup \ldots \cup P_n}$. By maximality, S contains all formulas $B(t)$. By the induction hypothesis $S \Vdash B(t)$ for all such t. But $\mathcal{T}(\Sigma \cup P_1 \cup \cdots \cup P_n)$ is the domain of S, hence $S \Vdash \forall x B(x)$.

CASE : A is $\exists x B(x)$. Since S is downward saturated, S contains $B(t)$ for some $t \in \mathcal{T}(\Sigma \cup P_1 \cup \cdots \cup P_n)$, therefore $S \Vdash B(t)$. But t belongs to the domain of S, hence $S \Vdash \exists x B(x)$.

CASE : A is $\langle t \rangle B$. For every choice of logic $\mathcal{L}$ we have $S^{[t]} \cup \{B\} \in \mathbf{C}^*$. Take any downward saturated S' in a signature containing $\Sigma \cup P_1 \cup \cdots \cup P_n$ such that $S^{[t]} \cup \{B\}$. By our construction we have $S \stackrel{t}{\longrightarrow} S'$. By the induction hypothesis $S' \Vdash B$, hence $S \Vdash \langle t \rangle B$.

CASE : A is $[t]B$. We take any S' such that $S^{[t]} \subseteq S'$ and S' contains all parameters in S and claim $B \in S'$, then by the induction hypothesis $S' \Vdash B$. Consider two cases.

> SUBCASE : $\mathcal{L} = \mathsf{S4}$. By the definition of $S^{[t]}$ for $\mathsf{S4}$, if $[t]B \in S$, then $[t]B \in S^{[t]}$, hence $[t]B \in S'$. By the definition of consistency property for $\mathsf{S4}$, since $[t]B \in S'$, then $S' \cup \{B\} \in \mathbf{C}^*$. Since S' is maximal, $B \in S'$.
>
> SUBCASE : $\mathcal{L}$ is any other logic. By the definition of $S^{[t]}$ for $\mathcal{L}$, if $[t]B \in S$, then $B \in S^{[t]}$, hence $B \in S'$.

So (8.7) is proved. We return to the proof of the Model Existence Theorem. Recall that we intend to prove that under $\mathfrak{S}$ and V all formulas occurring

in any member of $\mathbf{C}$ are locally satisfied and all formulas in Ψ are globally satisfied.

1. Take any $A \in S \in \mathbf{C}$. Extend S to a downward saturated $S' \in \mathbf{C}^*$ (Proposition 28 and Lemma 33). Then S' is a world in $\mathfrak{S}$ and $A \in S'$. By (8.7), A is satisfied in this world.

2. Take any $A \in \Psi$ and $S \in \mathcal{W}$. Let S be downward saturated in $\mathbf{C}^*\restriction_{P_1 \cup \ldots \cup P_n}$. By Lemma 30, $\mathbf{C}^*\restriction_{P_1 \cup \ldots \cup P_n}$ is a Ψ-compatible alternate $\mathcal{L}$-consistency property, hence $S \cup \{A\} \in \mathbf{C}^*\restriction_{P_1 \cup \ldots \cup P_n}$. Since S is maximal, $A \in S$. By (8.7), A is satisfied in the world S. Since S was arbitrary, A is satisfied in every world of $\mathfrak{S}$.

The proof of the Model Existence Theorem is completed. □

8.2 THE COMPLETENESS THEOREM

The Completeness Theorem 18 can now be proved using the Model Existence Theorem 21 in a rather straightforward way.

PROOF (of Theorem 18). Take an infinite set of parameters P and consider the following set $\mathbf{C}$ of sequents of the signature Σ: we put S in $\mathbf{C}$ if S uses only a finite number of parameters and S has no refutation in $\mathcal{L}$ with global assumptions Ψ. We claim

(8.8) $\mathbf{C}$ is a Ψ-compatible $\mathcal{L}$-consistency property.

We consider only some conditions of $\mathcal{L}$-consistency property, others are rather straightforward. Take any $S \in \mathbf{C}$.

(A) *We prove: S contains no atomic formula A and its negation $\neg A$.* Indeed, if S contains A and $\neg A$, it is an axiom of $\mathcal{L}$, hence has a refutation.

($\wedge$) *We prove: if $A \wedge B \in S$, then $S \cup \{A, B\} \in \mathbf{C}$.* Suppose $A \wedge B \in S$. Consider the inference

$$\frac{S, A, B}{S, A \wedge B}\ (\wedge).$$

If $S \cup \{A, B\}$ had a refutation, so would $S \cup \{A \wedge B\} = S$, hence $S \cup \{A, B\}$ has no refutation. By the definition of $\mathbf{C}$, we have $S \cup \{A, B\} \in \mathbf{C}$.

($\langle t \rangle$) *We prove: if* $\langle t \rangle A \in S$, *then* $S^{[t]} \cup \{A\} \in \mathbf{C}$. Suppose $\langle t \rangle A \in S$. Consider the inference

$$\frac{S^{[t]}, A}{S, \langle t \rangle A} \ (\langle t \rangle).$$

If $S^{[t]} \cup \{A\}$ had a refutation, so would $S \cup \{\langle t \rangle A\} = S$, hence $S^{[t]} \cup \{A\}$ has no refutation. By the definition of $\mathbf{C}$, we have $S^{[t]} \cup \{A\} \in \mathbf{C}$.

($\exists$) *We prove: if* $\exists x A(x) \in S$, *then* $S \cup \{A(t)\} \in \mathbf{C}$ *for* some *ground term* t. Suppose $\exists x A(x) \in S$. Consider the inference

$$\frac{S, A(p)}{S, \exists x A(x)} \ (\exists).$$

If $S \cup \{A(p)\}$ had a refutation, so would $S \cup \{\exists x A(x)\} = S$, hence $S \cup \{A(p)\}$ has no refutation. By the definition of $\mathbf{C}$, since we have an infinite number of parameters and S uses only a finite number of them, we can always choose a new parameter p so that $S \cup \{A(p)\} \in \mathbf{C}$.

(Ψ) *We prove: if* $S \in \mathbf{C}$ *and* $A \in \Psi$, *then* $S \cup \{A\} \in \mathbf{C}$. Suppose $S \in \mathbf{C}$ and $A \in \Psi$. Consider the inference

$$\frac{S, A}{S} \ (\Psi).$$

If $S \cup \{A\}$ had a refutation, so would S, hence $S \cup \{A\}$ has no refutation. By the definition of $\mathbf{C}$, we have $S \cup \{A\} \in \mathbf{C}$.

Now take S that has no refutation. By our construction, $S \in \mathbf{C}$. By Model Existence Theorem 21 there exists a $\mathcal{L}$-structure and valuation V in it under which S is locally satisfied and every formula $A \in \Psi$ globally satisfied. $\square$

Free-variable Tableaux

In this chapter we change the tableau systems introduced in Section 6.2 into free-variable tableau systems. We will use the definitions introduced so far, except that we now allow free variables to occur in sequents, and hence in tableaux as well.

To avoid problems with the parameter condition in the ($\forall$)-rules, we introduce so-called *occurrence constraints* similar to those used in (Voronkov 1996) and in (Voronkov 2001). Note that we could use the "dynamic skolemization" technique introduced in (Fitting 1988) as well.

In this section we assume knowledge of the standard notions of substitutions and (idempotent, most general) unifiers (see, e.g., Eder 1985). The application of a substitution σ to a term or formula E is denoted $E\sigma$. As usual, we may need to rename bound variables in a formula before we apply a substitution to it. Any idempotent most general unifier of n expressions $E_1, \ldots, E_n$ is denoted by $mgu(E_1, \ldots, E_n)$. The set of free variables of any expression E (e.g. formula or set of formulas) is denoted by $vars(E)$.

Definition 34 (Occurrence constraint) A *simple occurrence constraint* is either $\bot$ or an expression $p \notin X$, where p is a parameter and X is a finite set of variables. An *occurrence constraint* is a conjunction of zero or more simple occurrence constraints. A conjunction of zero simple occurrence constraints is denoted by $\top$.

For any substitution σ and simple occurrence constraint $\mathcal{C} = (p \notin X)$, we denote by $\mathcal{C}\sigma$ the following simple occurrence constraint:

$$\mathcal{C}\sigma = \begin{cases} \bot, & \text{if } p \text{ occurs in } X\sigma; \\ p \notin vars(X\sigma), & \text{otherwise.} \end{cases}$$

When $\mathcal{C}$ is a conjunction $\mathcal{C}_1 \wedge \cdots \wedge \mathcal{C}_n$ of simple occurrence constraints, we denote by $\mathcal{C}\sigma$ the following occurrence constraint:

$$\mathcal{C}\sigma = \begin{cases} \bot, & \text{if } \mathcal{C}_i = \bot \text{ for some } i; \\ \mathcal{C}_1\sigma \wedge \cdots \wedge \mathcal{C}_n\sigma, & \text{otherwise.} \end{cases}$$

An occurrence constraint $\mathcal{C}$ is called *satisfiable* if $\mathcal{C}$ is not $\bot$. A *solution* to an occurrence constraint $\mathcal{C}$ is any substitution σ such that $\mathcal{C}\sigma \neq \bot$ and $x\sigma$ is ground for every variable x occurring in $\mathcal{C}$. Evidently, an occurrence constraint $\mathcal{C}$ is satisfiable if and only if it has a solution: indeed, one can take as a solution any substitution mapping all variables of $\mathcal{C}$ into any ground term not containing parameters in $\mathcal{C}$.

We call a *constrained tableau* any pair consisting of a tableau $\mathcal{T}$ and constraint $\mathcal{C}$, denoted $\mathcal{T} \cdot \mathcal{C}$. Let $\mathcal{L}$ be one of the logics $\mathsf{K}, \mathsf{D}, \mathsf{T}, \mathsf{K4}, \mathsf{D4}$ and $\mathsf{S4}$. The *free-variable tableau calculi* for $\mathcal{L}$ are shown in Figure 9.1.

We claim

Theorem 35 (Equivalence of free-variable and sequent calculi) Let S be a set of sentences of the signature Σ. Then S has a refutation in the sequent calculus for $\mathcal{L}$ (with global assumptions Ψ) if and only if there exists a derivation of $\# \cdot \mathcal{C}$ from $S \cdot \top$ in the tableau calculus for $\mathcal{L}$ (with the global assumptions Ψ) such that $\mathcal{C}$ is satisfiable.

In order to prove this theorem, we will prove two results showing bisimulation between tableau derivations and free-variable tableau derivations.

Let $\mathcal{T} \cdot \mathcal{C}$ be a constrained tableau and σ be a substitution. We call the tableau $\mathcal{T}\sigma$ the *σ-instance* of $\mathcal{T} \cdot \mathcal{C}$ if $\mathcal{C}\sigma$ is satisfiable. A tableau $\mathcal{T}'$ is called an *instance* of $\mathcal{T} \cdot \mathcal{C}$ if it is a σ-instance of $\mathcal{T} \cdot \mathcal{C}$ for some σ.

The following lemma establishes a simulation of free-variable tableau derivations by tableau derivations.

Lemma 36 Suppose there exists a derivation of $\mathcal{T}_2 \cdot \mathcal{C}$ from $\mathcal{T}_1 \cdot \top$ in the free-variable tableau calculus for $\mathcal{L}$ with global assumptions Ψ. Then any instance of $\mathcal{T}_2 \cdot \mathcal{C}$ has a derivation from $\mathcal{T}_1$ in the tableau calculus for $\mathcal{L}$ with the global assumptions Ψ.

PROOF. The proof is by induction on the length of derivations in the free-variable tableau calculus. When the derivation is of length 0, the claim is obvious, since $\mathcal{T}_1$ is the only instance of $\mathcal{T}_1 \cdot \top$, when $\mathcal{T}_1$ has no free variables. For derivations with at least one inference, consider the last inference of the derivation. We will consider only two cases, other cases are similar.

CASE : the last inference is |ax|.

For all logics (K, D, T, K4, D4, S4):

$$\frac{S, A(\bar{s}), \neg A(\bar{t}) \mid \mathcal{T} \cdot \mathcal{C}}{\mathcal{T} mgu(\bar{s}, \bar{t}) \cdot \mathcal{C} mgu(\bar{s}, \bar{t})} \; |\mathrm{ax}|$$

$$\frac{S, A \vee B \mid \mathcal{T} \cdot \mathcal{C}}{S, A \mid S, B \mid \mathcal{T} \cdot \mathcal{C}} \; |\vee| \qquad \frac{S, A \wedge B \mid \mathcal{T} \cdot \mathcal{C}}{S, A, B \mid \mathcal{T} \cdot \mathcal{C}} \; |\wedge|$$

$$\frac{S, \exists x A(x) \mid \mathcal{T} \cdot \mathcal{C}}{S, A(p) \mid \mathcal{T} \cdot \mathcal{C} \wedge p \notin vars(S, \exists x A(x))} \; |\exists|^* \qquad \frac{S, \forall x A(x) \mid \mathcal{T} \cdot \mathcal{C}}{S, \forall x A(x), A(y) \mid \mathcal{T} \cdot \mathcal{C}} \; |\forall|^*$$

$$\frac{S, \langle t \rangle A \mid \mathcal{T} \cdot \mathcal{C}}{S^{[t_1]}, \ldots, S^{[t_n]}, A \mid \mathcal{T}\sigma \cdot \mathcal{C}\sigma} \; |\langle t, t_1, \ldots, t_n \rangle|^*$$

For serial logics (D, D4):

$$\frac{S \mid \mathcal{T} \cdot \mathcal{C}}{S^{[t_1]}, \ldots, S^{[t_n]} \mid \mathcal{T}\sigma \cdot \mathcal{C}\sigma} \; |[t_1, \ldots, t_n]|^*$$

For reflexive logics (T, S4):

$$\frac{S, [t] A \mid \mathcal{T} \cdot \mathcal{C}}{S, A \mid \mathcal{T} \cdot \mathcal{C}} \; |[t]|$$

For logics with global assumptions Ψ:

$$\frac{S \mid \mathcal{T} \cdot \mathcal{C}}{S, A \mid \mathcal{T} \cdot \mathcal{C}} \; |\Psi|^*$$

* In the rule $|\langle t, t_1, \ldots, t_n \rangle|$, $\sigma = mgu(t, t_1, \ldots, t_n)$. In the rule $|[t_1, \ldots, t_n]|$, $\sigma = mgu(t_1, \ldots, t_n)$. In the rule $|\exists|$, p is new parameter, not occurring in the premise. In the rule $|\forall|$, y is a new variable, not occurring in the premise. In the rule $|\Psi|$, $A \in \Psi$.

Figure 9.1: Free-variable tableau with constraints calculi

$$\frac{S, A(\bar{s}), \neg A(\bar{t}) \mid \mathcal{T} \cdot \mathcal{C}}{\mathcal{T}\sigma \cdot \mathcal{C}\sigma} \; |\mathrm{ax}|,$$

where $\sigma = mgu(\bar{s}, \bar{t})$.

Take any instance of $\mathcal{T}\sigma \cdot \mathcal{C}\sigma$, then this instance has the form $\mathcal{T}\sigma\tau$ for some substitution τ such that $\mathcal{C}\sigma\tau$ is satisfiable. We have to prove that $\mathcal{T}\sigma\tau$ is derivable from $\mathcal{T}_1$.

We claim that the following is a valid inference in the tableau calculus:

$$\frac{(S, A(\bar{s}), \neg A(\bar{t}) \mid \mathcal{T})\sigma\tau}{\mathcal{T}\sigma\tau} \; |\mathrm{ax}|. \tag{9.1}$$

Indeed, since σ is a unifier of $\bar{s}$ and $\bar{t}$, then $A(\bar{s})\sigma = A(\bar{t})\sigma$, hence $A(\bar{s})\sigma\tau = A(\bar{t})\sigma\tau$.

Since $\mathcal{C}\sigma\tau$ is satisfiable, we get that $(S, A(\bar{s}), \neg A(\bar{t}) \mid \mathcal{T})\sigma\tau$ is a $\sigma\tau$-instance of $S, A(\bar{s}), \neg A(\bar{t}) \mid \mathcal{T} \cdot \mathcal{C}$. By the induction hypothesis, this instance has a derivation from $\mathcal{T}_1$. Add to this derivation the inference (9.1), then we obtain a required derivation of $\mathcal{T}\sigma\tau$.

CASE : the last inference is $|\exists|$.

$$\frac{S, \exists x A(x) \mid \mathcal{T} \cdot \mathcal{C}}{S, A(p) \mid \mathcal{T} \cdot \mathcal{C} \wedge p \notin vars(S, \exists x A(x))} \; |\exists|.$$

Take any instance of $S, A(p) \mid \mathcal{T} \cdot \mathcal{C} \wedge p \notin vars(S, \exists x A(x))$, then this instance has the form $S\tau, A(p)\tau \mid \mathcal{T}\tau$ for some substitution τ such that $(\mathcal{C} \wedge p \notin vars(S, \exists x A(x)))\tau$ is satisfiable. We have to prove that $S\tau, A(p)\tau \mid \mathcal{T}\tau$ is derivable from $\mathcal{T}_1$.

Since $(\mathcal{C} \wedge p \notin vars(S, \exists x A(x)))\tau$ is satisfiable, by definition of constraint satisfiability, $\mathcal{C}\tau$ is satisfiable and p does not occur in $S\tau, \exists x A(x)\tau$. Since p does not occur in $S\tau, \exists x A(x)\tau$, the following is a valid inference in the tableau calculus:

$$\frac{S\tau, \exists x A(x)\tau \mid \mathcal{T}\tau}{S\tau, A(p)\tau \mid \mathcal{T}\tau} \; |\exists|. \tag{9.2}$$

By the induction hypothesis, every instance of $S, \exists x A(x) \mid \mathcal{T} \cdot \mathcal{C}$ has a derivation from $\mathcal{T}_1$. Since $\mathcal{C}\tau$ is satisfiable, we can take its τ-instance $S\tau, \exists x A(x)\tau \mid \mathcal{T}\tau$, this instance has a derivation from $\mathcal{T}_1$. Add to this derivation inference (9.2) and we obtain a required derivation of $S\tau, A(p)\tau \mid \mathcal{T}\tau$ from $\mathcal{T}_1$.

□

Now we want to prove a simulation result in the inverse direction. If we defined a sequent as a multiset of formulas, we could use an argument similar to the previous lemma. The use of sets instead of multisets causes some technical problems because the notion of instance does not work properly any more. To avoid these technical problems we give a definition of generalization that is nearly inverse to the notion of instance but takes into account some specific problems in the inverse simulation proof.

Let $\mathcal{T} = S_1 \mid \cdots \mid S_n$ and $\mathcal{T}' = S'_1 \mid \cdots \mid S'_n$ be two tableaux. We write $\mathcal{T} \sqsubseteq \mathcal{T}'$ if (i) for every $i = 1 \ldots n$ we have $S_i \subseteq S'_i$ and (ii) each parameter occurring in some $\mathcal{T}'$ also occurs in $\mathcal{T}$. Let $\mathcal{T} \cdot \mathcal{C}$ be a constrained tableau and $\mathcal{T}'$ a tableau. We call $\mathcal{T} \cdot \mathcal{C}$ a *σ-generalization* of $\mathcal{T}'$ if $\mathcal{T}' \sqsubseteq \mathcal{T}\sigma$ and $\mathcal{C}\sigma$ is satisfiable. We call $\mathcal{T} \cdot \mathcal{C}$ a *generalization* of $\mathcal{T}'$ if $\mathcal{T} \cdot \mathcal{C}$ is a σ-generalization of $\mathcal{T}'$ for some σ.

Lemma 37 Suppose there exists a derivation of $\mathcal{T}_2$ from $\mathcal{T}_1$ in the tableau calculus for $\mathcal{L}$ with global assumptions Ψ. Then some generalization of $\mathcal{T}_2$ has a derivation from $\mathcal{T}_1 \cdot \top$ in the free-variable tableau calculus for $\mathcal{L}$ with the global assumptions Ψ.

PROOF. The proof is by induction on the length of derivations in the tableau calculus. When the derivation is of length 0, the claim is obvious, since $\mathcal{T}_1 \cdot \top$ is a generalization of $\mathcal{T}_1$. For derivations with at least one inference, consider the last inference of the derivation. We will consider only two cases, other cases are similar.

CASE : the last inference is |ax|.

$$\frac{S, A, \neg A \mid \mathcal{T}}{\mathcal{T}} \ |\mathrm{ax}|.$$

By the induction hypothesis, some σ-generalization of $S, A, \neg A \mid \mathcal{T}$ is derivable from $\mathcal{T}_1 \cdot \top$. Then this generalization has a form $S', A', \neg B' \mid \mathcal{T}' \cdot \mathcal{C}$ such that $A'\sigma = A$, $B'\sigma = A$, $\mathcal{T} \sqsubseteq \mathcal{T}'\sigma$ and $\mathcal{C}\sigma$ is satisfiable. Then σ is a unifier of A' and B', therefore, there exists a most general unifier τ of A' and B' and a substitution δ such that $\tau\delta = \sigma$. Consider the following inference in the free-variable tableau calculus.

$$\frac{S', A', \neg B' \mid \mathcal{T}' \cdot \mathcal{C}}{\mathcal{T}'\tau \cdot \mathcal{C}\tau} \ |\mathrm{ax}|.$$

We claim that the conclusion of this inference is a generalization of $\mathcal{T}$, which will complete the proof of this case. To prove the claim, we have to find

a substitution δ' such that (i) $\mathcal{T} \sqsubseteq \mathcal{T}'\tau\delta'$ and (ii) $\mathcal{C}\tau\delta'$ is satisfiable. Well, take δ' to be δ, then both (i) and (ii) follow from $\tau\delta = \sigma$.

CASE : the last inference is $|\exists|$.

$$\frac{S, \exists x A(x) \mid \mathcal{T}}{S, A(p) \mid \mathcal{T}} \; |\exists| \tag{9.3}$$

By the induction hypothesis, some σ-generalization of $S, \exists x A(x) \mid \mathcal{T}$ is derivable from $\mathcal{T}_1 \cdot \top$. Then this generalization has form $S', \exists x A'(x) \mid \mathcal{T}' \cdot \mathcal{C}$ such that (i) $S \subseteq S'\sigma$, (ii) every parameter occurring in $(S', \exists x A'(x))\sigma$ also occurs in $S, \exists x A(x) \mid \mathcal{T}$, (iii) $\exists x A'(x)\sigma = \exists x A(x)$, (iv) $\mathcal{T} \sqsubseteq \mathcal{T}'\sigma$, and (v) $\mathcal{C}\sigma$ is satisfiable.

Consider the following inference in the free-variable tableau calculus.

$$\frac{S', \exists x A'(x) \mid \mathcal{T}' \cdot \mathcal{C}}{S', A'(p) \mid \mathcal{T}' \cdot \mathcal{C} \wedge p \notin \mathit{vars}(S', \exists x A'(x))} \; |\exists|.$$

Let us check that the parameter condition is satisfied. Suppose, by contradiction, that p occurs in $S', \exists x A'(x) \mid \mathcal{T}'$, then it also occurs in $(S', \exists x A'(x) \mid \mathcal{T}')\sigma$, hence also in $S, \exists x A(x) \mid \mathcal{T}$. This violates the parameter condition of (9.3).

We claim that the conclusion of this inference is a generalization of $S, A(p) \mid \mathcal{T}$, which will complete the proof of this case. We actually claim that the conclusion is the σ-generalization of $S, A(p) \mid \mathcal{T}$. All conditions on σ-generalization except for constraint satisfaction immediately follow from (i)–(iv) above. It remains to verify that $(\mathcal{C} \wedge p \notin \mathit{vars}(S', \exists x A'(x)))\sigma$ is satisfiable. $\mathcal{C}\sigma$ is satisfiable by (v) above, so it remains to check that p does not occur in $(S', \exists x A'(x))\sigma$. If p occurred in $(S', \exists x A'(x))\sigma$, then by (ii) above p would also occur in $S, \exists x A(x) \mid \mathcal{T}$, but this is impossible because of the parameter condition in (9.3).

□

Now we can prove soundness and completeness of the free-variable calculi.

PROOF (of Theorem 35).

1. Suppose S has a refutation in the sequent calculus for $\mathcal{L}$ with global assumptions Ψ. Then by Theorem 16 there exists a derivation of $\#$ from S in the tableau calculus for $\mathcal{L}$ with the global assumptions Ψ. Hence, by Lemma 37 there exists a derivation of some generalization of $\#$ from $S \cdot \top$ in the free-variable tableau calculus for $\mathcal{L}$. But any generalization of $\#$ has the form $\# \cdot \mathcal{C}$ for a satisfiable $\mathcal{C}$.

