Modal Higher-order Logic for Agents

J.W. Lloyd

Computer Sciences Laboratory
Research School of Information Sciences and Engineering
Australian National University

Abstract

This paper introduces a modal higher-order logic for representing belief states of agents. The syntax and semantics of the logic and a tableau system for proving theorems are presented and an indication given as to how the logic can be used for building multi-agent systems. The main ideas are illustrated by an extended example concerning a user agent for the Internet.

1 Introduction

This paper is intended primarily as a contribution to the foundations of multi-agent systems. It introduces a multi-modal, typed, higher-order logic for representing belief states of agents. The syntax and semantics of the logic and a tableau system for proving theorems are presented. Also it is shown how the logic can be used for building multi-agent systems.

The general approach towards agent construction adopted in this paper is a familiar one, the main loop in the agent program being as follows: the agent receives some information about the context in which it is operating, it reasons about this information and adopts some intention consistent with its overall objectives, and then it performs an action that should contribute towards achieving the intention. Here I am primarily concerned with the representation of, and reasoning about, information collected by the agent. This information is variously called knowledge or belief. While it is common to make a distinction between knowledge and belief (a piece of knowledge is a *true* belief), the framework of this paper is flexible enough to handle both. For the sake of uniformity, I use the terminology of belief in the following.

What sort of representation language should be used for agent beliefs? Of course, there are many possibilities, but one successful approach is to use modal logic. In many papers about agents, modal propositional logic is employed and, for multi-agents systems, it is natural to use a multi-modal logic with a belief modality for each agent. (See, for example, [FHMV95].) However, propositional logic is clearly not expressive enough for representing the beliefs of many practical agents. Consider, for example, a movie or TV recommender. In the first case, the belief state of the agent should include detailed information about many movies; in the second, the belief state should include the current TV guide. In neither case will propositional logic anywhere near suffice.

The usual next step to address this problem is to move to first-order logic. While this suffices in a narrow technical sense, first-order logic is not really expressive enough either. The main deficiency is that first-order logic does not give a convenient representation of sets,

multisets, and related data types. For this reason, it is desirable to move to higher-order logic which provides a simple representation for sets: a set is a predicate, that is, a set is identified with its characteristic function. Since higher-order functions can have other functions as arguments, it is straightforward to provide functions that process sets. Similar comments apply to multisets and related data types.

It turns out that, in the context of higher-order logic, there is a definition of a class of terms, called basic terms [Llo03], which neatly captures the main data types that are needed in many different kinds of applications. This approach is adopted here and provides a rich and expressive language in which to represent 'individuals' such as a movie or a television program. Furthermore, the use of higher-order logic is strongly motivated by the elegance, convenience, and expressiveness of functional languages, such as Haskell [Je], and functional logic languages, such as Curry [He], that exploit the ability of higher-order functions to have other functions as arguments.

Thus the characteristics of a suitable logic for beliefs have been identified: modalities are needed for adequately capturing subtle aspects of knowledge or belief, multi-modalities are needed for multi-agent systems, and types and higher-order facilities are needed for adequately representing individuals and for reasoning about these individuals. This particular combination of facilities of the logic appears to be new. There are a few treatments of modal higher-order logic (for example, [Gal75] and [Fit02]), but none that I know about has a proper type system that is needed for actual applications. Treatments of multi-modal higher-order logic are even rarer, although the key ideas of multi-modalities are already present in the propositional case so this is not much of a deal.

Much of the literature on modal logic for agents, especially for BDI agents, is about the use of modal logic for the *specification* of agents. An excellent example of this is [Woo00]. However, this is not at all what this paper is concerned about (although the logic introduced here is highly suitable for this purpose). Here I am interested in the use of modal logic inside the agent system itself. Thus the motivation is heavily practical: how can one use such a sophisticated logic directly in the construction of the agent. For this use, matters of primary interest are having an expressive type system and higher-order facilities for knowledge representation, and also the efficiency and decidability of theorem proving in the logic. These matters are addressed below.

In [Fit02], Fitting describes modal higher-order logic as ' ... a rare flower in a remote field. But it is a pretty flower'. It is indeed pretty and it deserves to be much better known. For this reason, the following account gives detailed coverage of the main ideas, including the proof of the soundness result for the tableau system. An extended example about user agents is intended to support the contention that the logic is thoroughly practical.

In the next section, the syntax of the logic is presented. Section 3 gives its semantics. In Section 4, basic terms are introduced. Section 5 contains a detailed presentation of a tableau system for the logic, together with a specialised mechanism for equational reasoning. Section 6 presents an extensive example illustrating the use of the earlier ideas for a user agent. The last section gives some conclusions.

2 Syntax

This section contains the definitions of types and terms in the logic. These definitions are modal versions of the corresponding concepts in [Llo03] (but leaving out polymorphism).

Definition 2.1. An *alphabet* consists of three sets:

1. A set $\mathfrak{T}$ of type constructors.
2. A set $\mathfrak{C}$ of constants.
3. A set $\mathfrak{V}$ of variables.

Each type constructor in $\mathfrak{T}$ has an arity. The set $\mathfrak{T}$ always includes the type constructors *1* and Ω both of arity 0. A *nullary* type constructor is a type constructor of arity 0. *1* is the type of some distinguished singleton set and Ω is the type of the booleans. Each constant in $\mathfrak{C}$ has a signature (see below). The set $\mathfrak{V}$ is denumerable. Variables are typically denoted by $x, y, z, \ldots$. For any particular application, the alphabet is assumed fixed and all definitions are relative to the alphabet.

Types are built up from the set of type constructors, using the symbols $\rightarrow$ and $\times$.

Definition 2.2. A *type* is defined inductively as follows.

1. If T is a type constructor of arity k and $\alpha_1, \ldots, \alpha_k$ are types, then $T\ \alpha_1 \ldots \alpha_k$ is a type. (For $k = 0$, this reduces to a type constructor of arity 0 being a type.)
2. If α and β are types, then $\alpha \rightarrow \beta$ is a type.
3. If $\alpha_1, \ldots, \alpha_n$ are types, then $\alpha_1 \times \cdots \times \alpha_n$ is a type. (For $n = 0$, this reduces to *1* being a type.)

Notation. $\mathfrak{S}$ denotes the set of all types obtained from an alphabet ($\mathfrak{S}$ for 'sort').

The symbol $\rightarrow$ is right associative, so that $\alpha \rightarrow \beta \rightarrow \gamma$ means $\alpha \rightarrow (\beta \rightarrow \gamma)$. Each variable has a type. It is assumed that $\mathfrak{V}$ contains infinitely many variables of each type.

Example 2.1. In practical applications of the logic, a variety of types is needed. For example, declarative programming languages typically admit the following types (which are nullary type constructors): *1*, Ω, *Nat* (the type of natural numbers), *Int* (the type of integers), *Float* (the type of floating-point numbers), *Real* (the type of real numbers), *Char* (the type of characters), and *String* (the type of strings).

Other useful type constructors are those used to define lists, trees, and so on. In the logic, *List* denotes the (unary) list type constructor. Thus, if α is a type, then *List* α is the type of lists whose elements have type α.

Definition 2.3. A *signature* is the declared type for a constant.

The fact that a constant C has signature α is often denoted by $C : \alpha$.

Two different kinds of constants, *data constructors* and *functions*, are distinguished. In a knowledge representation context, data constructors are used to represent individuals. In a programming language context, data constructors are used to construct data values. In contrast, functions are used to compute on data values; functions have definitions while data constructors do not. In the semantics for the logic, the data constructors are used to construct models. As examples, the constants $\top$ (true) and $\bot$ (false) are data constructors, as is each integer, floating-point number, and character. The constant $\sharp$ (cons) used to construct lists is a data constructor.