2. Suppose there exists a derivation of $\# \cdot \mathcal{C}$ from $S \cdot \top$ in the tableau calculus for $\mathcal{L}$ with the global assumptions Ψ such that $\mathcal{C}$ is satisfiable. By Lemma 36 any instance of $\# \cdot \mathcal{C}$ has a derivation from S in the tableau calculus for $\mathcal{L}$ with the global assumptions Ψ. Obviously, $\#$ is such an instance, so it is derivable from S as well. By Theorem 16 S has a refutation in the sequent calculus for $\mathcal{L}$ with the global assumptions Ψ.

□

9.1 EXAMPLE REFUTATION

Consider the following formula valid in term-modal K. (For better readability, we will omit parenthesis in terms like $f(x)$ and write fx instead.)

$$\forall x \exists y([y]R(y,y) \wedge [fy](R(fy,fy) \supset R(y,fy)) \supset [fx]R(x,fx)).$$

We will establish the validity of this formula, i.e. unsatisfiability of its negation using the free-variable tableau calculus for K. First, we negate the formula and transform it into its negation normal form:

$$\exists x \forall y([y]R(y,y) \wedge [fy](\neg R(fy,fy) \vee R(y,fy)) \wedge \langle fx \rangle \neg R(x,fx)),$$

and then show its refutation. The refutation is given in Figure 9.2. In the refutation we do not show the constraint, since it always has the form $p \notin \emptyset$ and is satisfiable. For better readability, we denote the inference steps by $\rightarrow$ followed by the name of the inference rule. We also group similar inferences into one. For example, by $|\forall|^*$ we denote a sequence of $|\forall|$ inferences, and by $|\wedge|^*$ a sequence of $|\wedge|$ inferences.

$$\exists x \forall y([y]R(y,y) \wedge [fy](\neg R(fy,fy) \vee R(y,fy)) \wedge \langle fx\rangle \neg R(x,fx)) \rightarrow |\exists|$$

$$\forall y([y]R(y,y) \wedge [fy](\neg R(fy,fy) \vee R(y,fy)) \wedge \langle fp\rangle \neg R(p,fp)) \rightarrow \quad |\forall|^*$$

$$\begin{array}{l}\forall y([y]R(y,y) \wedge [fy](\neg R(fy,fy) \vee R(y,fy)) \wedge \langle fp\rangle \neg R(p,fp)),\\ [z]R(z,z) \wedge [fz](\neg R(fz,fz) \vee R(z,fz)) \wedge \langle fp\rangle \neg R(p,fp),\\ [u]R(u,u) \wedge [fu](\neg R(fu,fu) \vee R(u,fu)) \wedge \langle fp\rangle \neg R(p,fp) \rightarrow \quad |\wedge|^*\end{array}$$

$$\begin{array}{l}\forall y([y]R(y,y) \wedge [fy](\neg R(fy,fy) \vee R(y,fy)) \wedge \langle fp\rangle \neg R(p,fp)),\\ [z]R(z,z),\\ [fz](\neg R(fz,fz) \vee R(z,fz)),\\ \langle fp\rangle \neg R(p,fp),\\ [u]R(u,u),\\ [fu](\neg R(fu,fu) \vee R(u,fu)) \rightarrow \quad |\langle fp,z,fu\rangle|\end{array}$$

$$\begin{array}{l}R(fp,fp),\\ \neg R(p,fp),\\ \neg R(fp,fp) \vee R(p,fp) \rightarrow \quad |\vee|\end{array}$$

$$\begin{array}{l}R(fp,fp), \neg R(p,fp), \neg R(fp,fp) \ \ |\\ R(fp,fp), \neg R(p,fp), R(p,fp) \rightarrow \quad |\mathrm{ax}|^*\end{array}$$

$$\#$$

Figure 9.2: Example refutation in the free-variable calculus

Conclusion and Future Work

A complete sequent calculus was presented for a logic in which it is possible to quantify over modalities. We note that even though we have restricted ourselves to the logics $\mathsf{K}, \mathsf{D}, \mathsf{T}, \mathsf{K4}, \mathsf{D4}$, and $\mathsf{S4}$, other logics can be included among the term-modal logics.

Term-modal logic can be used to reason about epistemic multi-agent systems or to develop action logics. Interesting future work includes joining epistemic term-modal logic with dynamic logic for actions, e.g. knowledge updates.

There are several interesting fragments of the term-modal logic, which might be interesting to study. We suggest some of the possibilities.

1. Restricting all predicate symbols to arity zero.
2. Restricting modal indexes to variables, instead of arbitrary terms.
3. Various fragments of the logic without function symbols.

Part II

Quantifier-free Dynamic Assignment Logic

SYNTAX

We define the syntax of quantifier-free dynamic assignment logic. The Basic syntax (Section 11.1) is the language used to express various properties about programs. The Extended syntax (Section 11.3) is the language used to prove the properties.

We will, for Part II, denote equality with $=$, syntactic identity with $\equiv$ and definitional equality with $\stackrel{def}{=}$.

11.1 BASIC SYNTAX

Assume pairwise disjoint sets for *relation symbols* $\mathcal{R} \stackrel{def}{=} \{P, Q, \ldots\}$, *function symbols* $\mathcal{F} \stackrel{def}{=} \{f, g, \ldots\}$, *program variables* $\mathcal{P} = \{a, b, \ldots\}$, and *logic variables* $\mathcal{V} \stackrel{def}{=} \{x, y, z, \ldots\}$. The triple $\Sigma = \langle \mathcal{R}, \mathcal{F}, \mathcal{P} \rangle$ is called the *signature* of the logic. Each relation and function symbol has an associated arity n. The subset of the function symbols with arity 0 are called *constant symbols*, and will be denoted $c, d, \ldots$. Program variables have arity 0.[1]

The syntactic components of the logic are defined by[2]

$$
\begin{array}{llll}
\textit{Formulas} & \varphi, \psi & ::= & P(t_1, \ldots, t_n) \mid t_1 = t_2 \mid \\
 & & & (\varphi \wedge \psi) \mid \neg\varphi \mid \langle p \rangle \varphi \mid \{\lambda x \leftarrow t.\ \varphi(x)\} \\
\textit{Programs} & p, q & ::= & a := t \mid (p; q) \mid (p \cup q) \mid p* \mid \varphi? \\
\textit{Terms} & t & ::= & a \mid c \mid x \mid f(t_1, \ldots, t_n)
\end{array}
$$

where a is a program variable, c is a constant symbol, x is a logic variable, f is a function symbol, P is a relation symbol and t is a term. Programs of the form $a := t$ are called *atomic programs*, and formulas of the form $P(t_1, \ldots, t_n)$ or $t_1 = t_2$ are called *atomic formulas*. The term t in the

[1] We could also introduce program variables of higher arity, thereby introducing arrays into the logic. But, in order to keep the presentation simple, we have chosen not to.

[2] Note that Fitting and Mendelsohn (1998) only allow logic variables in atomic formulas $P(x_1, \ldots, x_n)$, while we allow arbitrary terms in atomic formulas $P(t_1, \ldots, t_n)$.

program $a := t$ has to be *ground*, i.e. it has to be a term which do not contain any logic variables. Note that the logic is quantifier-free.

The formula $\langle p \rangle \varphi$ intuitively says that φ is a possible outcome when running program p. The program $p; q$ is the *composition* of program p and program q. Intuitively this means that p is executed first, and q second. The program $p \cup q$ is the *choice* of programs p and q, i.e, non-deterministically one of them is executed. The *star-program* $p*$ means that p is executed any non-deterministic number of times. The program $\varphi?$ is a *test* checking that the formula φ holds.

When verifying programs, it is important to distinguish between the value of a program variable *before* program execution and the value of the program variable *after* program execution. Consider, for instance, the formula $\langle a := t \rangle P(a)$. In our semantics, the a in $P(a)$ refers to the value of a *after* execution. If we instead want to talk about the value of a *before* execution of $a := t$, we need to use the *scoping operator* λ and write the *abstraction formula*

$$\{\lambda x \leftarrow a.\ \langle a := t \rangle P(x)\}.$$

The position of λx in the formula determines which value of a we are referring to. In the formula above, λx stands before the modal operator $\langle a := t \rangle$, so by x in the subformula $P(x)$, we mean the value of a before program $a := t$ has been executed.

A formula is called *closed* if every logic variable occurrence x is bound by a scoping operator λx. A logic variable x is *free* in a formula, if x occurs non-bound in the formula. We write $\{\lambda x, y \leftarrow t_1, t_2.\ \varphi\}$ as an abbreviation of $\{\lambda x \leftarrow t_1.\ \{\lambda y \leftarrow t_2.\ \varphi\}\}$. Other connectives are defined in the usual way, i.e. $[p]\varphi \stackrel{def}{=} \neg\langle p \rangle \neg\varphi$, $(\varphi \vee \psi) \stackrel{def}{=} \neg(\neg\varphi \wedge \neg\psi)$ and $(\varphi \supset \psi) \stackrel{def}{=} \neg(\varphi \wedge \neg\psi)$. Any formula $(\varphi \wedge \neg\varphi)$ or $\neg t = t$ can be abbreviated by $\bot$ and $\neg\bot$ is abbreviated $\top$.

Also, $(p_1; (p_2; (\cdots (p_{n-1}; p_n) \cdots)))$ is abbreviated by $(p_1; p_2; \cdots p_{n-1}; p_n)$ and similarly for $\cup$.

11.2 EXAMPLES

More complex programming constructs can be defined using the above primitive dynamic constructs. We can, for instance, express the following traditional programs as dynamic programs.

SKIP	$\stackrel{def}{=}$	$\top?$
IF (φ) THEN (p_1) ELSE (p_2)	$\stackrel{def}{=}$	$((\varphi?; p_1) \cup (\neg\varphi?; p_2))$
WHILE (φ) DO (p)	$\stackrel{def}{=}$	$((\varphi?; p)*; \neg\varphi?)$

Swapping two program variables a and b, $\mathrm{SWAP}(a, b)$, can be expressed by the program $(a' := a; a := b; b := a')$.

It is well-known that everything expressible in Hoare logic (Hoare 1969) can be expressed in dynamic logic, see e.g. (Kozen and Tiuryn 1989). The *total correctness formula* of p with respect to *precondition* φ and *postcondition* ψ, sometimes written $\{\varphi\}p\{\psi\}$, is written in dynamic logic as

$$\varphi \supset [p]\psi.$$

We can express that there exists an execution of $(a := f(a))*$, such that the final value of a equal the value of $f(f(x))$, where x denotes the initial value of a:

$$\{\lambda x \leftarrow a.\ \langle (a := f(a))*\rangle a = f(f(x))\}.$$

If a term t is locally rigid with respect to a program p (i.e. the value of t is the same before and after p has been executed), then the abstraction operator and the box operator commute, in the following sense (a special case of this formula is proved in Example 70, Page 81)

$$\{\lambda x \leftarrow t.\ [p]x = t\} \supset (\{\lambda x \leftarrow t.\ [p]\varphi(x)\} \supset [p]\{\lambda x \leftarrow t.\ \varphi(x)\}).$$

Since there is no need to use the scoping operator in modal-free subformulas, if $\varphi(x)$ is an atomic formula $P(x)$, then we can write

$$\{\lambda x \leftarrow t.\ [p]x = t\} \supset (\{\lambda x \leftarrow t.\ [p]P(x)\} \supset [p]P(t)).$$

We can express "if a and b are different, then no number of applications of f on both a and b can make them denote the same value", by the formula

$$\neg(a = b) \supset \neg\langle (a := f(a); b := f(b))*\rangle a = b.$$

This formula is not generally true in our logic, since function symbols have no fixed meaning in the logic. The function f could e.g. be interpreted as a constant function, which would make the formula false.

11.3 EXTENDED SYNTAX

We introduce prefixes to name states.

Definition 38 (Prefix) A *prefix* is a sequence $[a_1 := t_1]\cdots[a_n := t_n]$ where $n \geq 0$, each a_i is a program variable, and each t_i is a ground term. The empty sequence is denoted ϵ and is called the *empty prefix*.

The intuition is that ϵ names the initial state, i.e. the state before any program has been executed. The state named $[a := t]$ intuitively denotes the state after the initial state, when a has been assigned t.

Definition 39 (Subscripted, fixed, subscript-free) A *subscripted term* is a term in which program variables can be subscripted with a prefix. A *fixed term* is a subscripted term in which every program variable has a prefix as subscript.[3] A formula, program, or subscripted term is *subscript-free* if it contains no fixed program variables.

The set of subscripted terms includes all terms introduced in Section 11.1.

From this point on, we will allow *formulas* and *programs* to contain subscripted terms, instead of only the terms defined in Section 11.1.

The fixed program variable a_τ intuitively stands for the value of a in the state named τ. Note that a ground term can be either fixed (e.g. a_τ) or non-fixed (e.g. a), and a fixed term can be either ground (e.g. $f(a_\tau)$) or non-ground (e.g. $f(x)$). As we will later see, there is a difference between the program variable a fixed by the empty prefix ϵ (that is a_ϵ), and the subscript-free program variable a. The first means that a should be interpreted in the state named by the empty prefix, while the latter provides no means to determine what state it should be interpreted in.

Definition 40 ($t@\tau$) If τ is a prefix and t is a subscripted term, we denote by $t@\tau$ the fixed term resulting from subscripting every non-fixed program variable in t with τ. We say that the term t is *fixed* by prefix τ in the term $t@\tau$.

For any function symbol f with arity $n \geq 1$, we have

$$f(t_1, \ldots, t_n)@\tau = f(t_1@\tau, \ldots, t_n@\tau).$$

Definition 41 (Prefixed formula) A *prefixed formula* is an expression of the form $\tau : \varphi$, where τ is a prefix and φ is a formula.

The prefixed formula $\tau : \varphi$ intuitively means that φ is true in the state named by prefix τ. The prefixed formula $\tau : \langle p \rangle \varphi$ intuitively means that φ is true in a state resulting from executing the program in the state named τ. The prefixed formula $\tau : [p]\varphi$ similarly means that φ is true in every state resulting from executing p in the state named τ.

[3] Fitting and Mendelsohn (1998) calls fixed terms *grounded*.

11.4 SUBSTITUTION

Substitution is defined in the usual way, see e.g. (Eder 1985). The concept of free substitutions is used much in the same way as in (Kleene 1952) and (Fitting 1996a, Fitting 1999).

Definition 42 (Substitution) A *substitution* is a mapping σ from the set of logic variables $\mathcal{V}$ to the set of subscripted terms, such that there is only a finite number of variables x such that $\sigma(x) \neq x$.

We will write $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ for the substitution mapping each x_i to t_i, and all other variables to themselves.

Definition 43 (Free substitution) A substitution $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ is *free for* φ (or p) if for every $x_i \in \{x_1, \ldots, x_n\}$, there is no free occurrence of x_i in φ (or p) within the scope of a λy, such that y occurs in t_i. Let $\sigma = [x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ be a substitution free for φ. Then by $\varphi\sigma$ we denote the formula in which every free occurrence of variable x_i has been replaced by the term t_i. Similarly for $t\sigma$ and $p\sigma$.

We will write $\varphi(x_1, \ldots, x_n)$ for a formula with zero or more occurrences of variables $x_1, \ldots, x_n$. When later writing $\varphi(t_1, \ldots, t_n)$, we mean the formula resulting from applying the substitution $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ to $\varphi(x_1, \ldots, x_n)$. We require this substitution to be free for the formula. If it is not, then we rename bound variables in the formula before applying the substitution. This can always be done so that the substitution becomes free for the formula. We use the same convention for programs, writing $p(t_1, \ldots, t_n)$ for the program resulting from applying the free substitution $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ to $p(x_1, \ldots, x_n)$.

In computer science, program variables are considered to have two values, the l-value and the r-value, see e.g. (Aho, Sethi and Ullman 1985). The r stands for 'right', indicating that this is the value we are interested in when the program variable occurs on the right hand side of an assignment, and similarly l stands for 'left'. The r-value is the actual value of the program variable, and the l-value is the "storage location". When a program variable occurs on the left hand side of an assignment, we are not interested in the actual value of it, but rather its storage location, or name. This can be compared with the ideas of Frege (1892), where a distinction is made between *sense* (Sinn) and *reference* (Bedeutung) of names. The sense corresponds to the r-value, and the reference to the l-value.

In our logic, program variables have both r-values and l-values. Two distinct program variables a and b might have the same r-value (i.e. $a = b$), but different l-values ($a := c$ and $b := c$ have different meaning). Consider, as

an example of this, the following formula, which is not generally true (a countermodel of the formula is presented in Example 53, p. 70.)

$$a = b \supset (\langle a := c\rangle a = c \supset \langle b := c\rangle a = c).^4$$

In classical logic, there is a principle of substitution. If two constants have the same value, it is possible to substitute one for the other in a formula without changing the meaning of the formula. For program variables in our logic, this is not the case. A program variable is not generally substitutable for another, even if they have the same value.[5] Consider e.g. the formula $a = b \supset [p]a = b$. Even though a and b might have the same value in the current state, they might have different values in a later state (i.e. after p has been executed).

Consider the formula $\langle a := t\rangle\varphi$. Fixing a by τ in the assignment does not make sense. Fixing t by τ in the assignment is intuitively more reasonable, but introduces complications. As we will later see, the proof procedure will fix program variables in formulas, but never when they occur in an assignment program. This is a consequence of the requirement that every t in an assignment program $a := t$ has to be ground.

[4] If we want to be explicit about program variable interpretation (more about this in Section 12.1), we can write the formula in the following way:

$$\{\lambda x, y \leftarrow a, b.\ x = y\} \supset (\langle a := c\rangle\{\lambda x, y \leftarrow a, c.\ x = y\} \supset \langle b := c\rangle\{\lambda x, y \leftarrow a, c.\ x = y\}).$$

Use of the scoping operator in modal-free subformulas is redundant though.

[5] What is allowed, though, is to replace *every* occurrence of a program variable by a program variable *not* occurring in the formula. This corresponds to the principle that names of program variables are not important and can be changed without affecting the meaning of the program.

Semantics

The logic presented is a *dynamic logic*. Traditional dynamic logic use modal connectives $*$, $;$, $\cup$, and $?$. The program $p*$ has the same meaning as nondeterministically repeating p, $\langle (p;q)\rangle\varphi$ means the same as $\langle p\rangle\langle q\rangle\varphi$, and similarly for choice, test and assignment.

In our framework we use only one type of atomic programs, the assignment. Furthermore, a scoping operator λ is used instead of quantifiers.

In Section 12.1, we introduce semantics of subscript-free formulas and programs, and in Section 12.2 we introduce semantics for formulas and programs containing subscripted terms.

12.1 BASIC SEMANTICS

We start with two introductory concepts. Firstly, we define the meaning of logic variables, program variables, relation and function symbols. Secondly, we present how this meaning can be *updated*.

Assume a set of logic variables $\mathcal{V}$, a signature $\Sigma = \langle \mathcal{R}, \mathcal{F}, \mathcal{P}\rangle$, and two nonempty sets, $\mathcal{D}$ and $\mathcal{W}$, called the *domain* and the set of *worlds* or *states*, respectively. We now define the semantics.

Definition 44 (Valuation) A *valuation (of logic variables)* is a function ν from the set of logic variables to the domain, $\nu : \mathcal{V} \to \mathcal{D}$.

Definition 45 (Interpretation) An *interpretation $\mathcal{I}$ of relation symbols, function symbols and program variables* over $\langle \mathcal{W}, \mathcal{D}\rangle$ is a function that assigns

1. to every relation symbol of arity $n \geq 0$, a subset of $\mathcal{D}^n$,
2. to every function symbol of arity $n \geq 0$, a function $\mathcal{D}^n \to \mathcal{D}$, and

3. to every world $w \in \mathcal{W}$, a function $\mathcal{P} \to \mathcal{D}$, from program variables to elements of the domain.

We will write $\mathcal{I}(w, a)$ for $\mathcal{I}(w)(a)$. Note that when $\mathcal{I}$ is applied to only one argument, $\mathcal{I}(w)$ is a function taking each program variable in $\mathcal{P}$ to an element of $\mathcal{D}$.

Definition 46 (Interpretation of terms) Given a valuation ν and an interpretation $\mathcal{I}$, an *interpretation of terms* $(\nu \star \mathcal{I})$ is defined by

$$(\nu \star \mathcal{I})(w,t) \stackrel{def}{=} \begin{cases} \nu(t), & \text{if } t \text{ is a logic variable} \\ \mathcal{I}(w,t), & \text{if } t \text{ is a program variable} \\ \mathcal{I}(f)((\nu \star \mathcal{I})(w,t_1), & \text{if } t \equiv f(t_1,\ldots,t_n) \\ \qquad \ldots, & \\ \qquad (\nu \star \mathcal{I})(w,t_n)), & \end{cases}$$

We define update operations for valuations and interpretations. The update $\nu[x \mapsto t^w]$ is like ν except that x is mapped to the value of t in world w, and the update $\mathcal{I}(w)[a \mapsto t]$ is like $\mathcal{I}(w)$ except that a is mapped to the value of t in world w.[1]

Definition 47 (Update operations) Let ν be a valuation and $\mathcal{I}$ an interpretation. Then, for any logic variable x, program variables a, b and term t, let $\nu[x \mapsto t^w]$ be a valuation $\mathcal{V} \to \mathcal{D}$ defined by:

$$\nu[x \mapsto t^w](y) \stackrel{def}{=} \begin{cases} (\nu \star \mathcal{I})(w,t), & \text{if } y \equiv x \\ \nu(y), & \text{if } y \not\equiv x \end{cases}$$

And let $\mathcal{I}(w)[a \mapsto t]$ be a function $\mathcal{P} \to \mathcal{D}$, defined by:

$$\mathcal{I}(w)[a \mapsto t](b) \stackrel{def}{=} \begin{cases} (\nu \star \mathcal{I})(w,t), & \text{if } b \equiv a \\ \mathcal{I}(w,b), & \text{if } b \not\equiv a \end{cases}$$

Note that $\nu[x \mapsto t^w]$ and $\mathcal{I}(w)[a \mapsto t]$ are both dependent on $(\nu \star \mathcal{I})$, since t may contain both logic and program variables. The notation $\nu[x \mapsto t^w]$ will be used unambiguously in any case, since we will only use it in the context of some structure in which $\mathcal{I}$ is fixed. Similarly, $\mathcal{I}(w)[a \mapsto t]$ will only be used in the context of some ν, or when t does not contain any logic variables.

Definition 48 (Structure) A *(first-order dynamic) structure* is a tuple $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I} \rangle$, where

[1] In (Fitting and Mendelsohn 1998) the update $\nu[x \mapsto t^w]$ is called "the x-variant of ν, such that $\nu(x) = (\nu \star \mathcal{I})(w,t)$".

1. $\mathcal{W}$ and $\mathcal{D}$ are arbitrary (non-empty and disjoint) sets, and
2. $\mathcal{I}$ is an interpretation over $\langle \mathcal{W}, \mathcal{D} \rangle$, such that for every world $w_1 \in \mathcal{W}$, program variable $a \in \mathcal{P}$, and domain element $d \in \mathcal{D}$, there exists a world $w_2 \in \mathcal{W}$, such that $\mathcal{I}(w_2, a) = d$ and for each $b \not\equiv a$, we have $\mathcal{I}(w_2, b) = \mathcal{I}(w_1, b)$.

The set $\mathcal{D}$ is called the *domain* of the structure, and the set $\mathcal{W}$ is called the set of *worlds* or *states*. The requirement on the interpretation is called the *seriality condition*, and enforces that all assignment programs are executable in every state.

Definition 49 (Interpretation of programs, truth relation) Given a structure $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I} \rangle$, we define the *interpretation of subscript-free programs* $(\cdot)^{\mathcal{M},\nu}$ and the *truth relation* $\Vdash$ *of subscript-free formulas* simultaneously. For any valuation ν and $w \in \mathcal{W}$:

1. $\mathcal{M}, w, \nu \Vdash P(t_1, \ldots, t_n)$ iff $\langle (\nu \star \mathcal{I})(w, t_1), \ldots, (\nu \star \mathcal{I})(w, t_n) \rangle \in \mathcal{I}(P)$
2. $\mathcal{M}, w, \nu \Vdash (\varphi \wedge \psi)$ iff $\mathcal{M}, w, \nu \Vdash \varphi$ and $\mathcal{M}, w, \nu \Vdash \psi$
3. $\mathcal{M}, w, \nu \Vdash \neg\varphi$ iff it is not the case that $\mathcal{M}, w, \nu \Vdash \varphi$
4. $\mathcal{M}, w, \nu \Vdash \langle p \rangle \varphi$ iff there exists $w' \in \mathcal{W}$ such that $\langle w, w' \rangle \in (p)^{\mathcal{M},\nu}$ and $\mathcal{M}, w', \nu \Vdash \varphi$
5. $\mathcal{M}, w, \nu \Vdash \{\lambda x \leftarrow t.\, \varphi(x)\}$ iff $\mathcal{M}, w, \nu[x \mapsto t^w] \Vdash \varphi(x)$
6. $\mathcal{M}, w, \nu \Vdash t_1 = t_2$ iff $(\nu \star \mathcal{I})(w, t_1) = (\nu \star \mathcal{I})(w, t_2)$
7. $(a := t)^{\mathcal{M},\nu} \stackrel{def}{=} \{\langle w_1, w_2 \rangle \mid \mathcal{I}(w_2) = \mathcal{I}(w_1)[a \mapsto t]\}$
8. $(p \cup q)^{\mathcal{M},\nu} \stackrel{def}{=} (p)^{\mathcal{M},\nu} \cup (q)^{\mathcal{M},\nu}$
9. $(p; q)^{\mathcal{M},\nu} \stackrel{def}{=} (p)^{\mathcal{M},\nu} \circ (q)^{\mathcal{M},\nu} = \{\langle w, w'' \rangle \mid \exists w'. \langle w, w' \rangle \in (p)^{\mathcal{M},\nu} \wedge \langle w', w'' \rangle \in (q)^{\mathcal{M},\nu}\}$
10. $(p*)^{\mathcal{M},\nu} \stackrel{def}{=}$ the reflexive and transitive closure of $(p)^{\mathcal{M},\nu}$
11. $(\varphi?)^{\mathcal{M},\nu} \stackrel{def}{=} \{\langle w, w \rangle \mid \mathcal{M}, w, \nu \Vdash \varphi\}$

The interpretation of programs $(\cdot)^{\mathcal{M},\nu}$ and the truth relation $\Vdash$ are both completely determined by the structure $\mathcal{M}$ and valuation ν.

In modal logic, an accessibility relation between worlds is often explicitly defined for every modal operator. Here, we can instead extract an accessibility relation from Definition 49 by stating that a world w_2 is *p-accessible* from w_1 if $\langle w_1, w_2 \rangle \in (p)^{\mathcal{M},\nu}$. It easy to see that the $(a := t)$-accessibility relation is serial because of the seriality condition.

We state some simple properties of the truth relation.

Proposition 50

1. $\mathcal{M}, w, \nu \Vdash (\varphi \vee \psi)$ iff $\mathcal{M}, w, \nu \Vdash \varphi$ or $\mathcal{M}, w, \nu \Vdash \psi$.
2. $\mathcal{M}, w, \nu \Vdash [p]\varphi$ iff for all w' such that $\langle w, w' \rangle \in (p)^{\mathcal{M},\nu}$, $\mathcal{M}, w', \nu \Vdash \varphi$.
3. $\mathcal{M}, w, \nu \Vdash \langle p; q \rangle \varphi$ iff $\mathcal{M}, w, \nu \Vdash \langle p \rangle \langle q \rangle \varphi$.
4. $\mathcal{M}, w, \nu \Vdash \langle p \cup q \rangle \varphi$ iff $\mathcal{M}, w, \nu \Vdash (\langle p \rangle \varphi \vee \langle q \rangle \varphi)$.
5. $\mathcal{M}, w, \nu \Vdash \langle \psi? \rangle \varphi$ iff $\mathcal{M}, w, \nu \Vdash (\psi \wedge \varphi)$.

PROOF. Most properties are trivial to prove. We prove 2 and 3.

2. $\mathcal{M}, w, \nu \Vdash [p]\varphi$ iff $\mathcal{M}, w, \nu \Vdash \neg\langle p \rangle \neg\varphi$ iff we do not have that $\mathcal{M}, w, \nu \Vdash \langle p \rangle \neg\varphi$ iff there exists no w' such that $\langle w, w' \rangle \in (p)^{\mathcal{M},\nu}$ and $\mathcal{M}, w', \nu \Vdash \neg\varphi$ iff for all w' such that $\langle w, w' \rangle \in (p)^{\mathcal{M},\nu}$ we have $\mathcal{M}, w', \nu \Vdash \varphi$.
3. $\mathcal{M}, w, \nu \Vdash \langle p; q \rangle \varphi$ holds iff there exists a w'', such that $\langle w, w'' \rangle \in (p; q)^{\mathcal{M},\nu}$ and $\mathcal{M}, w'', \nu \Vdash \varphi$. This is true iff there exists w' and w'' such that $\langle w, w' \rangle \in (p)^{\mathcal{M},\nu}$ and $\langle w', w'' \rangle \in (q)^{\mathcal{M},\nu}$ and $\mathcal{M}, w'', \nu \Vdash \varphi$. Which is the same as that there exists w' such that $\langle w, w' \rangle \in (p)^{\mathcal{M},\nu}$ and $\mathcal{M}, w', \nu \Vdash \langle q \rangle \varphi$, which holds iff $\mathcal{M}, w, \nu \Vdash \langle p \rangle \langle q \rangle \varphi$.

□

Generalizing Propositions 50:3 and 50:4, we get the following equivalences.

$$\begin{array}{lll} \mathcal{M}, w, \nu \Vdash \langle p_1; \cdots; p_n \rangle \varphi & \text{iff} & \mathcal{M}, w, \nu \Vdash \langle p_1 \rangle \cdots \langle p_n \rangle \varphi. \\ \mathcal{M}, w, \nu \Vdash \langle p_1 \cup \cdots \cup p_n \rangle \varphi & \text{iff} & \mathcal{M}, w, \nu \Vdash (\langle p_1 \rangle \varphi \vee \cdots \vee \langle p_n \rangle \varphi). \end{array}$$

Note that there is still a need for complex modal connectives to retain the expressibility of the language, since formulas such as $\langle p?* \rangle \varphi$ and $\langle (p; q)* \rangle \varphi$ can not be expressed without using operators for test, choice etc, see (Berman and Paterson 1981).

Definition 51 (Truth, satisfiability, validity, model, countermodel) Let $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I} \rangle$ be a structure and let φ be a subscript-free formula.

1. φ is *true*, *holds*, or is *locally satisfied in world* $w \in \mathcal{W}$ *under valuation* ν if $\mathcal{M}, w, \nu \Vdash \varphi$.
2. φ is *locally satisfiable in* $\mathcal{M}$ if it is locally satisfied in some world of $\mathcal{M}$ under some valuation. Is it *locally satisfiable* if there exists a structure in which it is locally satisfiable.

3. $\mathcal{M}$ is a *model* of φ if φ is locally satisfiable in $\mathcal{M}$.

4. $\mathcal{M}$ is a *countermodel* of φ if there exists a world w in $\mathcal{M}$ and a valuation ν, such that $\mathcal{M}, w, \nu \Vdash \varphi$ does *not* hold.

5. φ is *valid* if it is true in every world in every structure under every valuation.

It is easy to see that a subscript-free formula φ is valid iff no countermodel exists for φ, and it can be shown, that φ is locally satisfiable iff $\neg\varphi$ is non-valid.

Example 52 (Model) Consider the valid formula[2]

$$\{\lambda x \leftarrow a.\ \langle a' := a\rangle\langle a := b\rangle\langle b := a'\rangle x = b\},$$

in signature $\Sigma = \langle\emptyset, \emptyset, \{a, b, a'\}\rangle$. We consider a particular model $\mathcal{M}$ of the formula, and show that the formula holds in a world w_1. Let

$$\mathcal{M} = \langle\{w_1, \ldots, w_9\}, \{1, 2\}, \mathcal{I}\rangle^3$$

where

$$\begin{array}{ll} \mathcal{I}(w_1) = \{(a,1),(b,2),\ldots\} & \mathcal{I}(w_2) = \{(a,1),(b,2),(a',1)\} \\ \mathcal{I}(w_3) = \{(a,2),(b,2),(a',1)\} & \mathcal{I}(w_4) = \{(a,2),(b,1),(a',1)\} \\ \mathcal{I}(w_5) = \ldots \text{(not interesting for the example)} & \end{array}$$

Let ν be an arbitrary valuation, and let $\nu' \stackrel{def}{=} \nu[x \mapsto a^{w_1}]$. We show that $\mathcal{M}$ is a model of the formula.

$$\begin{array}{lcll} \mathcal{M}, w_1, \nu & \Vdash & \{\lambda x \leftarrow a.\ \langle a' := a\rangle\langle a := b\rangle\langle b := a'\rangle x = b\} & \text{if} \\ \mathcal{M}, w_1, \nu' & \Vdash & \langle a' := a\rangle\langle a := b\rangle\langle b := a'\rangle x = b & \text{if} \\ \mathcal{M}, w_2, \nu' & \Vdash & \langle a := b\rangle\langle b := a'\rangle x = b & \text{if} \\ \mathcal{M}, w_3, \nu' & \Vdash & \langle b := a'\rangle x = b & \text{if} \\ \mathcal{M}, w_4, \nu' & \Vdash & x = b & \end{array}$$

Which is true, since we have $(\nu' \star \mathcal{I})(w_4, x) = \nu'(x) = \nu[x \mapsto a^{w_1}](x) = (\nu \star \mathcal{I})(w_1, a) = \mathcal{I}(w_1, a) = 1 = \mathcal{I}(w_4, b) = (\nu' \star \mathcal{I})(w_4, b)$.

[2] The formula intuitively says: "After the program SWAP(a, b) has been executed, b has the same value as a had before execution of the program."

[3] The seriality condition implies that $\mathcal{M}$ has at least 9 worlds, since we have 2 domain elements and 3 program variables. In general, the number of states of a structure is the number of domain elements raised to the power of the number of program variables. For our example, only 4 states of the structure are interesting though.

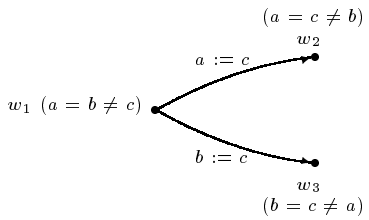


Figure 12.1: Countermodel of $a = b \supset (\langle a := c\rangle a = c \supset \langle b := c\rangle a = c)$

Example 53 (Countermodel) Consider the formula

$$a = b \supset (\langle a := c\rangle a = c \supset \langle b := c\rangle a = c)$$

in signature $\Sigma = \langle \emptyset, \{c\}, \{a, b\}\rangle$. The following is a countermodel of the formula.

$$\mathcal{M} = \langle \{w_1, \ldots, w_4\}, \{1, 2\}, \mathcal{I}\rangle, \text{ where } \mathcal{I}(c) = 2 \text{ and}$$

$$\mathcal{I}(w_1) = \{(a, 1), (b, 1)\} \qquad \mathcal{I}(w_2) = \{(a, 2), (b, 1)\}$$
$$\mathcal{I}(w_3) = \{(a, 1), (b, 2)\} \qquad \mathcal{I}(w_4) = \{(a, 2), (b, 2)\}$$

A picture of the countermodel is given in Figure 12.1 (not all transitions and worlds are shown). It is easy to see for any valuation ν, that $\mathcal{M}, w_1, \nu \Vdash a = b$ and $\mathcal{M}, w_1, \nu \Vdash \langle a := c\rangle a = c$, but $\mathcal{M}, w_1, \nu \Vdash \langle b := c\rangle a = c$ does not hold.

12.2 EXTENDED SEMANTICS

The interpretations of relation symbols, function symbols, program variables and terms are extended to subscripted terms, and we define satisfiability for sets of prefixed formulas.

Definition 54 (Prefix mapping) Let $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I}\rangle$ be a structure. A *prefix mapping for* $\mathcal{M}$ is any total mapping θ from prefixes to $\mathcal{W}$ such that for any two prefixes τ and $\tau[a := t]$,

$$\mathcal{I}(\theta(\tau[a := t])) = \mathcal{I}(\theta(\tau))[a \mapsto t].$$

It is easy to see that for any prefix mapping θ for $\mathcal{M}$, prefixes τ, $\tau[a := t]$ and valuation ν, we have $\langle \theta(\tau), \theta(\tau[a := t])\rangle \in (a := t)^{\mathcal{M},\nu}$.

Definition 55 (Extended interpretation) Let $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I}\rangle$ be a structure. Then the *extended interpretation* with respect to a prefix mapping θ for $\mathcal{M}$ and interpretation $\mathcal{I}$, is the function $\mathcal{I}^\theta$ from relation symbols,

function symbols and $\langle w, a\rangle$-pairs (where w is a world and a is a subscripted program variable), to the domain $\mathcal{D}$, such that

1. $\mathcal{I}^\theta(P) = \mathcal{I}(P)$, for each relation symbol $P \in \mathcal{R}$,
2. $\mathcal{I}^\theta(f) = \mathcal{I}(f)$, for each function symbol $f \in \mathcal{F}$,
3. $\mathcal{I}^\theta(w, a) = \mathcal{I}(w, a)$, for each non-fixed program variable $a \in \mathcal{P}$, and
4. $\mathcal{I}^\theta(w, a_\tau) = \mathcal{I}(\theta(\tau), a)$, for each fixed program variable a_τ, where $a \in \mathcal{P}$.

The *interpretation of subscripted terms* $(\nu \star \mathcal{I}^\theta)$ is defined by

$$(\nu \star \mathcal{I}^\theta)(w,t) \stackrel{def}{=} \begin{cases} \nu(t), & \text{if } t \text{ is a logic variable} \\ \mathcal{I}^\theta(w,t), & \text{if } t \text{ is a (fixed or non-fixed) program variable} \\ \mathcal{I}^\theta(f)((\nu \star \mathcal{I}^\theta)(w,t_1), \ldots, (\nu \star \mathcal{I}^\theta)(w,t_n)), & \text{if } t \equiv f(t_1,\ldots,t_n) \text{ and } n \geq 0 \end{cases}$$

Updates of extended interpretations are defined similarly to updates of interpretations. For any non-fixed program variable a and subscript-free term t, let $\mathcal{I}^\theta(w)[a \mapsto t]$ be the function from the set of subscripted program variables to the domain $\mathcal{D}$, defined by:

$$\mathcal{I}^\theta(w)[a \mapsto t](b) \stackrel{def}{=} \begin{cases} (\nu \star \mathcal{I}^\theta)(w,t), & \text{if } b \equiv a \\ \mathcal{I}^\theta(w,b), & \text{if } b \not\equiv a \\ \mathcal{I}^\theta(w,b), & \text{if } b \text{ is a fixed program variable} \end{cases}$$

For any fixed program variable b_τ (including a_τ), we have $\mathcal{I}^\theta(w)[a \mapsto t](b_\tau) = \mathcal{I}(\theta(\tau), b)$.

If we have an extended interpretation which in two different worlds agree on all program variables, then so does the interpretation of subscripted terms on all terms.

Proposition 56 Suppose that $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I}\rangle$ is a structure, θ is a prefix mapping for $\mathcal{M}$, and $w_1, w_2 \in \mathcal{W}$. If $\mathcal{I}^\theta(w_1) = \mathcal{I}^\theta(w_2)$, then for any valuation ν and subscripted term t, $(\nu \star \mathcal{I}^\theta)(w_1, t) = (\nu \star \mathcal{I}^\theta)(w_2, t)$.

PROOF. By induction on the structure of t.

a. If t is a non-fixed program variable a, then $(\nu \star \mathcal{I}^\theta)(w_1, a) = \mathcal{I}^\theta(w_1, a) = \mathcal{I}^\theta(w_2, a) = (\nu \star \mathcal{I}^\theta)(w_2, a)$.

a_τ. If t is a fixed program variable a_τ, then $(\nu \star \mathcal{I}^\theta)(w_1, a_\tau) = \mathcal{I}^\theta(w_1, a_\tau) = \mathcal{I}^\theta(\theta(\tau), a) = \mathcal{I}^\theta(w_2, a_\tau) = (\nu \star \mathcal{I}^\theta)(w_2, a_\tau)$.

x. If t is a logic variable x, then $(\nu \star \mathcal{I}^\theta)(w_1, x) = \nu(x) = (\nu \star \mathcal{I}^\theta)(w_2, x)$.

f. If t is a subscripted term $f(t_1, \ldots, t_n)$, with $n \geq 0$, then

$$(\nu \star \mathcal{I}^\theta)(w_1, f(t_1, \ldots, t_n)) = \mathcal{I}^\theta(f)((\nu \star \mathcal{I}^\theta)(w_1, t_1), \ldots, (\nu \star \mathcal{I}^\theta)(w_1, t_n)),$$

which by the induction hypothesis is equal to

$$\mathcal{I}^\theta(f)((\nu \star \mathcal{I}^\theta)(w_2, t_1), \ldots, (\nu \star \mathcal{I}^\theta)(w_2, t_n)) = (\nu \star \mathcal{I}^\theta)(w_2, f(t_1, \ldots, t_n)).$$

□

The interpretation of a subscripted term t in world $\theta(\tau)$, is the same as the interpretation of $t@\tau$ in any world, which we state as the following proposition.

Proposition 57 Suppose that $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I} \rangle$ is a structure and that θ is a prefix mapping for $\mathcal{M}$. Then, for any subscripted term t, prefix τ, and world $w \in \mathcal{W}$, we have $(\nu \star \mathcal{I}^\theta)(w, t@\tau) = (\nu \star \mathcal{I}^\theta)(\theta(\tau), t)$.

PROOF. By induction on the structure of t.

a. If t is a non-fixed program variable a, then $(\nu \star \mathcal{I}^\theta)(w, a@\tau) = (\nu \star \mathcal{I}^\theta)(w, a_\tau) = \mathcal{I}^\theta(w, a_\tau) = \mathcal{I}^\theta(\theta(\tau), a) = (\nu \star \mathcal{I}^\theta)(\theta(\tau), a)$.

$a_{\tau'}$. If t is a fixed program variable $a_{\tau'}$, then $(\nu \star \mathcal{I}^\theta)(w, a_{\tau'}@\tau) = (\nu \star \mathcal{I}^\theta)(w, a_{\tau'}) = \mathcal{I}^\theta(w, a_{\tau'}) = \mathcal{I}^\theta(\theta(\tau'), a) = \mathcal{I}^\theta(\theta(\tau), a_{\tau'})$, which, finally, is equal to $(\nu \star \mathcal{I}^\theta)(\theta(\tau), a_{\tau'})$.

x. If t is a logic variable x, then $(\nu \star \mathcal{I}^\theta)(w, x@\tau) = (\nu \star \mathcal{I}^\theta)(w, x) = \nu(x) = (\nu \star \mathcal{I}^\theta)(\theta(\tau), x)$.

f. If t is a subscripted term $f(t_1, \ldots, t_n)$, where $n \geq 0$, then

$$\begin{aligned}&(\nu \star \mathcal{I}^\theta)(w, f(t_1, \ldots, t_n)@\tau) \quad = \\ &\mathcal{I}^\theta(f)((\nu \star \mathcal{I}^\theta)(w, t_1@\tau), \ldots, (\nu \star \mathcal{I}^\theta)(w, t_n@\tau))\end{aligned}$$

which by the induction hypothesis is equal to

$$\begin{aligned}&\mathcal{I}^\theta(f)((\nu \star \mathcal{I}^\theta)(\theta(\tau), t_1), \ldots, (\nu \star \mathcal{I}^\theta)(\theta(\tau), t_n)) \quad = \\ &(\nu \star \mathcal{I}^\theta)(\theta(\tau), f(t_1, \ldots, t_n)).\end{aligned}$$

□

Definition 58 (Interpretation of programs, truth relation) Given a structure $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I} \rangle$ and a prefix mapping θ for $\mathcal{M}$, we define a new *interpretation of programs* $(\cdot)^{\mathcal{M},\nu,\theta}$ and a new *truth relation* $\Vdash_\theta$ for formulas and programs in the same way as $\Vdash$ and $(\cdot)^{\mathcal{M},\nu}$ are defined for subscript-free formulas and programs in Definition 49, except that every occurrence of $(\nu \star \mathcal{I})$ is replaced by $(\nu \star \mathcal{I}^\theta)$, every $\Vdash$ by $\Vdash_\theta$, and every $(\cdot)^{\mathcal{M},\nu}$ by $(\cdot)^{\mathcal{M},\nu,\theta}$.

For any subscript-free formula φ, it is easy to see that $\mathcal{M}, w, \nu \Vdash \varphi$ iff $\mathcal{M}, w, \nu \Vdash_\theta \varphi$ (by induction on φ). But if φ contains fixed program variables, then the interpretation of these program variables is dependent on the prefix mapping θ.

Note that $\Vdash_\theta$ is a relation between structures, worlds, valuations and formulas. To express that the prefixed formula $\tau : \varphi$ holds in structure $\mathcal{M}$ under valuation ν and prefix mapping θ, we write $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \varphi$.

Proposition 59 Suppose that $\mathcal{I}$ is an interpretation in some structure $\mathcal{M}$ and that θ is a prefix mapping for $\mathcal{M}$. Then, $\mathcal{I}^\theta(w_1) = \mathcal{I}^\theta(w_2)$ implies that for any expression E (formula or program) and valuation ν, we have

1. if E is a formula, then $\mathcal{M}, w_1, \nu \Vdash_\theta E$ implies $\mathcal{M}, w_2, \nu \Vdash_\theta E$, and
2. if E is a program, then for all worlds w_1' in $\mathcal{M}$, if $\langle w_1, w_1' \rangle \in (E)^{\mathcal{M},\nu,\theta}$, then there exists a w_2', such that $\langle w_2, w_2' \rangle \in (E)^{\mathcal{M},\nu,\theta}$, and $\mathcal{I}^\theta(w_1') = \mathcal{I}^\theta(w_2')$.

PROOF. By induction on the structure of E. Most cases are trivial. We show the most interesting ones. (The case for $=$ is similar to P. Cases $\wedge$, $\neg$, λ, ;, and $\cup$ are all trivial applications of the induction hypothesis.)

P. $\mathcal{M}, w_1, \nu \Vdash_\theta P(t_1, \ldots, t_n)$ iff $\langle (\nu \star \mathcal{I}^\theta)(w_1, t_1), \ldots, (\nu \star \mathcal{I}^\theta)(w_1, t_n) \rangle \in \mathcal{I}(P)$. By Proposition 56, we have for each i, that $(\nu \star \mathcal{I}^\theta)(w_1, t_i) = (\nu \star \mathcal{I}^\theta)(w_2, t_i)$, which gives us $\mathcal{M}, w_2, \nu \Vdash_\theta P(t_1, \ldots, t_n)$.

$\diamond$. $\mathcal{M}, w_1, \nu \Vdash_\theta \langle p \rangle \varphi$ iff there exists $w_1' \in \mathcal{W}$ such that $\langle w_1, w_1' \rangle \in (p)^{\mathcal{M},\nu,\theta}$ and $\mathcal{M}, w_1', \nu \Vdash_\theta \varphi$. By the induction hypothesis, there exists a world w_2 such that $\langle w_2, w_2' \rangle \in (p)^{\mathcal{M},\nu,\theta}$ and $\mathcal{I}^\theta(w_1') = \mathcal{I}^\theta(w_2')$. Then, by applying the induction hypothesis again, we get $\mathcal{M}, w_2', \nu \Vdash_\theta \varphi$. Thus $\mathcal{M}, w_2, \nu \Vdash_\theta \langle p \rangle \varphi$.

$:=$. Suppose $\langle w_1, w_1' \rangle \in (a := t)^{\mathcal{M},\nu,\theta}$. Then $\mathcal{I}^\theta(w_1') = \mathcal{I}^\theta(w_1)[a \mapsto t]$. But, since $\mathcal{I}^\theta(w_1) = \mathcal{I}^\theta(w_2)$, we have $\mathcal{I}^\theta(w_1)[a \mapsto t] = \mathcal{I}^\theta(w_2)[a \mapsto t]$. Then, there exists w_2' (namely w_1'), such that $\langle w_2, w_2' \rangle \in (a := t)^{\mathcal{M},\nu,\theta}$ and $\mathcal{I}^\theta(w_1') = \mathcal{I}^\theta(w_2')$.

$*$. Suppose $\langle w_1, w_1^n\rangle \in (p*)^{\mathcal{M},\nu,\theta}$. Then there exists a sequence of worlds $w_1, w_1^1, \ldots, w_1^n$, such that each pair of consecutive worlds in the sequence is a member of the relation $(p)^{\mathcal{M},\nu,\theta}$. Using the induction hypothesis, we get that there exists a world w_2^1, such that $\langle w_2, w_2^1\rangle \in (p)^{\mathcal{M},\nu,\theta}$ and $\mathcal{I}^\theta(w_1^1) = \mathcal{I}^\theta(w_2^1)$. Continuing in the same fashion $n-1$ times, we get that $\langle w_2, w_2^n\rangle \in (p*)^{\mathcal{M},\nu,\theta}$ and $\mathcal{I}^\theta(w_1^n) = \mathcal{I}^\theta(w_2^n)$.