A *predicate* is a function having signature of the form $\alpha \rightarrow \Omega$, for some α.

The set $\mathfrak{C}$ always includes the following constants.

1. (), having signature *1*.
2. $=_\alpha$, having signature $\alpha \to \alpha \to \Omega$, for each $\alpha \in \mathfrak{S}$.
3. $\top$ and $\bot$, having signature Ω.
4. $\neg$, having signature $\Omega \to \Omega$.
5. $\wedge$, $\vee$, $\longrightarrow$, $\longleftarrow$, and $\longleftrightarrow$, having signature $\Omega \to \Omega \to \Omega$.
6. Σ_α and Π_α, having signature $(\alpha \to \Omega) \to \Omega$, for each $\alpha \in \mathfrak{S}$.

The intended meaning of $=_\alpha$ is identity (that is, $=_\alpha\ x\ y$ is $\top$ iff x and y are identical), the intended meaning of $\top$ is true, the intended meaning of $\bot$ is false, and the intended meanings of the connectives $\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\longleftarrow$, and $\longleftrightarrow$ are as usual. The intended meanings of Σ_α and Π_α are that Σ_α maps a predicate to $\top$ iff the predicate maps at least one element to $\top$ and Π_α maps a predicate to $\top$ iff the predicate maps all elements to $\top$.

Data constructors always have a signature of the form $\sigma_1 \to \cdots \to \sigma_n \to (T\ \alpha_1 \ldots \alpha_k)$, where T is a type constructor of arity k. The *arity* of the data constructor is n. A *nullary* data constructor is a data constructor of arity 0.

The next task is to define the central concept of a term. For the modal part of the definition, I assume the existence of agent i and corresponding necessity modal operator $\Box_i$, for $i = 1, \ldots, m$.

Definition 2.4. A *term*, together with its type, is defined inductively as follows.

1. A variable in $\mathfrak{V}$ of type α is a term of type α.
2. A constant in $\mathfrak{C}$ having signature α is a term of type α.
3. If t is a term of type β and x a variable of type α, then $\lambda x.t$ is a term of type $\alpha \to \beta$.
4. If s is a term of type $\alpha \to \beta$ and t a term of type α, then $(s\ t)$ is a term of type β.
5. If $t_1, \ldots, t_n$ are terms of type $\alpha_1, \ldots, \alpha_n$, respectively, then $(t_1, \ldots, t_n)$ is a term of type $\alpha_1 \times \cdots \times \alpha_n$ (for $n \geq 0$).
6. If t is a term of type Ω and $i \in \{1, \ldots, m\}$, then $\Box_i t$ is a term of type Ω.

Notation. $\mathfrak{L}$ denotes the set of all terms obtained from an alphabet and is called the *language* given by the alphabet.

Definition 2.5. A *non-modal term* is a term not containing any $\Box_i$, for $i = 1, \ldots, m$.

In Definition 2.4, a term of the form $\lambda x.t$ in Part 3 is an *abstraction*, a term of the form $(s\ t)$ in Part 4 is an *application*, a term of the form $(t_1, \ldots, t_n)$ in Part 5 is a *tuple*, and a term of the form $\Box_i t$ in Part 6 is a *box term*.

More precisely, the meaning of Definition 2.4 is as follows. Let $\mathfrak{E}$ denote the set of all expressions obtained from the alphabet, where an expression is a finite sequence of symbols drawn from the set of constants $\mathfrak{C}$, the set of variables $\mathfrak{V}$, $\Box_i$, for $i = 1, \ldots, m$, and '(', ')', 'λ','.', and ','. Then $\mathfrak{L}$ is the intersection of all sets $\mathfrak{X} \subseteq \mathfrak{E} \times \mathfrak{S}$ satisfying the following conditions.