?. Suppose $\langle w_1, w_1'\rangle \in (\varphi?)^{\mathcal{M},\nu,\theta}$. Then $w_1 = w_1'$ and $\mathcal{M}, w_1, \nu \Vdash_\theta \varphi$. By the induction hypothesis, we get $\mathcal{M}, w_2, \nu \Vdash_\theta \varphi$. Then, there exists a w_2' (namely w_2), such that $\langle w_2, w_2'\rangle \in (\varphi?)^{\mathcal{M},\nu,\theta}$ and $\mathcal{I}^\theta(w_1') = \mathcal{I}^\theta(w_2')$.

□

We define when a set of prefixed formulas is satisfiable.

Definition 60 (Satisfiability, validity for set of prefixed formulas) Let S be a set of prefixed formulas. Then S is *satisfiable in structure* $\mathcal{M}$ *under valuation* ν if there exists a prefix mapping θ for $\mathcal{M}$, such that

$$\text{for every } \tau{:}\varphi \in S, \text{ we have } \mathcal{M}, \theta(\tau), \nu \Vdash_\theta \varphi.$$

S is *satisfiable in structure* $\mathcal{M}$ if there exists a valuation under which S is satisfiable in $\mathcal{M}$. S is *satisfiable* if it is satisfiable in some structure.

S is *valid* if for every structure $\mathcal{M}$, valuation ν and prefix mapping θ for $\mathcal{M}$, we have

$$\text{for every } \tau{:}\varphi \in S, \text{ that } \mathcal{M}, \theta(\tau), \nu \Vdash_\theta \varphi.$$

There is a close correspondence between Definition 51 and Definition 60, which we formulate in the following proposition.

Proposition 61 Let φ be a subscript-free formula. Then, φ is locally satisfiable iff $\{\epsilon : \varphi\}$ is satisfiable, and φ is valid iff $\{\epsilon : \varphi\}$ is valid.

PROOF. When a subscript-free formula φ is locally satisfiable, there exists structure $\mathcal{M}$, world w in $\mathcal{M}$, and valuation ν, such that $\mathcal{M}, w, \nu \Vdash \varphi$. Take any prefix mapping θ for $\mathcal{M}$ such that $\theta(\epsilon) = w$.[4] Then it is easy to prove by induction, that $\mathcal{M}, w, \nu \Vdash \varphi$ iff $\mathcal{M}, \theta(\epsilon), \nu \Vdash_\theta \varphi$. For the other direction, let $w = \theta(\tau)$.

[4] It is easy to construct such a prefix mapping for any structure $\mathcal{M}$. Let $\theta(\epsilon) = w$. For any prefix of the form $[a := t]$, let $\theta([a := t])$ be any world w' such that $\mathcal{I}(w') = \mathcal{I}(w)[t \mapsto a]$. The world w' must exist in $\mathcal{M}$, since $\mathcal{M}$ satisfies the seriality condition. Continue inductively in the same way for longer prefixes.

A subscript-free formula φ is valid iff $\neg\varphi$ is not locally satisfiable. By the previous paragraph, $\neg\varphi$ is not locally satisfiable iff $\{\epsilon : \neg\varphi\}$ is not satisfiable. By Definition 60 above, $\{\epsilon : \neg\varphi\}$ is not satisfiable iff $\{\epsilon : \varphi\}$ is valid. $\square$

Tableau calculi

We introduce a calculus for the logic.

Definition 62 (Branch extension rule, premise, conclusion) The *branch extension rules* are given in Figure 13.1. They are all of the form

$$\frac{S_1 \cdots S_n}{T_1 \mid \cdots \mid T_m}$$

where the S_i's are prefixed formulas, and the T_j's are sets of prefixed formulas. The S_i's are called the *premises* of the rule and the T_j's are called the *conclusions* of the rule.

The intuition behind the extension rules is that whenever every premise S_i is satisfiable, then so are all formulas of some conclusion T_j.

The propositional rules and the rules for composition, test and choice are standard. The scoping rules are the same as in (Fitting and Mendelsohn 1998). The conclusion of the assignment rules is a prefixed formula, saying that φ (or $\neg\varphi$) holds in the new state named $\tau[a := t]$.

For equality we have the substituitivity rules. In these rules every occurrence of the fixed term t_1 is replaced with the fixed term t_2.

There are two axioms. The first tells us that the value of a in state $\tau[a := t]$ is the value t had in state τ. The second is a persistency axiom. It says that an assignment to a program variable b does not affect the values of any other program variables.

Definition 63 (Tableau derivation, branch) A *tableau derivation*, or simply *tableau* for a closed prefixed formula $\tau : \varphi$, is a (possibly infinite) tree of prefixed formulas, constructed in the following way.

1. First, add $\tau : \varphi$ to the tree. This will be called the *root* node.

2. Then, take any leaf node in the tree, such that the prefixed formulas $S_1, \ldots, S_n$ belong to the path from this leaf to the root and that there exists a branch extension rule of the form

$$\frac{S_1 \cdots S_n}{T_1 \mid \cdots \mid T_m}$$

and add $T_1, \ldots, T_m$ as linear paths of formulas anchored at the node.

We will identify a *branch* $\mathcal{B}$ of a tableau with the set of prefixed formulas in it. When we have a prefixed formula $\tau : \varphi \in \mathcal{B}$, we say that $\tau : \varphi$ *occurs in branch* $\mathcal{B}$.

The following proposition is easily proved by induction on the length of the derivation.

Proposition 64 If $\tau : \varphi$ is a closed prefixed formula, then every prefixed formula in a tableau derivation for $\tau : \varphi$ is closed too.

Definition 65 (Closed tableau, tableau proof) A tableau branch $\mathcal{B}$ is *closed* if there exists either

1. a formula φ, such that both $\tau : \varphi$ and $\tau : \neg\varphi$ occur in $\mathcal{B}$, or
2. a prefixed formula $\tau : \neg t = t$ in $\mathcal{B}$.

A branch is *open* if it is not closed. A tableau is *closed* if every branch in it is closed. A closed tableau for a closed prefixed formula $\epsilon : \varphi$ is also called a *tableau proof of* φ, or simply a *proof*.

It is easy to see that any tableau proof can be transformed into a tableau proof with finite branches. Then, since every branch of the new proof is finite and every rule in Figure 13.1 is finitely branching, by König's Lemma, the new proof must be finite.

So, if there exists a tableau proof using the rules of Figure 13.1, then there exists a finite proof.

Definition 66 (New, present) A prefix τ is *present in a branch* $\mathcal{B}$ if there exists a formula φ such that $\tau : \varphi \in \mathcal{B}$. A prefix τ is *new for a branch* $\mathcal{B}$ if it is not present.

Propositional rules

$$\frac{\tau : (\varphi \wedge \psi)}{\begin{array}{c}\tau : \varphi \\ \tau : \psi\end{array}}\ (\wedge) \qquad \frac{\tau : \neg(\varphi \wedge \psi)}{\tau : \neg\varphi \mid \tau : \neg\psi}\ (\neg\wedge) \qquad \frac{\tau : \neg\neg\varphi}{\tau : \varphi}\ (\neg\neg)$$

Rules for composition, test and choice

$$\frac{\tau : \langle p; q\rangle\varphi}{\tau : \langle p\rangle\langle q\rangle\varphi}\ (;) \qquad \frac{\tau : \langle\psi?\rangle\varphi}{\begin{array}{c}\tau : \varphi \\ \tau : \psi\end{array}}\ (?) \qquad \frac{\tau : \langle p \cup q\rangle\varphi}{\tau : \langle p\rangle\varphi \mid \tau : \langle q\rangle\varphi}\ (\cup)$$

$$\frac{\tau : \neg\langle p; q\rangle\varphi}{\tau : \neg\langle p\rangle\langle q\rangle\varphi}\ (\neg;) \qquad \frac{\tau : \neg\langle\psi?\rangle\varphi}{\tau : \neg\psi \mid \tau : \neg\varphi}\ (\neg?) \qquad \frac{\tau : \neg\langle p \cup q\rangle\varphi}{\begin{array}{c}\tau : \neg\langle p\rangle\varphi \\ \tau : \neg\langle q\rangle\varphi\end{array}}\ (\neg\cup)$$

Scoping rules

$$\frac{\tau : \{\lambda x \leftarrow t.\ \varphi(x)\}}{\tau : \varphi(t@\tau)}\ (\lambda) \qquad \frac{\tau : \neg\{\lambda x \leftarrow t.\ \varphi(x)\}}{\tau : \neg\varphi(t@\tau)}\ (\neg\lambda)$$

Assignment rules

$$\frac{\tau : \langle a := t\rangle\varphi}{\tau[a := t] : \varphi}\ (\Diamond) \qquad \frac{\tau : \neg\langle a := t\rangle\varphi}{\tau[a := t] : \neg\varphi}\ (\neg\Diamond)$$

Equality rules

$$\frac{\epsilon : t_1 = t_2 \quad \tau : \varphi(t_1)}{\tau : \varphi(t_2)}\ (=_L)\ \textit{where } t_1 \textit{ and } t_2 \textit{ are fixed.}$$

$$\frac{\epsilon : t_2 = t_1 \quad \tau : \varphi(t_1)}{\tau : \varphi(t_2)}\ (=_R)\ \textit{where } t_1 \textit{ and } t_2 \textit{ are fixed.}$$

Axioms

$$\frac{}{\epsilon : a_{\tau[a:=t]} = t@\tau}\ (=_1) \qquad \frac{}{\epsilon : a_{\tau[b:=t]} = a_\tau}\ (=_2)\ \textit{where } a \not\equiv b.$$

Fix rules

$$\frac{\tau : P(t_1, \ldots, t_n)}{\epsilon : P(t_1@\tau, \ldots, t_n@\tau)}\ (P_@) \qquad \frac{\tau : \neg P(t_1, \ldots, t_n)}{\epsilon : \neg P(t_1@\tau, \ldots, t_n@\tau)}\ (\neg P_@)$$

$$\frac{\tau : t_1 = t_2}{\epsilon : t_1@\tau = t_2@\tau}\ (=_@) \qquad \frac{\tau : \neg t_1 = t_2}{\epsilon : \neg t_1@\tau = t_2@\tau}\ (\neg=_@)$$

Star rules

$$\frac{\tau : \langle p*\rangle\varphi}{\begin{array}{l}\tau : \varphi \mid \tau : \neg\varphi \\ \qquad\quad \mid \tau : \langle p*\rangle\langle p\rangle\varphi\end{array}}\ (*) \qquad \frac{\tau : \neg\langle p*\rangle\varphi}{\begin{array}{c}\tau : \neg\varphi \\ \tau : \neg\langle p*\rangle\langle p\rangle\varphi\end{array}}\ (\neg*)$$

Figure 13.1: The basic tableau system

(1)	$\epsilon:$	$\neg\{\lambda x \leftarrow b.\ \neg\langle a := b\rangle \neg x = a\}$	(assumed)
(2)	$\epsilon:$	$\neg\neg\langle a := b\rangle \neg b_\epsilon = a$	(from 1 by $(\neg\lambda)$)
(3)	$\epsilon:$	$\langle a := b\rangle \neg b_\epsilon = a$	(from 2 by $(\neg\neg)$)
(4)	$\tau_1:$	$\neg b_\epsilon = a$	(from 3 by $(\Diamond)$)
(5)	$\epsilon:$	$a_{\tau_1} = b_\epsilon$	(by $(=_1)$)
(6)	$\epsilon:$	$\neg b_\epsilon = a_{\tau_1}$	(from 4 by $(=_@)$)
(7)	$\epsilon:$	$\neg b_\epsilon = b_\epsilon$	(from 5 and 6 by $(=_L)$)
		$\times$	(closed by 7)

where $\tau_1 \equiv [a := b]$.

Figure 13.2: Proof of $\{\lambda x \leftarrow b.\ [a := b]x = a\}$

13.1 DERIVABLE RULES

Any branch containing $\tau : \neg\top$ or $\tau : \bot$ can be extended to a closed branch. In the future we will use the rules below, but it is easy to see that any proof using these rules can be re-written to a proof using only the rules of Figure 13.1.

$$\frac{\tau : (\varphi \vee \psi)}{\tau : \varphi \mid \tau : \psi}\ (\vee) \qquad \frac{\tau : \neg(\varphi \vee \psi)}{\tau : \neg\varphi \\ \tau : \neg\psi}\ (\neg\vee)$$

$$\frac{\tau : (\varphi \supset \psi)}{\tau : \neg\varphi \mid \tau : \psi}\ (\supset) \qquad \frac{\tau : \neg(\varphi \supset \psi)}{\tau : \varphi \\ \tau : \neg\psi}\ (\neg\supset)$$

13.2 EXAMPLE PROOFS

Example 67 (Fundamental properties of assignments) Whenever a program variable is assigned to another, the value of the updated program variable is equal to the value the other program variable had before the assignment

$$\{\lambda x \leftarrow b.\ [a := b]x = a\}.$$

Since $[p]\varphi$ is an abbreviation of $\neg\langle p\rangle\neg\varphi$, we prove, in Figure 13.2, the formula $\{\lambda x \leftarrow b.\ \neg\langle a := b\rangle \neg x = a\}$. We continue to study the assignment, and give in Figure 13.3 proofs of

1. $\{\lambda x \leftarrow a.\ [a := f(a)]\{\lambda y \leftarrow a.\ y = f(x)\}\}$, and
2. $\{\lambda x \leftarrow a.\ \langle a := f(a)\rangle\{\lambda y \leftarrow a.\ y = f(x)\}\}$.

The use of the scoping operator λy in these formulas is redundant.

(1)	ϵ :	$\neg\{\lambda x \leftarrow a.\ \neg\langle a := f(a)\rangle\neg\{\lambda y \leftarrow a.\ y = f(x)\}\}$	(assumed)
(2)	ϵ :	$\neg\neg\langle a := f(a)\rangle\neg\{\lambda y \leftarrow a.\ y = f(a_\epsilon)\}$	(from 1 by $(\neg\lambda)$)
(3)	ϵ :	$\langle a := f(a)\rangle\neg\{\lambda y \leftarrow a.\ y = f(a_\epsilon)\}$	(from 2 by $(\neg\neg)$)
(4)	τ_1 :	$\neg\{\lambda y \leftarrow a.\ y = f(a_\epsilon)\}$	(from 3 by $(\diamond)$)
(5)	τ_1 :	$\neg a_{\tau_1} = f(a_\epsilon)$	(from 4 by $(\neg\lambda)$)
(6)	ϵ :	$\neg a_{\tau_1} = f(a_\epsilon)$	(from 5 by $(\neg{=_@})$)
(7)	ϵ :	$a_{\tau_1} = f(a_\epsilon)$	(by $(=_1)$)
		$\times$	(closed by 6 and 7)

(1)	ϵ :	$\neg\{\lambda x \leftarrow a.\ \langle a := f(a)\rangle\{\lambda y \leftarrow a.\ y = f(x)\}\}$	(assumed)
(2)	ϵ :	$\neg\langle a := f(a)\rangle\{\lambda y \leftarrow a.\ y = f(a_\epsilon)\}$	(by $(\neg\lambda)$)
(3)	τ_1 :	$\neg\{\lambda y \leftarrow a.\ y = f(a_\epsilon)\}$	(from 2 by $(\neg\diamond)$)
		$\vdots$	
		$\times$	

The second proof continues similarly to the first proof, recognizing that step 3 is the same as step 4 of the first proof. We use $\tau_1 \equiv [a := f(a)]$ in both proofs.

Figure 13.3: Proof of $\{\lambda x \leftarrow a.\ [a := f(a)]\{\lambda y \leftarrow a.\ y = f(x)\}\}$ and $\{\lambda x \leftarrow a.\ \langle a := f(a)\rangle\{\lambda y \leftarrow a.\ y = f(x)\}\}$.

Example 68 (Swap) We prove that swapping two variables change the value of one to the value of the other before the swap, i.e.

$$\{\lambda x \leftarrow a.\ [\mathrm{SWAP}(a,b)]x = b\},$$

(using the definition of SWAP from Page 61). Since $[p]$ abbreviates $\neg\langle p\rangle\neg$, we actually prove $\{\lambda x \leftarrow a.\ \neg\langle \mathrm{SWAP}(a,b)\rangle\neg x = b\}$. The proof is given in Figure 13.4.

Example 69 (Boxes and diamonds) In Figure 13.5, we prove the formula $[a := t \cup b := t]P \supset \langle a := t \cup b := t\rangle P$.

We make some remarks. The formula $\langle a := t \cup b := t\rangle P \supset [a := t \cup b := t]P$ is not provable, but for any atomic program $a := t$, both $\langle a := t\rangle\varphi \supset [a := t]\varphi$ and $[a := t]\varphi \supset \langle a := t\rangle\varphi$ are.

Example 70 (Commutation) We study a special case of the formula given on Page 61, which expresses that $[p]$ and λx commute in the formula $\{\lambda x \leftarrow a.\ [p]P(x)\}$, provided a is a locally rigid program variable, i.e. the program does not change the value of a,

$$\{\lambda x \leftarrow a.\ [p]x = a\} \supset (\{\lambda x \leftarrow a.\ [p]P(x)\} \supset [p]P(a)).$$

(1)	ϵ :	$\neg\{\lambda x \leftarrow a.\ \neg\langle a' := a; a := b; b := a'\rangle\neg x = b\}$	(assumed)
(2)	ϵ :	$\neg\neg\langle a' := a; a := b; b := a'\rangle\neg a_\epsilon = b$	(from 1 by $(\neg\lambda)$)
(3)	ϵ :	$\langle a' := a; a := b; b := a'\rangle\neg a_\epsilon = b$	(from 2 by $(\neg\neg)$)
(4)	ϵ :	$\langle a' := a\rangle\langle a := b; b := a'\rangle\neg a_\epsilon = b$	(from 3 by $(;)$)
(5)	τ_1 :	$\langle a := b; b := a'\rangle\neg a_\epsilon = b$	(from 4 by $(\Diamond)$)
(6)	ϵ :	$a'_{\tau_1} = a_\epsilon$	(by $(=_1)$)
(7)	τ_1 :	$\langle a := b\rangle\langle b := a'\rangle\neg a_\epsilon = b$	(from 5 by $(;)$)
(8)	τ_2 :	$\langle b := a'\rangle\neg a_\epsilon = b$	(from 7 by $(\Diamond)$)
(9)	ϵ :	$a_{\tau_2} = b_{\tau_1}$	(by $(=_1)$)
(10)	τ_3 :	$\neg a_\epsilon = b$	(from 8 by $(\Diamond)$)
(11)	ϵ :	$b_{\tau_3} = a'_{\tau_2}$	(by $(=_1)$)
(12)	ϵ :	$\neg a_\epsilon = b_{\tau_3}$	(from 10 by $(=_@)$)
(13)	ϵ :	$\neg a_\epsilon = a'_{\tau_2}$	(from 11 and 12 by $(=_L)$)
(14)	ϵ :	$a'_{\tau_2} = a'_{\tau_1}$	(by $(=_2)$)
(15)	ϵ :	$a'_{\tau_2} = a_\epsilon$	(from 6 and 14 by $(=_L)$)
(16)	ϵ :	$\neg a_\epsilon = a_\epsilon$	(from 13 and 15 by $(=_L)$)
		$\times$	(closed by 16)

where $\tau_1 \equiv [a' := a]$, $\tau_2 \equiv [a' := a][a := b]$, and
$\tau_3 \equiv [a' := a][a := b][b := a']$.

Figure 13.4: Proof of $\{\lambda x \leftarrow a.\ [\mathrm{SWAP}(a, b)]x = b\}$

(1)	ϵ :	$\neg(\neg\langle a := t \cup b := t\rangle\neg P \supset \langle a := t \cup b := t\rangle P)$	(assumed)
(2)	ϵ :	$\neg\langle a := t \cup b := t\rangle\neg P$	(from 1 by derived rule $(\neg\supset)$)
(3)	ϵ :	$\neg\langle a := t \cup b := t\rangle P$	(from 1 by derived rule $(\neg\supset)$)
(4)	ϵ :	$\neg\langle a := t\rangle\neg P$	(from 2 by $(\neg\cup)$)
(5)	ϵ :	$\neg\langle b := t\rangle\neg P$	(from 2 by $(\neg\cup)$)
(6)	ϵ :	$\neg\langle a := t\rangle P$	(from 3 by $(\neg\cup)$)
(7)	ϵ :	$\neg\langle b := t\rangle P$	(from 3 by $(\neg\cup)$)
(8)	τ_1 :	$\neg\neg P$	(from 4 by $(\neg\Diamond)$)
(9)	τ_1 :	P	(from 8 by $(\neg\neg)$)
(10)	τ_1 :	$\neg P$	(from 6 by $(\neg\Diamond)$)
		$\times$	(closed by 9 and 10)

where $\tau_1 = [a := t]$.

Figure 13.5: Proof of $[a := t \cup b := t]P \supset \langle a := t \cup b := t\rangle P$

(1)	$\epsilon:$	$\neg\neg(\varphi \wedge \neg\psi)$	(assumed)
(2)	$\epsilon:$	$(\varphi \wedge \neg\psi)$	(from 1 by $(\neg\neg)$)
(3)	$\epsilon:$	$\{\lambda x \leftarrow a.\ \neg\langle b := t\rangle\neg x = a\}$	(from 2 by $(\wedge)$)
(4)	$\epsilon:$	$\neg(\{\lambda x \leftarrow a.\ \neg\langle b := t\rangle\neg P(x)\} \supset \neg\langle b := t\rangle\neg P(a))$	(from 2 by $(\wedge)$)
(5)	$\epsilon:$	$\neg\neg(\{\lambda x \leftarrow a.\ \neg\langle b := t\rangle\neg P(x)\} \wedge \neg\neg\langle b := t\rangle\neg P(a))$	(4 without abbreviations)
(6)	$\epsilon:$	$\{\lambda x \leftarrow a.\ \neg\langle b := t\rangle\neg P(x)\} \wedge \neg\neg\langle b := t\rangle\neg P(a)$	(from 5 by $(\neg\neg)$)
(7)	$\epsilon:$	$\{\lambda x \leftarrow a.\ \neg\langle b := t\rangle\neg P(x)\}$	(from 6 by $(\wedge)$)
(8)	$\epsilon:$	$\neg\neg\langle b := t\rangle\neg P(a)$	(from 6 by $(\wedge)$)
(9)	$\epsilon:$	$\neg\langle b := t\rangle\neg P(a_\epsilon)$	(from 7 by (λ))
(10)	$\epsilon:$	$\langle b := t\rangle\neg P(a)$	(from 8 by $(\neg\neg)$)
(11)	$\tau_1:$	$\neg P(a)$	(from 10 by $(\Diamond)$)
(12)	$\epsilon:$	$\neg P(a_{\tau_1})$	(from 11 by $(\neg P_@)$)
(13)	$\tau_1:$	$\neg\neg P(a_\epsilon)$	(from 9 by $(\neg\Diamond)$)
(14)	$\tau_1:$	$P(a_\epsilon)$	(from 13 by $(\neg\neg)$)
(15)	$\epsilon:$	$a_{\tau_1} = a_\epsilon$	(by $(=_2)$)
(16)	$\epsilon:$	$\neg P(a_\epsilon)$	(from 12 and 15 by $(=_L)$)
(17)	$\epsilon:$	$P(a_\epsilon)$	(from 14 by $(P_@)$)
		$\times$	(closed by 16 and 17)

where $\tau_1 = [b := t]$.

Figure 13.6: Proof of $\{\lambda x \leftarrow a.\ [b := t]x = a\} \supset (\{\lambda x \leftarrow a.\ [b := t]P(x)\} \supset [b := t]P(a))$.

This is true for any program p. We choose here to prove it for the atomic program $b := t$ (where $b \not\equiv a$). After substituting this program for p above, expanding the $[p]$-abbreviation and negating the formula, we get:

$$\neg(\quad \{\lambda x \leftarrow a.\ \neg\langle b := t\rangle\neg x = a\} \supset (\{\lambda x \leftarrow a.\ \neg\langle b := t\rangle\neg P(x)\} \supset \neg\langle b := t\rangle\neg P(a)) \quad).$$

This formula is of the form $\neg(\varphi \supset \psi)$, which is an abbreviation of $\neg\neg(\varphi \wedge \neg\psi)$. The proof is given in Figure 13.6.

Example 71 (Persistence) If a program variable a is assigned a value, then a should still have this value after another assignment has been made, provided that the new assignment is to another program variable. We prove $\{\lambda x \leftarrow t_1.\ [a := t_1][b := t_2]a = x\}$, and $\{\lambda x \leftarrow t_1.\ \langle a := t_1\rangle\langle b := t_2\rangle a = x\}$ in Figure 13.7.

Example 72 (Repeated assignment) In Figure 13.8, we prove the formula $\{\lambda x \leftarrow a.\ \langle(a := f(a))*\rangle a = f(f(x))\}$.

(1)	ϵ :	$\neg\{\lambda x \leftarrow t_1.\ \neg\langle a := t_1\rangle\neg\neg\langle b := t_2\rangle\neg a = x\}$	(assumed)
(2)	ϵ :	$\neg\neg\langle a := t_1\rangle\neg\neg\langle b := t_2\rangle\neg a = t_1@\epsilon$	(from 1 by $(\neg\lambda)$)
(3)	ϵ :	$\langle a := t_1\rangle\neg\neg\langle b := t_2\rangle\neg a = t_1@\epsilon$	(from 2 by $(\neg\neg)$)
(4)	τ_1 :	$\neg\neg\langle b := t_2\rangle\neg a = t_1@\epsilon$	(from 3 by $(\Diamond)$)
(5)	τ_1 :	$\langle b := t_2\rangle\neg a = t_1@\epsilon$	(from 4 by $(\neg\neg)$)
(6)	τ_2 :	$\neg a = t_1@\epsilon$	(from 5 by $(\Diamond)$)
(7)	ϵ :	$\neg a_{\tau_2} = t_1@\epsilon$	(from 6 by $(\neg=_@)$)
(8)	ϵ :	$a_{\tau_1} = t_1@\epsilon$	(by $(=_1)$)
(9)	ϵ :	$a_{\tau_2} = a_{\tau_1}$	(by $(=_2)$)
(10)	ϵ :	$\neg a_{\tau_1} = t_1@\epsilon$	(from 7 and 9 by $(=_L)$)
(11)	ϵ :	$\neg t_1@\epsilon = t_1@\epsilon$	(from 8 and 10 by $(=_L)$)
		$\times$	(closed by 11)

(1)	ϵ :	$\neg\{\lambda x \leftarrow t_1.\ \langle a := t_1\rangle\langle b := t_2\rangle a = x\}$	(assumed)
(2)	ϵ :	$\neg\langle a := t_1\rangle\langle b := t_2\rangle a = t_1@\epsilon$	(from 1 by $(\neg\lambda)$)
(3)	τ_1 :	$\neg\langle b := t_2\rangle a = t_1@\epsilon$	(from 2 by $(\neg\Diamond)$)
(4)	τ_2 :	$\neg a = t_1@\epsilon$	(from 3 by $(\neg\Diamond)$)
		$\vdots$	

The second proof continues similarly to the first, recognizing that step 4 is the same as step 6 of the first proof.
In both proofs, $\tau_1 = [a := t_1]$ and $\tau_2 = [a := t_1][b := t_2]$.

Figure 13.7: Proofs of $\{\lambda x \leftarrow t_1.\ [a := t_1][b := t_2]a = x\}$ and $\{\lambda x \leftarrow t_1.\ \langle a := t_1\rangle\langle b := t_2\rangle a = x\}$.

(1)	$\epsilon:$	$\neg\{\lambda x \leftarrow a.\ \langle (a := f(a))* \rangle a = f(f(x))\}$	(assumed)
(2)	$\epsilon:$	$\neg\langle (a := f(a))* \rangle a = f(f(a_\epsilon)$	(from 1 by $(\neg\lambda)$)
(3)	$\epsilon:$	$\neg a = f(f(a_\epsilon))$	(from 2 by $(\neg*)$)
(4)	$\epsilon:$	$\neg\langle (a := f(a))* \rangle \langle a := f(a) \rangle a = f(f(a_\epsilon))$	(from 2 by $(\neg*)$)
(5)	$\epsilon:$	$\neg\langle a := f(a) \rangle a = f(f(a_\epsilon))$	(from 4 by $(\neg*)$)
(6)	$\epsilon:$	$\neg\langle (a := f(a))* \rangle \langle a := f(a) \rangle \langle a := f(a) \rangle a = f(f(a_\epsilon))$	(from 4 by $(\neg*)$)
(7)	$\epsilon:$	$\neg\langle a := f(a) \rangle \langle a := f(a) \rangle a = f(f(a_\epsilon))$	(from 6 by $(\neg*)$)
(8)	$\epsilon:$	$\neg\langle (a := f(a))* \rangle \langle a := f(a) \rangle \langle a := f(a) \rangle \langle a := f(a) \rangle a = f(f(a_\epsilon))$	(from 6 by $(\neg*)$)
(9)	$\tau_1:$	$\neg\langle a := f(a) \rangle a = f(f(a_\epsilon))$	(from 7 by $(\neg\Diamond)$)
(10)	$\tau_2:$	$\neg a = f(f(a_\epsilon))$	(from 9 by $(\neg\Diamond)$)
(11)	$\epsilon:$	$\neg a_{\tau_2} = f(f(a_\epsilon))$	(from 10 by $(\neg=_@)$)
(12)	$\epsilon:$	$a_{\tau_1} = f(a_\epsilon)$	(by $(=_1)$)
(13)	$\epsilon:$	$a_{\tau_2} = f(a_{\tau_1})$	(by $(=_1)$)
(14)	$\epsilon:$	$\neg f(a_{\tau_1}) = f(f(a_\epsilon))$	(from 11 and 13 by $(=_L)$)
(15)	$\epsilon:$	$\neg f(f(a_\epsilon)) = f(f(a_\epsilon))$	(from 12 and 14 by $(=_L)$)
		$\times$	(closed by 15).

where $\tau_1 \equiv [a := f(a)]$ and $\tau_2 \equiv [a := f(a)][a := f(a)]$.

Figure 13.8: Proof of $\{\lambda x \leftarrow a.\ \langle (a := f(a))* \rangle a = f(f(x))\}$

Undecidability

First-order logic is undecidable. In some sense the dynamic assignment logic introduced here is weaker, since it is quantifier-free. Despite this, the validity problem for the logic is undecidable. We show this by considering Post's correspondence problem, see e.g. (Hopcroft and Ullman 1979). Our presentation is slightly different compared to (Hopcroft and Ullman 1979) to make our proofs easier to follow.

Definition 73 (PCP) An instance of *Post's correspondence problem* consists of two sequences $\mathcal{A} = A_1, \ldots, A_k$ and $\mathcal{B} = B_1, \ldots, B_k$ of words over some alphabet $\mathcal{F} = \{f_1, \ldots, f_n\}$. The instance is denoted $\mathcal{PCP}(\mathcal{A}, \mathcal{B})$. $\mathcal{PCP}(\mathcal{A}, \mathcal{B})$ has a *solution* if there exists a non-empty sequence of numbers $i_1, \ldots, i_m$, such that

$$A_{i_1} \cdots A_{i_m} = B_{i_1} \cdots B_{i_m},$$

where $A_{i_1} \cdots A_{i_m}$ denotes the concatenation of the words $A_{i_1}, \ldots, A_{i_m}$ and similarly for $B_{i_1} \cdots B_{i_m}$.

To express an instance $\mathcal{PCP}(\mathcal{A}, \mathcal{B})$ in quantifier-free dynamic assignment logic, we need to introduce a translation of words into terms. We want every word A to have a unique term representation in the signature $\langle \emptyset, \mathcal{F} \cup \{c\}, \emptyset \rangle$, where $\mathcal{F}$ is the alphabet of $\mathcal{PCP}(\mathcal{A}, \mathcal{B})$ (now considered to be unary function symbols) and c a constant symbol (not in $\mathcal{F}$). Any word $A = f_1 \cdots f_n$, is translated into $f_n(\cdots(f_1(c))\cdots)$. Note that the order of symbols in the term is reversed compared to the word, and that the empty word ε is translated to the constant symbol c. Note that every word A is translated to a unique term.

If $A = f_1 \cdots f_n$ and t is a term, we will write $A(t)$ as an abbreviation of the term $f_n(\cdots(f_1(t))\cdots)$. Concatenation of two words $A = f'_1 \cdots f'_n$ and $B = f''_1 \cdots f''_m$, i.e. $AB = f'_1 \cdots f'_n f''_1 \cdots f''_m$ is written

$$f''_m(\cdots(f''_1(f'_n(\cdots(f'_1(c))\cdots)))\cdots)$$

in the term representation.

For every $\mathcal{PCP}$-instance, we define a formula in the quantifier-free dynamic assignment logic, and show that the formula is valid iff the corresponding instance has a solution.

Definition 74 (PCP-formula) For any instance $\mathcal{PCP}(\mathcal{A}, \mathcal{B})$ with $\mathcal{A} = A_1, \ldots, A_k$ and $\mathcal{B} = B_1, \ldots, B_k$, we define the formula $\mathcal{PCP}^{\mathcal{F}}(\mathcal{A}, \mathcal{B})$ to be the following formula (where a and b are program variables not occurring in $\mathcal{F} \cup \{c\}$):

$$\langle a := b \rangle \langle p \rangle \langle p* \rangle a = b$$

where

$$p \equiv ((a := A_1(a); b := B_1(b)) \cup \cdots \cup (a := A_k(a); b := B_k(b))).$$

Before proving the undecidability theorem, we prove a lemma.

Lemma 75 The following implications hold.

1. If $i \in \{1, \ldots, n\}$, then

   $$\mathcal{M}, w, \nu \Vdash \langle p_i \rangle \varphi \text{ implies } \mathcal{M}, w, \nu \Vdash \langle p_1 \cup \cdots \cup p_n \rangle \varphi.$$

2. If $\{p_{i_1}, \ldots p_{i_m}\} \subseteq \{p_1, \ldots p_n\}$, then

   $$\mathcal{M}, w, \nu \Vdash \langle p_{i_1} \rangle \cdots \langle p_{i_m} \rangle \varphi \text{ implies } \mathcal{M}, w, \nu \Vdash \langle (p_1 \cup \cdots \cup p_n)* \rangle \varphi.$$

3. $\mathcal{M}, w, \nu \Vdash \langle (p_1 \cup \cdots \cup p_n)* \rangle \varphi$ implies that there exists a (possibly empty) sequence of programs $p_{i_2}, \ldots, p_{i_m}$, such that $\{p_{i_2}, \ldots p_{i_m}\} \subseteq \{p_1, \ldots p_n\}$ and $\mathcal{M}, w, \nu \Vdash \langle p_{i_2} \rangle \cdots \langle p_{i_m} \rangle \varphi$.

PROOF.

1. Assume that $\mathcal{M}, w, \nu \Vdash \langle p_i \rangle \varphi$ for some $i \in \{1, \ldots, n\}$. Then there exists w' such that $\langle w, w' \rangle \in (p_i)^{\mathcal{M},\nu}$ and $\mathcal{M}, w', \nu \Vdash \varphi$. But this means that $\langle w, w' \rangle \in (p_1)^{\mathcal{M},\nu} \cup \cdots \cup (p_i)^{\mathcal{M},\nu} \cup \cdots \cup (p_n)^{\mathcal{M},\nu}$, so therefore $\mathcal{M}, w, \nu \Vdash \langle p_1 \cup \cdots \cup p_n \rangle \varphi$.

2. If $\mathcal{M}, w_0, \nu \Vdash \langle p_{i_1} \rangle \cdots \langle p_{i_m} \rangle \varphi$, then there exists worlds $w_1, \ldots, w_m$ such that $\langle w_{j-1}, w_j \rangle \in (p_{i_j})^{\mathcal{M},\nu}$ for all $j \in \{1, \ldots, m\}$ and $\mathcal{M}, w_m, \nu \Vdash \varphi$. But then $\langle w_0, w_m \rangle$ must be in the reflexive and transitive closure of $(p_1)^{\mathcal{M},\nu} \cup \cdots \cup (p_n)^{\mathcal{M},\nu}$ (since $\{p_{i_1}, \ldots p_{i_m}\} \subseteq \{p_1, \ldots p_n\}$), so therefore $\mathcal{M}, w_0, \nu \Vdash \langle (p_1 \cup \cdots \cup p_n)* \rangle \varphi$.

3. If $\mathcal{M}, w, \nu \Vdash \langle (p_1 \cup \cdots \cup p_n)* \rangle \varphi$, then there exists a w_m such that $\langle w, w_m \rangle \in ((p_1 \cup \cdots \cup p_n)*)^{\mathcal{M},\nu}$ and $\mathcal{M}, w_m, \nu \Vdash \varphi$. This means that $\langle w, w_m \rangle$ is in the reflexive transitive closure of $(p_1 \cup \cdots \cup p_n)^{\mathcal{M},\nu}$, which implies that there exists a sequence of worlds $w = w_1, \ldots, w_m$ such that $\langle w_{j-1}, w_j \rangle \in (p_1 \cup \cdots \cup p_n)^{\mathcal{M},\nu}$, for every $j \in \{2, \ldots, m\}$.

   This, in turn, implies that for each of these pairs $\langle w_{j-1}, w_j \rangle$, there exists an index $i_j \in \{1, \ldots, n\}$, such that $\langle w_{j-1}, w_j \rangle \in (p_{i_j})^{\mathcal{M},\nu}$. But, then the formula $\mathcal{M}, w, \nu \Vdash \langle p_{i_2} \rangle \cdots \langle p_{i_m} \rangle \varphi$ holds too, since for any $j \in \{1, \ldots, m\}$, we have $\mathcal{M}, w_j, \nu \Vdash \langle p_{i_{j+1}} \rangle \cdots \langle p_{i_m} \rangle \varphi$.

□

Theorem 76 (Undecidability) The validity problem for quantifier-free dynamic assignment logic is undecidable.

PROOF. Since $\mathcal{PCP}$ is undecidable (Hopcroft and Ullman 1979), it is enough to prove that for any $\mathcal{PCP}(\mathcal{A}, \mathcal{B})$-instance, $\mathcal{PCP}(\mathcal{A}, \mathcal{B})$ is solvable iff $\mathcal{PCP}^{\mathcal{F}}(\mathcal{A}, \mathcal{B})$ is valid.

($\Rightarrow$) Suppose that $\mathcal{PCP}(\mathcal{A}, \mathcal{B})$ has a solution $i_1, \ldots, i_m$. We show that $\mathcal{PCP}^{\mathcal{F}}(\mathcal{A}, \mathcal{B})$ is valid.

We need to demonstrate for any structure $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I} \rangle$, valuation ν and world $w \in \mathcal{W}$, that

$$\mathcal{M}, w, \nu \Vdash \langle a := b \rangle \langle p \rangle \langle p* \rangle a = b$$

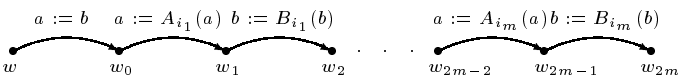
(where p is given in Definition 74).

By the seriality condition (Definition 48), there exists a world $w_0 \in \mathcal{W}$, such that for each $b \not\equiv a$, we have $\mathcal{I}(w_0, a) = \mathcal{I}(w, b)$ and $\mathcal{I}(w_0, b) = \mathcal{I}(w, b)$. This means that $\mathcal{I}(w_0) = \mathcal{I}(w)[a \mapsto b]$, so $\mathcal{M}, w, \nu \Vdash \langle a := b \rangle \langle p \rangle \langle p* \rangle a = b$ is true if $\mathcal{M}, w_0, \nu \Vdash \langle p \rangle \langle p* \rangle a = b$ is.

By Lemma 75:1 and 75:2, $\mathcal{M}, w_0, \nu \Vdash \langle p \rangle \langle p* \rangle a = b$ is implied by $\mathcal{M}, w_0, \nu \Vdash \langle p_{i_1} \rangle \cdots \langle p_{i_n} \rangle a = b$ (provided $\{p_{i_1}, \ldots p_{i_n}\} \subseteq \{p_1, \ldots, p_n\}$ and $\{p_{i_1}, \ldots p_{i_n}\}$ contains at least one program). It remains to show $\mathcal{M}, w_0, \nu \Vdash \langle p_{i_1} \rangle \cdots \langle p_{i_n} \rangle a = b$.

Suppose, for now, that there exists a sequence of worlds $w_1, \ldots, w_{2m} \in \mathcal{W}$ satisfying the conditions of Figure 14.1. We need to demonstrate that $\mathcal{M}, w_{2m}, \nu \Vdash a = b$. From the specification of the sequence of worlds, it follows that

$$\mathcal{I}(w_{2m})(a) = \mathcal{I}(w_0)[a \mapsto A_{i_m} \cdots A_{i_1}(a)](a), \text{ and}$$
$$\mathcal{I}(w_{2m})(b) = \mathcal{I}(w_0)[b \mapsto B_{i_m} \cdots B_{i_1}(b)](b).$$



(a) $\mathcal{I}(w_j) = \mathcal{I}(w_{j-1})[a \mapsto A_{i_{\lceil j/2 \rceil}}(a)]$, if $j \in \{1, 3, \ldots, 2m-1\}$.

(b) $\mathcal{I}(w_j) = \mathcal{I}(w_{j-1})[b \mapsto B_{i_{\lceil j/2 \rceil}}(b)]$, if $j \in \{2, 4, \ldots, 2m\}$.

Figure 14.1: Worlds $w_1, \ldots, w_{2m}$

We know that the words $A_{i_1} \cdots A_{i_m}$ and $B_{i_1} \cdots B_{i_m}$ are equal (since $i_1, \ldots, i_m$ is a solution to $\mathcal{PCP}(A, B)$), so the two terms $A_{i_n} \cdots A_{i_1}(a)$ and $B_{i_n} \cdots B_{i_1}(b)$ must have exactly the same function symbols. Let us call these functions symbols $f_1, \ldots, f_l$. We need to demonstrate that

$$\mathcal{I}(w_{2m}, a) = \mathcal{I}(f_1)(\mathcal{I}(f_2)(\cdots(\mathcal{I}(f_l)(\mathcal{I}(w_0, a)))\cdots))$$

is equal to

$$\mathcal{I}(w_{2m}, b) = \mathcal{I}(f_1)(\mathcal{I}(f_2)(\cdots(\mathcal{I}(f_l)(\mathcal{I}(w_0, b)))\cdots)).$$

But this is trivial since $\mathcal{I}(w_0) = \mathcal{I}(w)[a \mapsto b]$ gives us that $\mathcal{I}(w_0, a) = \mathcal{I}(w_0, b)$.

It remains to show that the worlds $w_1, \ldots, w_{2m}$ actually exist in $\mathcal{M}$. But this is easily shown by induction, using the seriality condition of structures. We leave this to the reader.

($\Leftarrow$) Suppose that $\mathcal{PCP}^{\mathcal{F}}(\mathcal{A}, \mathcal{B})$ is valid. We show that $\mathcal{PCP}(\mathcal{A}, \mathcal{B})$ has a solution.

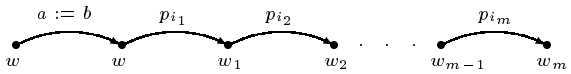
If $\mathcal{PCP}^{\mathcal{F}}(\mathcal{A}, \mathcal{B})$ is valid, then it holds in every world in every structure under any valuation ν. Thus we can specifically consider a structure in which terms are interpreted by words. We consider $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I} \rangle$, where:

1. $\mathcal{D}$ is the set of all words constructible from the alphabet $\mathcal{F}$,
2. $\mathcal{W}$ is the set of all pairs of words constructible from $\mathcal{F}$, and
3. $\mathcal{I}(w, a)$ is the first component of the pair w, $\mathcal{I}(w, b)$ is the second component of the pair w, $\mathcal{I}(c)$ is the empty word ε, and, for any $f \not\equiv c$, we have that $\mathcal{I}(f)(t)$ is the concatenation of the word represented by t and the single-letter word f.
   (Two examples: (1) If $(\nu \star \mathcal{I})(w, t) = f_1 f_2$, then $\mathcal{I}(f_3)(t) = f_1 f_2 f_3$. (2) If $(\nu \star \mathcal{I})(w, t) = \varepsilon$, then $\mathcal{I}(f_1)(t) = \varepsilon f_1 = f_1$.)

Let $w = \langle \varepsilon, \varepsilon \rangle$. Since $\mathcal{PCP}^{\mathcal{F}}(\mathcal{A}, \mathcal{B})$ is valid, we have

$$\mathcal{M}, w, \nu \Vdash \langle a := b \rangle \langle p \rangle \langle p* \rangle a = b.$$

We will now show that this implies that there exists a sequence of worlds, which could be pictured in the following way.



So, we know that $\mathcal{M}, w, \nu \Vdash \langle a := b\rangle\langle p\rangle\langle p*\rangle a = b$, but also that $\mathcal{I}(w) = \mathcal{I}(w)[a \mapsto b]$. Thus $\mathcal{M}, w, \nu \Vdash \langle p\rangle\langle p*\rangle a = b$. Since p is a choice construct of the form $p_1 \cup \cdots \cup p_n$, there must exist a world w_1 and an index $i \in \{1, \ldots, n\}$, such that $\langle w, w_1\rangle \in (p_i)^{\mathcal{M},\nu}$ and $\mathcal{M}, w_1, \nu \Vdash \langle p*\rangle a = b$. Set i_1 to be this index i.

Expanding $p*$, we now know that $\mathcal{M}, w_1, \nu \Vdash \langle (p_1 \cup \cdots \cup p_n)*\rangle\varphi$. By Lemma 75:3, there exists a sequence of programs $p_{i_2}, \ldots, p_{i_m}$, such that $\{p_{i_2}, \ldots, p_{i_m}\} \subseteq \{p_1, \ldots, p_n\}$ and $\mathcal{M}, w_1, \nu \Vdash \langle p_{i_2}\rangle \cdots \langle p_{i_m}\rangle\varphi$. Then it is easy to see that there must exist worlds $w_2, \ldots, w_m$, such that $\langle w_{j-1}, w_j\rangle \in (p_{i_j})^{\mathcal{M},\nu}$ for all $j \in \{2, \ldots, m\}$ and $\mathcal{M}, w_m, \nu \Vdash a = b$.

For any $j \in \{2, \ldots, m\}$, we have that $\mathcal{I}(w_j)(a)$ is A_{i_j} concatenated with $\mathcal{I}(w_{j-1})(a)$. And similarly for b. We get that $A_{i_1} \cdots A_{i_m} = \mathcal{I}(w_m, a) = \mathcal{I}(w_m, b) = B_{i_1} \cdots B_{i_m}$. Therefore, a solution exists to $\mathcal{PCP}(\mathcal{A}, \mathcal{B})$, namely $i_1, \ldots, i_m$.

□

Note that the formula $\mathcal{PCP}^{\mathcal{F}}(\mathcal{A}, \mathcal{B})$ in the proof is in signature $\langle \emptyset, \mathcal{F}, \{a, b\}\rangle$, implying that the validity is undecidable using only two program variables and two function symbols. (It is easy to encode any word into the binary representation of it, thus needing only two letters, i.e. function symbols.)

Substituitivity

Our next concern is substituitivity of terms in formulas. We start by considering substituitivity of terms in terms.

15.1 SUBSTITUITIVITY IN TERMS

In this section, we will use the following lemma, which is easily proved by induction on t.

Lemma 77 Let θ be a prefix mapping for a structure $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I} \rangle$. Then, for any subscripted term t and valuations ν', ν'', if $\nu'(z) = \nu''(z)$ for all logic variables z in t, then $(\nu' \star \mathcal{I}^\theta)(w, t) = (\nu'' \star \mathcal{I}^\theta)(w, t)$ for all worlds $w \in \mathcal{W}$.

If ν' and ν'' agree on all logic variables in t, and if t' and t'' have the same meaning under both ν' and ν'', then t'' can be substituted for t' in the term t. We state it formally in the following Lemma.

Lemma 78 (Substitution in terms) Let $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I} \rangle$ be a structure, and let $\sigma' = [y \mapsto t']$ and $\sigma'' = [y \mapsto t'']$ be two substitutions. Then, for any subscripted term t, valuations ν', ν'', and extended interpretation $\mathcal{I}^\theta$ (where θ is a prefix mapping for $\mathcal{M}$), we have that the following two premises

1. for every logic variable $z \not\equiv y$ in t, we have $\nu'(z) = \nu''(z)$, and,
2. for every $w \in \mathcal{W}$, we have $(\nu' \star \mathcal{I}^\theta)(w, t') = (\nu'' \star \mathcal{I}^\theta)(w, t'')$,

imply that

for every $w \in \mathcal{W}$, we have $(\nu' \star \mathcal{I}^\theta)(w, t\sigma') = (\nu'' \star \mathcal{I}^\theta)(w, t\sigma'')$.

PROOF. By induction on t.

a, x. Suppose t is a program variable a or a logic variable $x \not\equiv y$. Then $t\sigma' \equiv t \equiv t\sigma''$, so the lemma holds by Lemma 77.

y. Suppose t is the logic variable y. Then $t\sigma' \equiv t'$ and $t\sigma'' \equiv t''$, so the lemma holds by premise 2.

f. Suppose t is a subscripted term $f(t_1, \ldots, t_n)$, with $n \geq 0$. Then, for any w,

$$\begin{array}{ll} (\nu' \star \mathcal{I}^\theta)(w, f(t_1, \ldots, t_n)\sigma') & = \\ (\nu' \star \mathcal{I}^\theta)(w, f(t_1\sigma', \ldots, t_n\sigma')) & = \\ \mathcal{I}^\theta(f)((\nu' \star \mathcal{I}^\theta)(w, t_1\sigma'), \ldots, (\nu' \star \mathcal{I}^\theta)(w, t_n\sigma')) & \end{array}$$

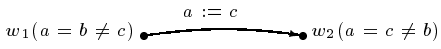
which by the induction hypothesis is equal to

$$\begin{array}{ll} \mathcal{I}^\theta(f)((\nu'' \star \mathcal{I}^\theta)(w, t_1\sigma''), \ldots, (\nu'' \star \mathcal{I}^\theta)(w, t_n\sigma'')) & = \\ (\nu'' \star \mathcal{I}^\theta)(w, f(t_1\sigma'', \ldots, t_n\sigma'')) & = \\ (\nu'' \star \mathcal{I}^\theta)(w, f(t_1, \ldots, t_n)\sigma''). & \end{array}$$

□

We make a remark on the second premise to provide some intuition as to why it is required to hold for *every* world.

Example 79 Consider the following partial picture of a structure $\mathcal{M}$.



In this picture a and b have the same meaning in world w_1, but different meanings in world w_2, i.e.

1. $(\nu \star \mathcal{I}^\theta)(w_1, a) = (\nu \star \mathcal{I}^\theta)(w_1, b)$, but
2. $(\nu \star \mathcal{I}^\theta)(w_2, a) \not\equiv (\nu \star \mathcal{I}^\theta)(w_2, b)$.

So, if premise 2 would hold for only one world (w_1), the conclusion would not follow for all worlds (especially not w_2).

15.2 SUBSTITUITIVITY IN FORMULAS AND PROGRAMS

In the next proposition, we generalize Lemma 78 to formulas and programs.

Proposition 80 (Substitution in formulas and programs) Let $\sigma' = [y \mapsto t']$ and $\sigma'' = [y \mapsto t'']$ be two substitutions free for an expression E (either a formula or a program). Then, for any structure $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I} \rangle$, prefix mapping θ for $\mathcal{M}$, and valuations ν', ν'', we have that the following two premises

1. For every logic variable $z \not\equiv y$ in E, we have $\nu'(z) = \nu''(z)$.
2. For every $w \in \mathcal{W}$, we have $(\nu' \star \mathcal{I}^\theta)(w, t') = (\nu'' \star \mathcal{I}^\theta)(w, t'')$.

imply that

$$\begin{cases} \mathcal{M}, w, \nu' \Vdash_\theta E\sigma' \text{ iff } \mathcal{M}, w, \nu'' \Vdash_\theta E\sigma'', & \text{if } E \text{ is a formula} \\ (E\sigma')^{\mathcal{M},\nu',\theta} = (E\sigma'')^{\mathcal{M},\nu'',\theta}, & \text{if } E \text{ is a program.} \end{cases}$$

PROOF. Proof by induction on E.

P. Suppose $\mathcal{M}, w, \nu' \Vdash_\theta P(t_1, \ldots, t_n)\sigma'$. Then we have that $\mathcal{M}, w, \nu' \Vdash_\theta P(t_1\sigma', \ldots, t_n\sigma')$. Thus, $\langle (\nu' \star \mathcal{I}^\theta)(w, t_1\sigma'), \ldots, (\nu' \star \mathcal{I}^\theta)(w, t_n\sigma') \rangle \in \mathcal{I}(P)$. By Term Substitution Lemma 78, $(\nu' \star \mathcal{I}^\theta)(w, t_i\sigma') = (\nu'' \star \mathcal{I}^\theta)(w, t_i\sigma'')$, for each t_i.

So, $\langle (\nu'' \star \mathcal{I}^\theta)(w, t_1\sigma''), \ldots, (\nu'' \star \mathcal{I}^\theta)(w, t_n\sigma'') \rangle \in \mathcal{I}(P)$. And thus $\mathcal{M}, w, \nu'' \Vdash_\theta P(t_1\sigma'', \ldots, t_n\sigma'')$ giving us $\mathcal{M}, w, \nu'' \Vdash_\theta P(t_1, \ldots, t_n)\sigma''$.

=. Suppose $\mathcal{M}, w, \nu' \Vdash_\theta (t_1 = t_2)\sigma'$. Then $(\nu' \star \mathcal{I}^\theta)(w, t_1\sigma') = (\nu' \star \mathcal{I}^\theta)(w, t_2\sigma')$. Then, by applying Term Substitution Lemma 78 twice, we get $(\nu'' \star \mathcal{I}^\theta)(w, t_1\sigma'') = (\nu' \star \mathcal{I}^\theta)(w, t_1\sigma') = (\nu' \star \mathcal{I}^\theta)(w, t_2\sigma') = (\nu'' \star \mathcal{I}^\theta)(w, t_2\sigma'')$, so $(\nu'' \star \mathcal{I}^\theta)(w, t_1\sigma'') = (\nu'' \star \mathcal{I}^\theta)(w, t_2\sigma'')$, giving us $\mathcal{M}, w, \nu'' \Vdash_\theta (t_1 = t_2)\sigma''$.

The cases for $\wedge$ and $\neg$ are trivial and left to the reader.

$\diamond$. Suppose $\mathcal{M}, w, \nu' \Vdash_\theta (\langle p \rangle \varphi)\sigma'$. Then there exists a world w' such that $\langle w, w' \rangle \in (p\sigma')^{\mathcal{M},\nu',\theta}$ and $\mathcal{M}, w', \nu' \Vdash_\theta \varphi\sigma'$. By the induction hypothesis, $(p\sigma')^{\mathcal{M},\nu',\theta} = (p\sigma'')^{\mathcal{M},\nu'',\theta}$, so $\langle w, w' \rangle \in (p\sigma'')^{\mathcal{M},\nu'',\theta}$. The induction hypothesis also gives us $\mathcal{M}, w', \nu'' \Vdash_\theta \varphi\sigma''$. Thus $\mathcal{M}, w, \nu'' \Vdash_\theta (\langle p \rangle \varphi)\sigma''$.

λ. Assume $\mathcal{M}, w, \nu' \Vdash_\theta (\{\lambda x \leftarrow t.\ \varphi(x)\})\sigma'$, i.e. $\mathcal{M}, w, \nu' \Vdash_\theta \{\lambda x \leftarrow t\sigma'.\ \varphi(x)\sigma'\}$. Then $\mathcal{M}, w, \nu'[x \mapsto (t\sigma')^w] \Vdash_\theta \varphi(x)\sigma'$. Provided we can show the following two facts

1. for every logic variable $z \not\equiv y$ in $\varphi(x)$, we have

$$\nu'[x \mapsto (t\sigma')^w](z) = \nu''[x \mapsto (t\sigma'')^w](z), \text{ and,}$$

2. for every $w' \in \mathcal{W}$, we have

$$(\nu'[x \mapsto (t\sigma')^w] \star \mathcal{I}^\theta)(w', t') = (\nu''[x \mapsto (t\sigma'')^w] \star \mathcal{I}^\theta)(w', t''),$$

then the induction hypothesis would give us $\mathcal{M}, w, \nu''[x \mapsto t\sigma''^w] \Vdash_\theta \varphi(x)\sigma''$, and thus $\mathcal{M}, w, \nu'' \Vdash_\theta (\{\lambda x \leftarrow t.\ \varphi(x)\})\sigma''$.

But, fact 1 is immediate from the first premise when $z \not\equiv x$. And when $z \equiv x$, we have that $\nu'[x \mapsto (t\sigma')^w](z) = (\nu' \star \mathcal{I}^\theta)(w, t\sigma') = (\nu'' \star \mathcal{I}^\theta)(w, t\sigma'') = \nu''[x \mapsto (t\sigma'')^w](z)$, by Term Substitution Lemma 78.

Fact 2 follows from premise 2, since x does not occur in either t' or t''. (Recall that both σ' or σ'' are free substitutions for E, and that x is bound in E.)

$:=$. Trivial. Since assignment programs do not contain any logic variables, $(a := t)\sigma' \equiv a := t \equiv (a := t)\sigma''$, so $((a := t)\sigma')^{\mathcal{M},\nu',\theta} = ((a := t)\sigma'')^{\mathcal{M},\nu',\theta}$.

$;$. $((p; q)\sigma')^{\mathcal{M},\nu',\theta} = (p\sigma')^{\mathcal{M},\nu',\theta} \circ (q\sigma')^{\mathcal{M},\nu',\theta}$, which by the induction hypothesis is equal to $(p\sigma'')^{\mathcal{M},\nu'',\theta} \circ (q\sigma'')^{\mathcal{M},\nu'',\theta} = ((p; q)\sigma'')^{\mathcal{M},\nu'',\theta}$.

$\cup$. Similar to previous case.

$*$. Immediate, since $((p*)\sigma')^{\mathcal{M},\nu',\theta}$ is the reflexive and transitive closure of $(p\sigma')^{\mathcal{M},\nu',\theta}$ which is equal to $(p\sigma'')^{\mathcal{M},\nu'',\theta}$ by the induction hypothesis.

$?$. $((\varphi?)\sigma')^{\mathcal{M},\nu',\theta} = \{\langle w, w\rangle \mid \mathcal{M}, w, \nu' \Vdash_\theta \varphi\sigma'\}$, which by the induction hypothesis is equal to $\{\langle w, w\rangle \mid \mathcal{M}, w, \nu'' \Vdash_\theta \varphi\sigma''\} = ((\varphi?)\sigma'')^{\mathcal{M},\nu'',\theta}$.

□

We study an example of the second premise.

Example 81 Consider the picture in Example 79. Let $\nu'(x) = \nu''(x)$ and let $\varphi(x) \equiv \langle a := c\rangle\langle x = c?\rangle\top$. Then consider the formulas $\varphi(a)$ and $\varphi(b)$. In world w_1, the first formula holds, but the second is false. This shows us that substituting a or b for x in a formula $\varphi(x)$ might change the truth value of the formula even though a and b denote the same value in one world.

Two special cases of the proposition are especially noteworthy. They are later used in the proof of the Rule Correctness Lemma 84.

In the first corollary we have the same valuation ν for both terms.

Corollary 82 (Substitution, same valuation) For any structure $\mathcal{M} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I} \rangle$, formula $\varphi(x)$, program $p(x)$, prefix mapping θ for $\mathcal{M}$, and valuation ν, such that $(\nu \star \mathcal{I}^\theta)(w, t') = (\nu \star \mathcal{I}^\theta)(w, t'')$ holds in every world $w \in \mathcal{W}$, and no logic variable occurring in t' or t'' is bound in $\varphi(x)$ or $p(x)$, we have that

1. $\mathcal{M}, w, \nu \Vdash_\theta \varphi(t')$ iff $\mathcal{M}, w, \nu \Vdash_\theta \varphi(t'')$, and
2. $(p(t'))^{\mathcal{M},\nu,\theta} = (p(t''))^{\mathcal{M},\nu,\theta}$.

PROOF. Immediate from Proposition 80. □

In the second corollary we note that the logic variable x in an updated valuation has the same meaning as the subscripted term $t@\tau$.

Corollary 83 (Substitution, fixed term) For any structure $\mathcal{M}$, prefix τ, prefix mapping θ for $\mathcal{M}$, and valuation ν, the following holds for any ground term t:

$$\mathcal{M}, \theta(\tau), \nu[x \mapsto t^{\theta(\tau)}] \Vdash_\theta \varphi(x) \quad \text{iff} \quad \mathcal{M}, \theta(\tau), \nu \Vdash_\theta \varphi(t@\tau).$$

PROOF. We consider the expression $E \equiv \varphi(x)$. Let σ' be the identity mapping and $\sigma'' \stackrel{def}{=} [x \mapsto t@\tau]$. The identity substitution is free for any expression, and σ'' is free for E, since t (and thus $t@\tau$) does not contain any logic variables.

Furthermore, let $\nu' \stackrel{def}{=} \nu[x \mapsto t^{\theta(\tau)}]$, $\nu'' \stackrel{def}{=} \nu$, $t' \equiv x$, and $t'' \equiv t@\tau$. Then the result follows from Proposition 80, since the two premises are satisfied.

The first premise is satisfied, since for any $z \not\equiv x$, we have

$$\nu'(z) = (\nu[x \mapsto t^{\theta(\tau)}] \star \mathcal{I}^\theta)(z) = \nu[x \mapsto t^{\theta(\tau)}](z) = \nu(z) = \nu''(z).$$

The second premise is satisfied, since for every $w \in \mathcal{W}$, we have

$$\begin{array}{ll} (\nu' \star \mathcal{I}^\theta)(w, t') & = \\ (\nu[x \mapsto t^{\theta(\tau)}] \star \mathcal{I}^\theta)(w, x) & = \\ \nu[x \mapsto t^{\theta(\tau)}](x) & = \\ (\nu \star \mathcal{I}^\theta)(\theta(\tau), t) & = [1] \\ (\nu \star \mathcal{I}^\theta)(w, t@\tau) & = \\ (\nu'' \star \mathcal{I}^\theta)(w, t''). & \end{array}$$

□

[1] By Proposition 57.

Soundness

We start by proving that each branch extension rule of the basic tableau system is correct with respect to the semantics.

Lemma 84 (Rule correctness) All branch extension rules of Figure 13.1 preserves satisfiability (under the same valuation, in the same structure).

Proof. We assume that a branch $\mathcal{B}$ is satisfiable in some structure $\mathcal{M}$ under valuation ν (and prefix mapping θ for $\mathcal{M}$). We show that if $\mathcal{B}$ contains the premises of a rule, then adding the set of formulas of some conclusion of the rule to $\mathcal{B}$ still produces a set of prefixed formulas satisfiable in $\mathcal{M}$ under ν.

It is easy to show that the Lemma holds for the propositional rules ($\wedge$), ($\neg\wedge$), and ($\neg\neg$), so we skip these cases of the proof.

For rules ($\neg;$), ($;$), ($\neg?$), ($?$), ($\neg\cup$), and ($\cup$), we use the fact that

3. $\mathcal{M}, w, \nu \Vdash_\theta \langle p; q\rangle\varphi$ iff $\mathcal{M}, w, \nu \Vdash_\theta \langle p\rangle\langle q\rangle\varphi$,
4. $\mathcal{M}, w, \nu \Vdash_\theta \langle p \cup q\rangle\varphi$ iff $\mathcal{M}, w, \nu \Vdash_\theta (\langle p\rangle\varphi \vee \langle q\rangle\varphi)$, and
5. $\mathcal{M}, w, \nu \Vdash_\theta \langle \psi?\rangle\varphi$ iff $\mathcal{M}, w, \nu \Vdash_\theta \psi \wedge \varphi$.

(Propositions 50:3, 50:4 and 50:5). We show only one case, the other cases are similar.

($\cup$) Suppose $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \langle p \cup q\rangle\varphi$. Proposition 50:4, gives us that $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta (\langle p\rangle\varphi \vee \langle q\rangle\varphi)$, and by Proposition 50:1, we get that either $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \langle p\rangle\varphi$ or $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \langle q\rangle\varphi$.

The case for rule (λ) is straightforward using Corollary 83.

(λ) Suppose $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \{\lambda x \leftarrow t.\, \varphi(x)\}$. Then $\mathcal{M}, \theta(\tau), \nu[x \mapsto t^{\theta(\tau)}] \Vdash_\theta \varphi(x)$. Note that t is ground by Proposition 64. We get $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \varphi(t@\tau)$ by Corollary 83.

The case for $(\neg\lambda)$ uses similar reasoning. The rest of the cases follow.

$(\Diamond)$ Suppose $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \langle a := t\rangle\varphi$. We show $\mathcal{M}, \theta(\tau[a := t]), \nu \Vdash_\theta \varphi$.

By our assumption, there exists a world $w' \in \mathcal{W}$ such that $\langle\theta(\tau), w'\rangle \in (a := t)^{\mathcal{M},\nu,\theta}$ and $\mathcal{M}, w', \nu \Vdash_\theta \varphi$.

But, $\langle\theta(\tau), w'\rangle \in (a := t)^{\mathcal{M},\nu,\theta}$ implies that $\mathcal{I}^\theta(w') = \mathcal{I}^\theta(\theta(\tau))[t \mapsto a]$. And, since θ is a prefix mapping for $\mathcal{M}$, we have $\mathcal{I}^\theta(\tau[a := t]) = \mathcal{I}^\theta(\theta(\tau))[t \mapsto a]$. But then, $\mathcal{I}^\theta(w') = \mathcal{I}^\theta(\tau[a := t])$, so Proposition 59 gives us $\mathcal{M}, \theta(\tau[a := t]), \nu \Vdash_\theta \varphi$.

$(\neg\Diamond)$ Suppose $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \neg\langle a := t\rangle\varphi$. We show $\mathcal{M}, \theta(\tau[a := t]), \nu \Vdash_\theta \neg\varphi$.

By our assumption, for every world $w' \in \mathcal{W}$ such that $\langle\theta(\tau), w'\rangle \in (a := t)^{\mathcal{M},\nu,\theta}$, we have that $\mathcal{M}, w', \nu \Vdash_\theta \varphi$ does *not* hold, which means that $\mathcal{M}, w', \nu \Vdash_\theta \neg\varphi$ holds. Since θ is a prefix mapping for $\mathcal{M}$, we have that $\langle\theta(\tau), \theta(\tau[a := t])\rangle \in (a := t)^{\mathcal{M},\nu,\theta}$, so $\mathcal{M}, \theta(\tau[a := t]), \nu \Vdash_\theta \neg\varphi$.

$(=_L)$ Suppose $\mathcal{M}, \theta(\epsilon), \nu \Vdash_\theta t_1 = t_2$, $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \varphi(t_1)$ and that t_1 and t_2 are fixed. We need to show $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \varphi(t_2)$.

Using $(\nu \star \mathcal{I}^\theta)(\theta(\epsilon), t_1) = (\nu \star \mathcal{I}^\theta)(\theta(\epsilon), t_2)$, the fact that $t_1 \equiv t_1@\epsilon$ and $t_2 \equiv t_2@\epsilon$ (since t_1 and t_2 are fixed), and Proposition 57, we get for any world w, that $(\nu \star \mathcal{I}^\theta)(w, t_1) = (\nu \star \mathcal{I}^\theta)(w, t_2)$.[1] Then, by Corollary 82, $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \varphi(t_2)$.

The case for $(=_R)$ is similar.

$(=_1)$ We need to show $\mathcal{M}, \theta(\epsilon), \nu \Vdash_\theta a_{\tau[a:=t]} = t@\tau$. Using Proposition 57, we have

$$\begin{array}{lll} (\nu \star \mathcal{I}^\theta)(\theta(\epsilon), a_{\tau[a:=t]}) & = & \mathcal{I}^\theta(\theta(\epsilon), a_{\tau[a:=t]}) \quad = \\ \mathcal{I}(\theta(\tau[a := t]), a) & = & \mathcal{I}(\theta(\tau))[a \mapsto t](a) \quad = \\ (\nu \star \mathcal{I})(\theta(\tau), t) & =^{2} & (\nu \star \mathcal{I}^\theta)(\theta(\tau), t) \quad = \\ (\nu \star \mathcal{I}^\theta)(\theta(\epsilon), t@\tau). & & \end{array}$$

The case for $(=_2)$ is similar. The cases for $(P_@)$, $(\neg P_@)$, $(=_@)$, and $(\neg=_@)$ all use similar reasoning. We show the case for $(P_@)$.

$(P_@)$ Suppose $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta P(t_1, \ldots, t_n)$.

Then $\langle(\nu \star \mathcal{I}^\theta)(\theta(\tau), t_1), \ldots, (\nu \star \mathcal{I}^\theta)(\theta(\tau), t_n)\rangle \in \mathcal{I}(P)$. Proposition 57 gives us $(\nu \star \mathcal{I}^\theta)(\theta(\tau), t_1) = (\nu \star \mathcal{I}^\theta)(\theta(\epsilon), t_1@\tau)$. So, using this on every t_i, we get $\langle(\nu \star \mathcal{I}^\theta)(\theta(\epsilon), t_1@\tau), \ldots, (\nu \star \mathcal{I}^\theta)(\theta(\epsilon), t_n@\tau)\rangle \in \mathcal{I}(P)$, which gives us $\mathcal{M}, \theta(\epsilon), \nu \Vdash_\theta P(t_1@\tau, \ldots, t_n@\tau)$.

[1] $(\nu \star \mathcal{I}^\theta)(w, t_1) = (\nu \star \mathcal{I}^\theta)(w, t_1@\epsilon) = (\nu \star \mathcal{I}^\theta)(\theta(\epsilon), t_1) = (\nu \star \mathcal{I}^\theta)(\theta(\epsilon), t_2) = (\nu \star \mathcal{I}^\theta)(w, t_2@\epsilon) = (\nu \star \mathcal{I}^\theta)(w, t_2)$.

[2] Since t occurs in a prefix, it must be subscript-free.

(∗) Suppose $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \langle p* \rangle \varphi$. Then there exists a world w' such that $\langle \theta(\tau), w' \rangle \in (p*)^{\mathcal{M},\nu,\theta}$ and $\mathcal{M}, w', \nu \Vdash_\theta \varphi$. Then there exists a sequence of worlds $\theta(\tau) = w_1, \ldots, w_n = w'$, such that $\mathcal{M}, w_n, \nu \Vdash_\theta \varphi$ and $\langle w_i, w_{i+1} \rangle \in (p)^{\mathcal{M},\nu,\theta}$, whenever $0 < i < n$ Let $w_1, \ldots, w_n$ be a minimal such sequence.

If $n = 1$, then $w_n = \theta(\tau)$, so $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \varphi$.

If $n > 1$, then $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \neg\varphi$, since if not, then the sequence would not be minimal. Also, we have that $\mathcal{M}, w_{n-1}, \nu \Vdash_\theta \langle p \rangle \varphi$ and $\langle w_1, w_{n-1} \rangle \in (p*)^{\mathcal{M},\nu,\theta}$, so $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \langle p* \rangle \langle p \rangle \varphi$.

(¬∗) Suppose $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \neg\langle p* \rangle \varphi$. Then there exists no world w such that $\langle \theta(\tau), w \rangle \in (p*)^{\mathcal{M},\nu,\theta}$ and $\mathcal{M}, w, \nu \Vdash_\theta \varphi$. We need to show $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \neg\varphi$ and $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \neg\langle p* \rangle \langle p \rangle \varphi$.

Since $(p*)^{\mathcal{M},\nu,\theta}$ is reflexive, we have $\langle \theta(\tau), \theta(\tau) \rangle \in (p*)^{\mathcal{M},\nu,\theta}$. Thus $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \neg\varphi$.

Consider any two worlds w_1 and w_2, such that $\langle \theta(\tau), w_1 \rangle \in (p*)^{\mathcal{M},\nu,\theta}$ and $\langle w_1, w_2 \rangle \in (p)^{\mathcal{M},\nu,\theta}$. Then, since the relation $(p*)^{\mathcal{M},\nu,\theta}$ includes $(p)^{\mathcal{M},\nu,\theta}$, we have $\langle w_1, w_2 \rangle \in (p*)^{\mathcal{M},\nu,\theta}$. And, since $(p*)^{\mathcal{M},\nu,\theta}$ is transitive, we get $\langle \theta(\tau), w_2 \rangle \in (p*)^{\mathcal{M},\nu,\theta}$. Then, $\mathcal{M}, w_2, \nu \Vdash_\theta \varphi$ does not hold. Since w_1 and w_2 were chosen arbitrarily, we get $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \neg\langle p* \rangle \langle p \rangle \varphi$.

□

Theorem 85 (Soundness) If a (subscript-free) formula is provable in the basic tableau system (Figure 13.1), then it is valid.

PROOF. Assume that some formula φ is provable, but non-valid. Since it is non-valid, then $\{\epsilon : \neg\varphi\}$ must be satisfiable. Then, according to the Rule Correctness Lemma 84, some branch of the proof of φ must be satisfiable. But a satisfiable branch can not be closed, since any set containing both $\tau : \varphi$ and $\tau : \neg\varphi$ is unsatisfiable in all structures, and so is any set containing $\tau : t \neq t$. So, the branch must be open, which contradicts the assumption that φ is provable. □

Completeness without the star operator

We prove that every valid formula is provable in the basic tableau system. This is sometimes called *weak completeness* or *deductive completeness*. We will simply call it *complete*.

A system is *strongly complete* or has *consequence completeness* if for any set of formulas, one can check whether a formula is a consequence of that set or not. Our tableau system is not strongly complete, since it is not compact, for the same reason that no strongly complete system exists for propositional dynamic logic. Consider the infinite set

$$\{\neg\langle p*\rangle\varphi, \varphi, \langle p\rangle\varphi, \langle p;p\rangle\varphi, \ldots\}.$$

This set is finitely satisfiable, i.e. every finite subset of it is satisfiable, but it is not itself satisfiable. It is impossible by any finite means to prove that this set is unsatisfiable.

Propositional dynamic logic was proved (weakly) complete by Parikh (1978). Kozen and Parikh (1981) gives a simple proof. In (Massacci 1998), there is a proof of completeness for propositional dynamic logic with the converse operator.

We now prove that our system is complete without the star operator. The system with the star operator is considered in Chapter 18.

17.1 ASSOCIATED STRUCTURE

Our aim is to consider a branch $\mathcal{B}$ of a fair tableau derivation and construct a structure $\mathcal{M}_{\mathcal{B}}$ in which all formulas of the branch are true.

Definition 86 (Fairness) A tableau derivation is *fair* if for every branch of the tableau derivation we have that either

1. the branch is closed, or
2. for every rule of Figure 13.1, if all formulas in the premise of the rule occur in the branch, then all formulas of some conclusion to the rule also occur in the branch.[1]

For any prefixed formula $\epsilon : \varphi$, a fair derivation exists. Furthermore, when a proof exists for a prefixed formula $\epsilon : \varphi$, then any strategy for creating a fair derivation of $\epsilon : \varphi$ will result in a proof of $\epsilon : \varphi$.

We continue by defining the domain of the structure $\mathcal{M}_{\mathcal{B}}$.

Definition 87 (Domain) Let $\mathcal{T}$ be the set of fixed terms of the language with respect to a branch $\mathcal{B}$ of a fair derivation, i.e. let $\mathcal{T}$ be the smallest set satisfying

1. If a is a program variable occurring in $\mathcal{B}$, and τ any prefix constructible from the signature, then $a_\tau \in \mathcal{T}$.
2. If f is a function symbol of arity n occurring in $\mathcal{B}$, and $t_1, \ldots, t_n \in \mathcal{T}$, then $f(t_1, \ldots, t_n) \in \mathcal{T}$.

Let $\doteq$ be the smallest relation $\mathcal{T} \times \mathcal{T}$ closed under reflexivity, transitivity and symmetry satisfying

1. $t_1 \doteq t_2$, if $\epsilon : t_1 = t_2$ occurs in $\mathcal{B}$ (and both t_1 and t_2 are fixed),
2. $f(t_1, \ldots, t_n) \doteq f(t'_1, \ldots, t'_n)$, if $t_1 \doteq t'_1, \ldots, t_n \doteq t'_n$.

For any fixed term $t \in \mathcal{T}$, let $|t|$ be the equivalence class of t with respect to $\doteq$, i.e. $|t| \stackrel{def}{=} \{t' \mid t' \doteq t\}$. Let $\mathcal{D}$ be the set of equivalence classes of $\mathcal{T}$ with respect to $\doteq$, i.e. $\mathcal{D} \stackrel{def}{=} \{|t| \mid t \in \mathcal{T}\}$. If $\mathcal{D}$ is empty, then add one element to it.

Proposition 88 (Branch conditions) The following holds for any open branch $\mathcal{B}$ in any fair tableau derivation.

1. The prefixed formulas $\tau : \varphi$ and $\tau : \neg\varphi$ can not both occur in $\mathcal{B}$ (since $\mathcal{B}$ is open), and all simple branch conditions given in Figure 17.1 holds.
2. For any two prefixes τ and $\tau[a := t]$ in the branch, we have that $\epsilon : a_{\tau[a:=t]} = t@\tau$ occurs in the branch.

[1] This specifically means that every instance of the conclusions to the axiom rules, $(=_1)$ and $(=_2)$, occurs in the branch.

If this occurs in $\mathcal{B}$:	Then this occurs in $\mathcal{B}$:
$\tau : (\varphi \wedge \psi)$	$\tau : \varphi$ and $\tau : \psi$
$\tau : \neg(\varphi \wedge \psi)$	$\tau : \neg\varphi$ or $\tau : \neg\psi$
$\tau : \neg\neg\varphi$	$\tau : \varphi$
$\tau : \neg\langle p; q\rangle\varphi$	$\tau : \neg\langle p\rangle\langle q\rangle\varphi$
$\tau : \langle p; q\rangle\varphi$	$\tau : \langle p\rangle\langle q\rangle\varphi$
$\tau : \neg\langle\psi?\rangle\varphi$	$\tau : \neg\psi$ or $\tau : \neg\varphi$
$\tau : \langle\psi?\rangle\varphi$	$\tau : \psi$ and $\tau : \varphi$
$\tau : \neg\langle p \cup q\rangle\varphi$	$\tau : \neg\langle p\rangle\varphi$ and $\tau : \neg\langle q\rangle\varphi$
$\tau : \langle p \cup q\rangle\varphi$	$\tau : \langle p\rangle\varphi$ or $\tau : \langle q\rangle\varphi$
$\tau : \{\lambda x \leftarrow t.\ \varphi(x)\}$	$\tau : \varphi(t@\tau)$
$\tau : \neg\{\lambda x \leftarrow t.\ \varphi(x)\}$	$\tau : \neg\varphi(t@\tau)$
$\tau : \langle a := t\rangle\varphi$	$\tau[a := t] : \varphi$
$\tau : \neg\langle a := t\rangle\varphi$	$\tau[a := t] : \neg\varphi$

Figure 17.1: Simple branch conditions

3. For any two prefixes τ and $\tau[b := t]$ in the branch and any program variable $a \not\equiv b$ occurring in $\mathcal{B}$, we have $\epsilon : a_{\tau[b:=t]} = a_\tau$ in $\mathcal{B}$.

4. For any fixed terms t'_1 and t'_n, if $\tau : \varphi(t'_1)$ occurs in $\mathcal{B}$ and $t'_1 \doteq t'_n$, then also $\tau : \varphi(t'_n)$ occurs in $\mathcal{B}$.

PROOF. Conditions 1–3 are immediate from the property of fairness (Definition 86). We prove condition 4.

Suppose $\tau : \varphi(t'_1)$ occurs in $\mathcal{B}$.

We have, in fact, that $\doteq$ is the congruence closure of all prefixed atomic equality formulas of the form $\epsilon : t_1 = t_2$ present in the branch. (Note that prefixed equality formulas of the form $\epsilon : a_{\tau[a:=t]} = t@\tau$ and $\epsilon : a_{\tau[b:=t]} = a_\tau$ corresponding to the equalities $a_{\tau[a:=t]} \doteq t@\tau$ and $a_{\tau[b:=t]} \doteq a_\tau$ of Definition 87 are also present in the branch by Branch Conditions 88:2 and 88:3.)

Let $E \stackrel{def}{=} \{t_1 = t_2 \mid \epsilon : t_1 = t_2 \in \mathcal{B} \text{ and } t_1, t_2 \text{ are fixed.}\}$. Then the relation $\doteq$ is equal to the congruence closure of $\{\langle t_1, t_2\rangle \mid t_1 = t_2 \in E\}$. Furthermore, let us define a relation $\stackrel{t_1,t_2}{\longleftrightarrow}$, by the following:

- $t'_1 \stackrel{t_1,t_2}{\longleftrightarrow} t'_2$ iff there exists an equality $t_1 = t_2 \in E$ and a term t such that either
  1. $t' \equiv t(t_1)$ and $t'' \equiv t(t_2)$, or
  2. $t' \equiv t(t_2)$ and $t'' \equiv t(t_1)$.

By Birkhoff's Theorem (Birkhoff 1944), we have that if $t'_1 \doteq t'_n$, then there must exist a sequence of subscripted terms $t'_1, \ldots, t'_n$, such that $t'_1 \overset{t_1,t_2}{\longleftrightarrow} \cdots \overset{t_{n-1},t_n}{\longleftrightarrow} t'_n$ where $t_i = t_{i+1} \in E$, whenever $1 \leq i < n$.

But this means that each $\epsilon : t_i = t_{i+1}$ occurs in the branch. Then, by applying the rules $(=_L)$ and $(=_R)$ in total $n-1$ times, we get a sequence of prefixed formulas $\tau : \varphi(t'_1), \tau : \varphi(t'_2), \ldots$, and finally $\tau : \varphi(t'_n)$. Since $\mathcal{B}$ is a branch of a fair derivation, we have that all these formulas must already be in the branch, so specifically we have $\tau : \varphi(t'_n)$ in the branch. □

Definition 89 (Associated structure) Let $\mathcal{B}$ be a branch of a fair derivation of a closed prefixed formula $\epsilon : \neg\varphi$. Then the *associated structure* is a tuple $\mathcal{M}_\mathcal{B} = \langle \mathcal{W}, \mathcal{D}, \mathcal{I} \rangle$ where

1. $\mathcal{W}$ is the set of all prefixes constructible from the signature.
2. $\mathcal{D}$ is defined as in Definition 87.
3. For every relation symbol P, let:
$$\mathcal{I}(P) \overset{def}{=} \{\langle |t_1@\epsilon|, \ldots, |t_n@\epsilon| \rangle \mid \epsilon : P(t_1, \ldots, t_n) \text{ occurs in } \mathcal{B}\}.$$
4. For every $w \in \mathcal{W}$ and function symbol f of arity $n \geq 0$, let $\mathcal{I}(f)$ be the function mapping sequences $|t_1|, \ldots, |t_n| \in \mathcal{D}$ to $|f(t_1, \ldots, t_n)|$.[2]
5. $\mathcal{I}(\tau, a) \overset{def}{=} |a_\tau|$, for every program variable a occurring in $\mathcal{B}$.

For this to be a structure by Definition 48 we require that for any world τ, program variable a, and $d \in \mathcal{D}$, there exists a world $w' \in \mathcal{W}$, such that $\mathcal{I}(w', a) = d$ and for each $b \not\equiv a$, we have $\mathcal{I}(w', b) = \mathcal{I}(\tau, b)$. We simply add all worlds needed to satisfy this condition to the set of worlds $\mathcal{W}$.

Lemma 90 For any two prefixes τ and $\tau[a := t]$ occurring in $\mathcal{B}$, we have $\mathcal{I}(\tau[a := t]) = \mathcal{I}(\tau)[a \mapsto t]$.

PROOF. We need to show that for any program variable $b \in \mathcal{P}$, we have $\mathcal{I}(\tau[a := t], b) = \mathcal{I}(\tau)[a \mapsto t](b)$. There are two cases.

1. Suppose $b \equiv a$. Then, since $a_{\tau[a:=t]} \doteq t@\tau$, we get $\mathcal{I}(\tau[a := t])(a) = |a_{\tau[a:=t]}| = |t@\tau| = \mathcal{I}(\tau)[a \mapsto t](a)$.
2. Suppose $b \not\equiv a$. Then, since $b_{\tau[a:=t]} \doteq b_\tau$, we get $\mathcal{I}(\tau[a := t])(b) = |b_{\tau[a:=t]}| = |b_\tau| = \mathcal{I}(\tau)[a \mapsto t](b)$.

□

[2] It is easy to verify that $\mathcal{I}(f)$, as defined here, is a function.

17.2 KEY FACT

We show that all formulas occurring in an open branch $\mathcal{B}$ in a fair derivation, are true in the constructed structure $\mathcal{M}_{\mathcal{B}}$ by induction on a well-ordering of the formulas.

Definition 91 Let $<$ be the least transitive relation on formulas satisfying

1. $\varphi < (\varphi \wedge \psi)$, and $\psi < (\varphi \wedge \psi)$,
2. $\varphi(t@\tau) < \{\lambda x \leftarrow t.\ \varphi(x)\}$,
3. $\varphi < \langle p \rangle \varphi$,
4. $\langle p \rangle \langle q \rangle \varphi < \langle p; q \rangle \varphi$,
5. $\langle p \rangle \varphi < \langle p \cup q \rangle \varphi$ and $\langle q \rangle \varphi < \langle p \cup q \rangle \varphi$,
6. $\varphi < \langle \varphi ? \rangle \psi$ and $\psi < \langle \varphi ? \rangle \psi$,
7. $\varphi < \neg \varphi$,
8. $\neg \varphi < \neg(\varphi \wedge \psi)$ and $\neg \psi < \neg(\varphi \wedge \psi)$,
9. $\neg \varphi(t@\tau) < \neg\{\lambda x \leftarrow t.\ \varphi(x)\}$,
10. $\neg \varphi < \neg \langle p \rangle \varphi$,
11. $\neg \langle p \rangle \langle q \rangle \varphi < \neg \langle p; q \rangle \varphi$,
12. $\neg \langle p \rangle \varphi < \neg \langle p \cup q \rangle \varphi$ and $\neg \langle q \rangle \varphi < \neg \langle p \cup q \rangle \varphi$,
13. $\neg \varphi < \neg \langle \varphi ? \rangle \psi$ and $\neg \psi < \neg \langle \varphi ? \rangle \psi$,
14. $\langle p \rangle \cdots \langle p \rangle \varphi < \langle p* \rangle \varphi$.

It is easy to see that $<$ is well-founded. We show that every star-free prefixed formula occurring in a branch $\mathcal{B}$ holds in the corresponding structure $\mathcal{M}_{\mathcal{B}}$.

Fact 92 (Key fact, star-free fragment) Let θ be the identity mapping. Then, for every valuation ν, prefixed star-free formula $\tau : \varphi$, and open branch $\mathcal{B}$ in a fair derivation,

$$\text{if } \tau : \varphi \text{ occurs in } \mathcal{B}, \text{ then } \mathcal{M}_{\mathcal{B}}, \tau, \nu \Vdash_{\theta} \varphi.$$

PROOF. By Lemma 90, we have that θ is a prefix mapping for $\mathcal{M}_{\mathcal{B}}$. The proof continues by well-founded induction on $<$. The Branch Conditions Proposition 88 is used repeatedly in this proof without further comment.

The cases for $\wedge$, $\neg\wedge$, $\neg\neg$, $(p; q)$, $\neg(p; q)$, $(p \cup q)$, $\neg(p \cup q)$, $\varphi?$, and $\neg\varphi?$ are trivial and left to the reader.

P. If $\tau : P(t_1, \ldots, t_n)$ occurs in $\mathcal{B}$, then so does $\epsilon : P(t_1@\tau, \ldots, t_n@\tau)$. Then $\langle |t_1@\tau|, \ldots, |t_n@\tau| \rangle \in \mathcal{I}(P)$, by Definition 89. Since no t_i contains logic variables (Proposition 64), $(\nu \star \mathcal{I}^\theta)(\tau, t_i) = |t_i@\tau|$. Thus $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta P(t_1, \ldots, t_n)$.

$\neg P$. If $\tau : \neg P(t_1, \ldots, t_n)$ occurs in $\mathcal{B}$, then so does $\epsilon : \neg P(t_1@\tau, \ldots, t_n@\tau)$. We need to show that $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta P(t_1, \ldots, t_n)$ does not hold. Suppose it did.

Since no t_i contains logic variables (Proposition 64), $(\nu \star \mathcal{I}^\theta)(\tau, t_i) = |t_i@\tau|$, which gives us that $\langle |t_i@\tau|, \ldots, |t_i@\tau| \rangle \in \mathcal{I}(P)$.

But this could only be, if there exists a formula $\epsilon : P(t'_1, \ldots, t'_n)$ in $\mathcal{B}$, such that $t'_i@\epsilon \in |t_i@\tau|$, whenever $1 \leq i \leq n$. This in turn, implies that $t'_i@\epsilon \doteq t_i@\tau$.

Since $\epsilon : P(t'_1, \ldots, t'_n)$ occurs in $\mathcal{B}$, then so does $\epsilon : P(t'_1@\epsilon, \ldots t'_1@\epsilon)$. Then by n applications of Proposition 88:4, $\epsilon : P(t_1@\tau, \ldots, t_n@\tau)$ also occurs in $\mathcal{B}$.

But, this would mean that the branch is closed, which would be a contradiction. So, $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta P(t_1, \ldots, t_n)$ can not hold, and thus $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \neg P(t_1, \ldots, t_n)$.

$=$. If $\tau : t_1 = t_2$ occurs in $\mathcal{B}$, then so does $\epsilon : t_1@\tau = t_2@\tau$, which implies that $|t_1@\tau| = |t_2@\tau|$. Since $(\nu \star \mathcal{I}^\theta)(\tau, t_i) = |t_i@\tau|$, we get $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta t_1 = t_2$.

$\neg =$. If $\tau : \neg t_1 = t_2$ occurs in $\mathcal{B}$, then so does $\epsilon : \neg t_1@\tau = t_2@\tau$. We need to show that $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta t_1 = t_2$ does not hold. Suppose it did.

Then $(\nu \star \mathcal{I}^\theta)(\tau, t_1) = (\nu \star \mathcal{I}^\theta)(\tau, t_2)$, i.e. $|t_1@\tau| = |t_2@\tau|$, and thus $t_1@\tau \doteq t_2@\tau$. Then by Proposition 88:4, we get that $\epsilon : \neg t_2@\tau = t_2@\tau$ occurs in the branch.

But, this would mean that the branch is closed. Contradiction. So, we must have that $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta t_1 = t_2$ does not hold, which implies $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \neg t_1 = t_2$.

λ. If $\tau : \{\lambda x \leftarrow t.\ \varphi(x)\}$ occurs in $\mathcal{B}$, then so does $\tau : \varphi(t@\tau)$. By induction, $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \varphi(t@\tau)$. By Corollary 83, $\mathcal{M}_\mathcal{B}, \tau, \nu[x \mapsto t^\tau] \Vdash_\theta \varphi(x)$, and thus $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \{\lambda x \leftarrow t.\ \varphi(x)\}$.

$\neg\lambda$. If $\tau : \neg\{\lambda x \leftarrow t.\ \varphi(x)\}$ occurs in $\mathcal{B}$, then so does the prefixed formula $\tau : \neg\varphi(t@\tau)$. By induction, we have $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \neg\varphi(t@\tau)$, so $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \varphi(t@\tau)$ does not hold. Then, by Corollary 83, neither does $\mathcal{M}_\mathcal{B}, \tau, \nu[x \mapsto t^\tau] \Vdash_\theta \varphi(x)$. So, $\mathcal{M}_\mathcal{B}, \tau, \nu[x \mapsto t^\tau] \Vdash_\theta \neg\varphi(x)$ holds, and thus $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \neg\{\lambda x \leftarrow t.\ \varphi(x)\}$.

$a{:=}t$. If $\tau : \langle a := t\rangle\varphi$ occurs in $\mathcal{B}$, then so does $\tau[a := t] : \varphi$. By induction, $\mathcal{M}_\mathcal{B}, \tau[a := t], \nu \Vdash_\theta \varphi$. By Lemma 90, we have $\langle \tau, \tau[a := t] \rangle \in (a := t)^{\mathcal{M}_\mathcal{B}, \nu, \theta}$, so $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \langle a := t\rangle\varphi$.

$\neg a{:=}t$. Suppose $\tau : \neg\langle a := t\rangle\varphi$ occurs in $\mathcal{B}$. We need to show that for every w' such that $\langle \tau, w'\rangle \in (a := t)^{\mathcal{M}_\mathcal{B},\nu,\theta}$, we have $\mathcal{M}_\mathcal{B}, w', \nu \Vdash_\theta \neg\varphi$.

We know that $\tau[a := t] : \neg\varphi$ occurs in $\mathcal{B}$, so by induction, we get $\mathcal{M}_\mathcal{B}, \tau[a := t], \nu \Vdash_\theta \neg\varphi$. By Lemma 90, $\mathcal{I}^\theta(\tau[a := t]) = \mathcal{I}^\theta(\tau)[t \mapsto a]$. Then, Proposition 59 tells us that, for any world w', such that $\mathcal{I}^\theta(w') = \mathcal{I}^\theta(\tau)[t \mapsto a]$, we have $\mathcal{M}_\mathcal{B}, w', \nu \Vdash_\theta \neg\varphi$.

Thus, $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \neg\langle a := t\rangle\varphi$.

□

17.3 COMPLETENESS THEOREM

Theorem 93 (Completeness, star-free fragment) For all valid star-free subscript-free formulas φ, the prefixed formula $\epsilon : \varphi$ is provable in the tableau calculus.

PROOF. We show the contrapositive, i.e. if there is no proof, then we have a structure in which $\epsilon : \varphi$ does not hold. If there is no proof, then any fair derivation of $\epsilon : \varphi$ must have at least one open branch $\mathcal{B}$. We construct a structure $\mathcal{M}_\mathcal{B}$ for the branch $\mathcal{B}$ using Definition 89. The branch $\mathcal{B}$ is satisfied in $\mathcal{M}_\mathcal{B}$ (Key Fact 92). Since $\epsilon : \neg\varphi$ is in the branch, we have that φ does not hold in world ϵ of structure $\mathcal{M}_\mathcal{B}$ under any valuation. Thus φ is not valid. □

We make a remark. In the proof of the Key Fact 92, we only use Proposition 88:4 for prefixed atomic formulas of the form $\epsilon : P(t_1, \ldots, t_n)$ and $\epsilon : \neg t_1 = t_2$. Since the equality rules $(=_L)$ and $(=_R)$ of our system

$$\frac{\epsilon : t_1 = t_2 \quad \tau : \varphi(t_1)}{\tau : \varphi(t_2)}\ (=_L)\ \text{(where } t_1 \text{ and } t_2 \text{ are fixed)}$$

$$\frac{\epsilon : t_2 = t_1 \quad \tau : \varphi(t_1)}{\tau : \varphi(t_2)}\ (=_R)\ \textit{where } t_1 \textit{ and } t_2 \textit{ are fixed.}$$

is not used except in the proof of Proposition 88:4, we could actually restrict these rules by saying that the prefixed formula $\tau : \varphi$ should be of the form $\epsilon : P(t_1, \ldots, t_n)$ or $\epsilon : \neg t_1 = t_2$, without compromising completeness of the tableau calculus.

Completeness with the star operator

To create a complete tableau calculus for the quantifier-free dynamic assignment logic, we add an omega rule. We prove completeness by modifying the proofs of Chapter 17.

$$\frac{\tau : \langle p* \rangle \varphi}{\tau : \varphi \mid \tau : \langle p \rangle \varphi \mid \tau : \langle p \rangle \langle p \rangle \varphi \mid \cdots} \ (\omega)$$

Derivations, proofs, closed branches etc. are all defined in the same way as earlier, with the exception that derivations can now be infinitely branching.

Example 94 (Commuting programs) Any program p commutes with $p*$. We here prove one direction of commutation for the special case when p is the program $a := t$, namely

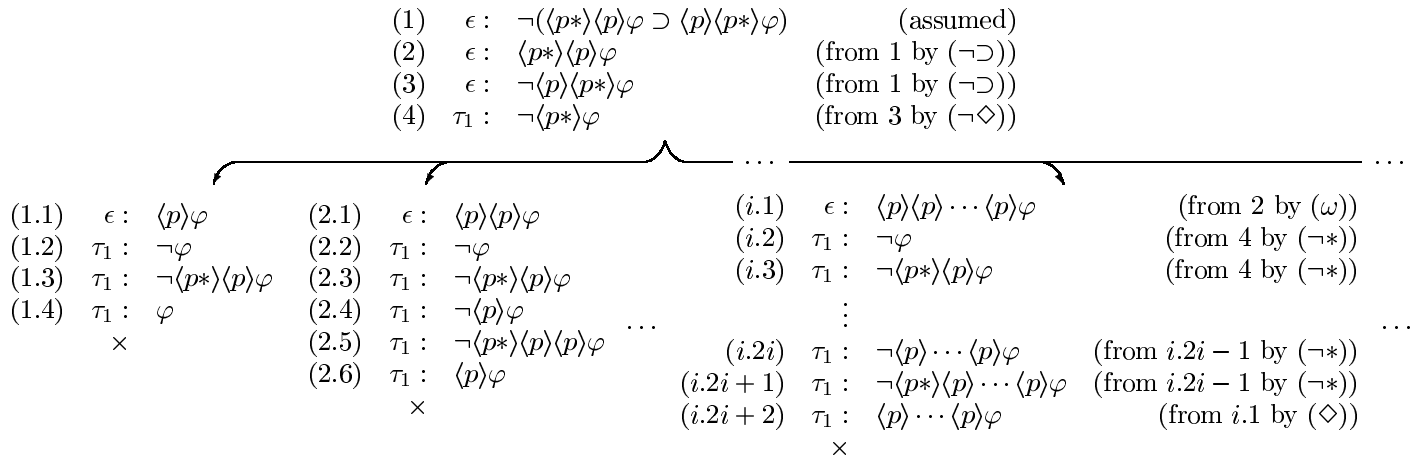
$$\langle (a := t)* \rangle \langle a := t \rangle \varphi \supset \langle a := t \rangle \langle (a := t)* \rangle \varphi.$$

The proof is given in Figure 18.1. Note that each branch is finite.

We establish soundness in the same way as before. First we need to check that the new omega rule is correct.

Lemma 95 (Correctness of omega-rule) The omega rule preserve satisfiability (under the same valuation, in the same structure).

Proof. We assume that a branch $\mathcal{B}$ is satisfiable in some structure $\mathcal{M}$ under valuation ν (and prefix mapping θ). We show that if $\mathcal{B}$ contains the premises of a rule, then adding one set of conclusions of the rule to $\mathcal{B}$ still produces a set of prefixed formulas satisfiable in $\mathcal{M}$ under ν.



where $p \equiv (a := t)$ and $\tau_1 \equiv [a := t]$.

Figure 18.1: An infinite proof

(ω) Suppose $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \langle p* \rangle \varphi$. Then there exists a world w, such that $\langle \theta(\tau), w \rangle \in (p*)^{\mathcal{M},\nu,\theta}$ and $\mathcal{M}, w, \nu \Vdash_\theta \langle p* \rangle \varphi$. Then, since $(p*)^{\mathcal{M},\nu,\theta}$ is the reflexive, transitive closure of $(p)^{\mathcal{M},\nu,\theta}$, we get that $\langle \theta(\tau), w \rangle \in (p)^{\mathcal{M},\nu,\theta} \circ \cdots \circ (p)^{\mathcal{M},\nu,\theta}$. for some number of $(p)^{\mathcal{M},\nu,\theta}$. It is then easy to see that $\mathcal{M}, \theta(\tau), \nu \Vdash_\theta \langle p \rangle \cdots \langle p \rangle \varphi$, with the same number of $\langle p \rangle$.

□

Theorem 96 (Soundness) If a subscript-free formula is provable in the basic tableau calculus (Figure 13.1) with the omega-rule, then it is valid.

PROOF. Similar to the proof of Theorem 85, but using both Lemmas 84 and 95. □

We note that every *open* branch in a fair derivation containing a formula of the form $\tau : \neg \langle p* \rangle \varphi$ is infinite.

Proposition 97 (Branch conditions) The following holds for any open saturated branch $\mathcal{B}$ constructed in the systematic tableau construction.

5. If $\tau : \langle p* \rangle \varphi$ is in $\mathcal{B}$, then so is $\tau : \langle p \rangle \cdots \langle p \rangle \varphi$ for some number of $\langle p \rangle$'s.
6. If $\tau : \neg \langle p* \rangle \varphi$ is in $\mathcal{B}$, then so is $\tau : \neg \langle p \rangle \cdots \langle p \rangle \varphi$, for every number of $\langle p \rangle$'s.

We show that all formulas occurring in a saturated open branch $\mathcal{B}$ are true in the constructed structure $\mathcal{M}_\mathcal{B}$ by induction on the well-ordering $<$ of the formulas (Definition 91).

Fact 98 (Key fact) For every valuation ν, prefix τ, formula φ, and open saturated branch $\mathcal{B}$,

$$\text{if } \tau : \varphi \text{ occurs in } \mathcal{B}, \text{ then } \mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \varphi.$$

PROOF. By well-founded induction on $<$. Most cases are given in the proof of the Key Fact 92.

$p*$. If $\tau : \langle p* \rangle \varphi$ occurs in $\mathcal{B}$, then so does $\tau : \langle p \rangle \cdots \langle p \rangle \varphi$ for some number of $\langle p \rangle$, according to the Branch Conditions (Proposition 97). Then, by induction, $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \langle p \rangle \cdots \langle p \rangle \varphi$, Then there exists a world w in $\mathcal{M}_\mathcal{B}$, such that $\langle \tau, w \rangle \in (p)^{\mathcal{M}_\mathcal{B},\nu,\theta} \circ \cdots \circ (p)^{\mathcal{M}_\mathcal{B},\nu,\theta}$, and $\mathcal{M}_\mathcal{B}, w, \nu \Vdash_\theta \varphi$, But also $\langle \tau, w \rangle \in (p*)^{\mathcal{M}_\mathcal{B},\nu,\theta}$, so $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \langle p* \rangle \varphi$.

$\neg p*$. Suppose $\tau : \neg\langle p*\rangle\varphi$ occurs in $\mathcal{B}$. We want to show $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \neg\langle p*\rangle\varphi$. To show this, we have to show that $\neg\varphi$ holds in every world w_n, such that $\langle \tau, w_n\rangle \in (p*)^{\mathcal{M}_\mathcal{B},\nu,\theta}$. But $\langle \tau, w_n\rangle \in (p*)^{\mathcal{M}_\mathcal{B},\nu,\theta}$ only holds if there exists a sequence of worlds $\tau = w_1, \ldots, w_n$, such that $\langle w_i, w_{i+1}\rangle \in (p)^{\mathcal{M}_\mathcal{B},\nu,\theta}$ whenever $1 \leq i < n$.

By the Branch Conditions (Proposition 97), we know that the prefixed formula $\tau : \neg\langle p\rangle \cdots \langle p\rangle\varphi$ occurs in $\mathcal{B}$ with any number n of $\langle p\rangle$'s. By induction, we get $\mathcal{M}_\mathcal{B}, \tau, \nu \Vdash_\theta \neg\langle p\rangle \cdots \langle p\rangle\varphi$, which implies that φ can not hold in w_n. We leave the details to the reader.

□

Theorem 99 (Completeness) For all valid subscript-free formulas φ, the prefixed formula $\epsilon : \varphi$ is provable in the tableau calculus with the omega rule.

PROOF. Similar to the proof of Theorem 93. □

Conclusion and Future Work

We have presented the semantics and a tableau calculus for the quantifier-free dynamic assignment logic. We have shown that

- The star-free calculus is complete;
- The calculus with the omega rule is complete;
- The validity problem for the logic is undecidable.

Several interesting results emerge if we restrict the language. The system with neither the star operator nor function symbols is decidable, but what is even more interesting is that the system without the star operator, but with function symbols, is also decidable. This was, however, discovered just a couple of days before the deadline of the dissertation, so the proof is not included here.

19.1 COMPLETENESS WITHOUT OMEGA RULE

We leave it as an open problem whether full quantifier-free dynamic assignment logic can be formalized in a complete calculus without omega-rule or similar infinite constructs.

There are complete finitary tableau systems for propositional dynamic logic, see e.g. (Massacci 1998).[1] Tableau approaches for propositional dynamic logic and temporal logics, generally work in the following way. First an

[1] In his thesis, Massacci actually shows that a tableau system for propositional dynamic logic with converse is complete. The converse of a program p, denoted p^-, intuitively means that the program p is run backwards. The formula $[p^-]\varphi$ means that φ must hold *before* the program p is run. Using the converse, preconditions of programs can be expressed.

and-or graph is created (the tableau), and then the corresponding pseudo-model is model-checked deleting nodes with unfulfilled eventualities.

This technique uses history variables introduced by Manna and Pnueli (1995). These were also used for modal μ-calculus in (Stirling 1996). The rules for interaction combines prefixed tableau with the ideas of Stirling and Walker (1991) and Stirling (1996) for model checking fixpoints in the modal μ-calculus.

What is it then, that makes the standard proofs of completeness of a tableau calculus for propositional dynamic logic not generalizable to dynamic assignment logic? The problem is the terms of our logic. We give an example. Consider the prefixed formula $\tau : \langle (a := f(a))* \rangle \varphi$. This prefixed formula, means that either $\tau : \varphi$ or $\tau : \langle a := f(a) \rangle \langle (a := f(a))* \rangle \varphi$ holds. A rule, similar to the one below, is used in several tableau proof systems for propositional dynamic logic.

$$\frac{\tau : \langle p* \rangle \varphi}{\begin{array}{c|c} \tau : \varphi & \tau : \neg\varphi \\ & \tau[p] : \langle p* \rangle \varphi \end{array}}$$

The completeness proof is carried out by identifying states satisfying the same set of formulas. That is, consider all formulas prefixed by τ and all formulas prefixed by $\tau[p]$. If these two sets of formulas are the same, we conclude that the two states named by τ and $\tau[p]$ are actually identical, and the branch is completed, i.e. there is no need to search further in this branch. (This is, of course, a rather simplified description. There is also a need to check that all possible formulas prefixed by τ have been inferred, i.e. that the branch is saturated in some sense.)

The problem is that, in our logic, the interpretation of a non-fixed program variable in a prefixed formula is dependent on the prefix. Our example formula above could generate a branch with prefixed formulas $\tau : \varphi(a), \tau[a := f(a)] : \varphi, \tau[a := f(a)][a := f(a)] : \varphi$, etc. In this branch the interpretation of a in the prefixed formulas varies, making it impossible to identify states named, for example, τ and $\tau[a := f(a)]$, just because they satisfy the same set of formulas. An even simpler example might be the following prefixed formulas: $\tau : a = b$ and $\tau[a := b] : a = b$. Here the second formula is valid, while the first one is not. Even though the two prefixes label the same formulas, they are still different.

19.2 SOME OPEN PROBLEMS

In general, there are two different ways to handle the semantics of atomic programs in dynamic logic.

The first choice is to say that atomic programs are always possible to execute and, they always terminate. This semantically means that some seriality condition has to be satisfied by every structure. In the dissertation, we have chosen this approach, since our atomic programs are assignments and in most programming languages, assignments are always possible.[2]

The other choice is to have no requirements on the atomic programs, and thus drop the seriality condition of structures. To get the corresponding tableau system, we could introduce the following requirements to our tableau calculus.

1. The rule ($\Diamond$) is only allowed, if the prefix $\tau[a := t]$ is already present in the branch.
2. The rule ($\neg\Diamond$) is only allowed, if the prefix $\tau[a := t]$ is new for the branch.

Another open problem is that of having program variables of arity greater that 0, i.e. having assignments to function symbols. This would make it possible to prove properties of arrays, like correctness of sorting algorithms etc.

[2]With some imagination, one can consider programming languages with communication to be an exception of this. Consider two processes $A = \cdots; send(X); \cdots$, $B = \cdots; receive(X); \cdots$, where message X is being sent from A to B. The $receive(X)$ in B can be seen as an implicit assignment $X :=?$ in B. When reaching $receive(X)$, process B suspends until X has been sent by A. This suspension restricts execution of the implicit assignment statement $X :=?$ in B, until X has been sent from A.

Concluding remarks

We have presented the term-modal logics and the quantifier-free dynamic assignment logic. Several directions for future research can be outlined.

The combination of term-modal logic and quantifier-free dynamic assignment logic is an interesting enterprise. Using the scoping operator instead of the quantifiers in term-modal logic, should lead to many interesting decidable fragments of term-modal logic. Also adding some kind of updates, maybe in the form of assignments, to term-modal logic would result in a strong multi-agent logic.

Various people have suggested embedding term-modal logic into first-order dynamic logic. The idea would be to use a dynamic operator $\langle a := t \rangle$ for each term-modal operator $\langle t \rangle$. It is, however, not possible to embed term-modal logic into dynamic assignment logic, since in this logic the modal operator $\langle a := t \rangle$ may not contain any logic variables. One of the main advantages of the term-modal logics is to have logic variables in the modalities, to be able to quantify over modal operators.

Extending dynamic assignment logic with logic variables in assignment programs is interesting, but leads to problems. In the resulting logic, we would be able to express formulas like

$$\neg a = b \supset \{\lambda x \leftarrow b.\ \langle b := a \rangle \langle a := x \rangle \neg a = b\}.$$

The resulting language is very expressive. We could then, for instance, parameterize programs by using logic variables in programs. This can be used to prove meta-properties about programs. An example of this is to prove that some re-use of program variables is harmless, i.e., using the same program variable in two sections of the code has the same meaning as the program using two different variables. This is important in compiler technology, since minimizing the number of program variables maximizes utilization of processor registers, thereby speeding up program execution.

But to define the semantics of this logic, we need to define prefixes and subscripted terms by mutual induction. We would then require that

$$a_{[a:=t@\tau']} = a_{[a:=t@\tau'']}.$$

whenever $(\nu\star\mathcal{I})(\theta(\tau'),t) = (\nu\star\mathcal{I})(\theta(\tau''),t)$, which would complicate things.

Bibliography

Aho, A. V., Sethi, R. and Ullman, J. D.: 1985, *Compilers: Principles, Techniques and Tools*, Addison-Wesley Publishing.

Areces, C., Blackburn, P. and Marx, M.: 2000, The computational complexity of hybrid temporal logics, *Logic Journal of the IGPL* **8**(5), 653–679.

Berman, F. and Paterson, M.: 1981, Propositional dynamic logic is weaker without tests, *Theoretical Computer Science* **16**(3), 321–328.

Birkhoff, G.: 1944, Subdirect unions in universal algebras, *Bull. Amer. Math. Soc.* **50**, 764–768.

Blackburn, P.: 1993, Nominal tense logic, *Notre Dame Journal of Formal Logic* **34**(1), 56–83.

Blackburn, P. and Seligman, J.: 1993, Hybrid languages, *Journal of Logic, Language, and Information* **4**(3), 251–272.

Blackburn, P. and Tzakova, M.: 1998, Hybrid completeness, *Logic Journal of the IGPL* **6**(4), 625–650.

Bressan, A.: 1972, *A General Interpreted Modal Calculus*, Yale University Press.

Chellas, B.: 1980, *Modal Logic, an Introduction*, Cambridge University Press, Cambridge.

Eder, E.: 1985, Properties of substitutions and unifications, *Journal of Symbolic Computations* **1**(1), 31–48.

Fagin, R., Halpern, J., Moses, Y. and Vardi, M.: 1995, *Reasoning about Knowledge*, The MIT Press, Cambridge.

Fischer, M. J. and Ladner, R. E.: 1977, Propositional modal logic of programs, *Ninth Annual ACM Symposium on Theory of Computing*, ACM, New York, N.Y., pp. 286–294.

Fischer, M. J. and Ladner, R. E.: 1979, Propositional dynamic logic of regular programs, *Journal of Computer and System Sciences* **18**(2), 194–211.

Fitting, M.: 1983, *Proof methods for modal and intuitionistic logics*, Vol. 169 of *Synthese Library*, Reidel Publ. Comp.

Fitting, M.: 1988, First-order modal tableaux, *Journal of Automated Reasoning* **4**, 191–213.

Fitting, M.: 1996a, *First Order Logic and Automated Theorem Proving*, Graduate Texts in Computer Science, 2nd edn, Springer Verlag, New York. 1st ed., 1990.

Fitting, M.: 1999, *Types, Tableaus and Gödels God*, Unpublished manuscript.

Fitting, M. C.: 1972, An epsilon-calculus system for first-order S4, *in* W. Hodges (ed.), *Conference in Mathematical Logic, London '70*, pp. 103–110. *Springer Lecture Notes in Mathematics, No. 255*.

Fitting, M. C.: 1973, A modal logic analog of Smullyan's fundamental theorem, *Zeitschrift für mathematische Logik und Gründlagen der Mathematik* **19**, 1–16.

Fitting, M. C.: 1975, A modal logic epsilon-calculus, *Notre Dame Journal of Formal Logic* **16**, 1–16.

Fitting, M. C.: 1991, Modal logic should say more than it does, *in* J.-L. Lassez and G. Plotkin (eds), *Computational Logic, Essays in Honor of Alan Robinson*, MIT Press, Cambridge, MA, pp. 113–135.

Fitting, M. C.: 1996b, A modal Herbrand theorem, *Fundamenta Informaticae* **28**, 101–122.

Fitting, M. and Mendelsohn, R. L.: 1998, *First-Order Modal Logic*, Vol. 277 of *Synthese Library*, Kluwer Academic Publishers, Dordrecht.

Fitting, M., Thalmann, L. and Voronkov, A.: 2000, Term-Modal Logics, *in* R. Dyckhoff (ed.), *Tableaux 2000*, Vol. 1847 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin Heidelberg, pp. 220–236.

Fitting, M., Thalmann, L. and Voronkov, A.: 2001, Term-Modal Logics, *Studia Logica* . To appear.

Frege, G.: 1879, *Begriffschrift, eine der Arithmetischen Nachgebildete Formalsprache des Reinen Denkens*, Halle. Reprinted in (Frege and Angelelli 1966); English translation in (van Heijenoort 1967).

Frege, G.: 1892, Über Sinn und Bedeutung, *Zeitschrift fur Philosophie und philosophische Kritik* **100**, 25–50. "On Sense and Reference" translated in (Frege 1952).

Frege, G.: 1952, *Translations from the Philosophical Writings of Gottlob Frege*, Basil Blackwell, Oxford. P. Geach and M. Black editors.

Frege, G. and Angelelli, I.: 1966, *Begriffschrift und Andere Aufsätze*, Olms, Hildesheim.

Gallier, J. H.: 1986, *Logic for Computer Science: Foundations of Automatic Theorem Proving*, Vol. 5 of *Computer Science and Technology Series*, Harper & Row, New York.

Gargov, G. and Goranko, V.: 1993, Modal logic with names, *Journal of Philosophical Logic* **22**(6), 607–636.

Garson, J.: 1984, Quantification in modal logic, *in* D. Gabbay and F. Guenther (eds), *Handbook in Philosophical Logic*, Vol. II, D. Reidel Publishing Company, chapter II.5, pp. 249–307.

Gentzen, G.: 1934, Untersuchungen über das logische Schließen, *Mathematical Zeitschrift* **39**, 176–210, 405–431. Translated as (Gentzen 1969).

Gentzen, G.: 1969, Investigations into logical deduction, *in* M. Szabo (ed.), *The Collected Papers of Gerhard Gentzen*, North Holland, Amsterdam, pp. 68–131. Originally appeared as (Gentzen 1934).

Grove, A.: 1995, Naming and identity in epistemic logics part II: A first-order logic for naming, *Artificial Intelligence* **74**, 311–350.

Grove, A. and Halpern, J.: 1991, Naming and identity in a multi-agent epistemic logic, *in* J. Allen, R. Fikes and E. Sandewall (eds), *KR'91. Proc. of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, Cambridge, Massachusets, pp. 301–312.

Halpern, J. Y.: 1993, Reasoning about knowledge: a survey circa 1991, *in* A. Kent and J. G. Williams (eds), *Encyclopedia of Computer Science and Technology, Volume 27 (Supplement 12)*, Marcel Dekker, New York.

Harel, D.: 1979, *First-Order Dynamic Logic*, Vol. 68 of *LNCS*, Springer.

Hintikka, J.: 1962, *Knowledge and Belief*, Cornell University Press, Ithaca, New York.

Hoare, C. A. R.: 1969, An axiomatic basis for computer programming, *Communications of the ACM* **12**(10), 576–580.

Hopcroft, J. and Ullman, J. D.: 1979, *Introduction to Automata Theory, Language, and Computation*, Addison–Wesley, Reading, MA.

Hughes, G. and Cresswell, M.: 1968, *An Introduction to Modal Logic*, Methuen, London.

Hughes, G. and Cresswell, M.: 1984, *A companion to modal logic*, Methuen.

Hughes, G. and Cresswell, M.: 1996, *A New Introduction to Modal Logic*, Routledge, London.

Kleene, S. C.: 1952, *Introduction to Metamathematics*, D. van Nostrand, Princeton, New Jersey.

Kozen, D. and Parikh, R.: 1981, An elementary proof of the completeness of PDL, *Theoretical Computer Science* **14**(1), 113–118.

Kozen, D. and Tiuryn, J.: 1989, Logics of programs, *in* J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, North Holland, Amsterdam.

Lenzen, W.: 1978, *Recent work in epistemic logic*, Vol. 30 of *Acta Philosophica Fennica*, North-Holland, Amsterdam.

Manna, Z. and Pnueli, A.: 1995, *Temporal Verification of Reactive Systems: Safety*, Springer-Verlag, New York.

Massacci, F.: 1998, *Efficient Approximate Deduction and an Application to Computer Security*, PhD thesis, Dottorato in Ingegneria Informatica, Università di Roma I "La Sapienza", Dipartimento di Informatica e Sistemistica.

Meyer, J.-J. C. and van der Hoek, W.: 1995, *Epistemic Logic for AI and Computer Science*, number 41 in *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press.

Moller, F. and Birtwistle, G. M.: 1996, *Logics for concurrency: structure versus automata*, Vol. 1043 of *Lecture Notes in Computer Science*, Springer-Verlag Inc., New York, NY, USA.

Parikh, R.: 1978, The completeness of propositional dynamic logic, *in* J. Winkowski (ed.), *Proceedings of the 7th Symposium on Mathematical Foundations of Computer Science*, Vol. 64 of *LNCS*, Springer, Zakopane, Poland, pp. 403–415.

Passay, S. and Tinchev, T.: 1991, An essay in combinatory dynamic logic, *Information and Computation* **93**(2), 263–332.

Passy, S. and Tinchev, T.: 1985, Quantifiers in combinatory PDL: completeness, definability, incompleteness, *in* L. Budach (ed.), *5th International Conference on Fundamentals of Computation Theory*, Vol. 199 of *Lecture notes in computer science*, Springer-Verlag, Cottbus, GDR, pp. 512–519.

Pratt, V. R.: 1976, Semantical Considerations on Floyd-Hoare Logic, *17th Annual Symposium on Foundations of Computer Science*, IEEE, pp. 109–121.

Pratt, V. R.: 1978, A practical decision method for propositional dynamic logic, *ACM Symposium on Theory of Computing (STOC '78)*, ACM Press, New York, pp. 326–337.

Pratt, V. R.: 1980, A near optimal method for reasoning about action, *Journal of Computer and System Sciences* **2**, 231–254.

Ryan, M., Fiadeiro, J. and Maibaum, T.: 1991, Sharing actions and attributes in modal action logic, *in* T. Ito and A. R. Meyer (eds), *Theoretical Aspects of Computer Software*, Vol. 526 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 569–593.

Schütte, K.: 1960, *Beweistheorie (in German)*, Springer Verlag.

Shoenfield, J. R.: 1967, *Mathematical Logic*, Addison-Wesley, Reading.

Smullyan, R.: 1963, A unifying principle in quantification theory, *Proc. Nat. Acad. Sci. U.S.A.*, Vol. 49, pp. 828–832.

Stalnaker, R. and Thomason, R.: 1968, Abstraction in first-order modal logic, *Theoria* **34**, 203–207.

Stirling, C.: 1996, Modal and temporal logics for processes. In Moller and Birtwistle (Moller and Birtwistle 1996).

Stirling, C. and Walker, D.: 1991, Local model checking in the modal mu-calculus, *Theoretical Computer Science* **89**(1), 161–177.

Thomason, R. and Stalnaker, R.: 1968, Modality and reference, *Nous* **2**, 359–372.

Tzakova, M.: 1999, Tableau calculi for hybrid logics, *in* N. V. Murray (ed.), *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX-99)*, Vol. 1617 of *LNAI*, Springer, Berlin, pp. 278–292.

van der Hoek, W. and Meyer, J.-J. C.: 1997, A complete epistemic logic for multiple agents–combining distributed and common knowledge, *in*

M. Bacharach, L. Gerard-Varet, P. Mongin and H. Shin (eds), *Epistemic Logic and the Theory of Games and Decisions*, Kluwer Academic Publishers, Dordrecht, pp. 35–68.

van Heijenoort, J. (ed.): 1967, *From Frege to Gödel, a Source Book in Mathematical Logic*, Harvard University Press, Cambridge.

van Orman Quine, W.: 1974, *Mathematical Logic*, 2 edn, Harvard University Press, Cambridge.

Voronkov, A.: 1996, Proof search in intuitionistic logic based on constraint satisfaction, *in* P. Miglioli, U. Moscato, D. Mundici and M. Ornaghi (eds), *Theorem Proving with Analytic Tableaux and Related Methods. 5th International Workshop, TABLEAUX '96*, Vol. 1071 of *Lecture Notes in Artificial Intelligence*, Terrasini, Palermo Italy, pp. 312–329.

Voronkov, A.: 2001, Proof-search in intuitionistic logic based on constraint satisfaction and related complexity problems, *Logic Journal of IGPL* . To appear.

Wallen, L.: 1990, *Automated Deduction in Nonclassical Logics*, The MIT Press.

Index

Symbols