1. If x is a variable in $\mathfrak{V}$ of type α (as a variable), then $(x, \alpha) \in \mathfrak{X}$.
2. If C is a constant in $\mathfrak{C}$ having signature α, then $(C, \alpha) \in \mathfrak{X}$.
3. If $(t, \beta) \in \mathfrak{X}$ and x is a variable of type α, then $(\lambda x.t, \alpha \to \beta) \in \mathfrak{X}$.
4. If $(s, \alpha \to \beta) \in \mathfrak{X}$ and $(t, \alpha) \in \mathfrak{X}$, then $((s\ t), \beta) \in \mathfrak{X}$.
5. If $(t_1, \alpha_1), \ldots, (t_n, \alpha_n) \in \mathfrak{X}$, then $((t_1, \ldots, t_n), \alpha_1 \times \cdots \times \alpha_n) \in \mathfrak{X}$ (for $n \geq 0$).
6. If $(t, \Omega) \in \mathfrak{X}$ and $i \in \{1, \ldots, m\}$, then $(\Box_i t, \Omega) \in \mathfrak{X}$.

There is always at least one set satisfying these conditions, namely $\mathfrak{E} \times \mathfrak{S}$. Thus the intersection is well defined and it satisfies Conditions 1 to 6. Hence $\mathfrak{L}$ is the smallest set satisfying Conditions 1 to 6.

Having made the meaning of Definition 2.4 precise, it will be convenient to follow the usual practice and refer to just the t in some (t, α) as being the term. Thus the type of the term is sometimes suppressed. Since the type of a term is uniquely determined by the term, as I show later, nothing is lost by this practice. This convention is applied immediately in the next proposition.

Proposition 2.1. *Let $t \in \mathfrak{L}$. Then exactly one of the following conditions holds.*

1. *$t \in \mathfrak{V}$.*
2. *$t \in \mathfrak{C}$.*
3. *t has the form $\lambda x.s$, where $s \in \mathfrak{L}$.*
4. *t has the form $(u\ v)$, where $u, v \in \mathfrak{L}$.*
5. *t has the form $(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n \in \mathfrak{L}$.*
6. *t has the form $\Box_i s$, where s is a term and $i \in \{1, \ldots, m\}$.*

Proof. It is clear that at most one of the conditions holds. Now suppose t is a term that is neither a variable, nor a constant, nor has the form $\lambda x.s$, $(u\ v)$, $(t_1, \ldots, t_n)$ or $\Box_i s$. Then $\mathfrak{L} \setminus \{t\}$ satisfies Conditions 1 to 6 in the definition of a term, which contradicts the definition of $\mathfrak{L}$ as being the smallest set satisfying Conditions 1 to 6. Thus t is either a variable, a constant, or has the form $\lambda x.s$, $(u\ v)$, $(t_1, \ldots, t_n)$ or $\Box_i s$.

Suppose that t has the form $\lambda x.s$, but that s is not a term. Then the set of terms $\mathfrak{L} \setminus \{t\}$ satisfies Conditions 1 to 6 in the definition of a term, which contradicts the definition of $\mathfrak{L}$ as being the smallest set satisfying Conditions 1 to 6. Thus s is a term. The arguments for Parts 4, 5 and 6 are similar. □

Proposition 2.2.

1. *An expression of the form $\lambda x.t$ is a term iff t is a term.*
2. *An expression of the form $(s\ t)$ is a term iff s and t are terms, and s has type of the form $\alpha \to \beta$ and t has type α.*
3. *An expression of the form $(t_1, \ldots, t_n)$ is a term iff $t_1, \ldots, t_n$ are terms.*

4. An expression of the form $\Box_i t$ is a term iff t is a term of type Ω.

Proof. 1. If $\lambda x.t$ is a term, then t is a term by Part 3 of Proposition 2.1. Conversely, if t is a term, then $\lambda x.t$ is a term, since $\mathfrak{L}$ satisfies Condition 3 of the definition of a term.

2. Suppose that $(s\ t)$ is a term. Then s and t are terms by Part 4 of Proposition 2.1. If either s does not have type of the form $\alpha \to \beta$ or t does not have type α, then $\mathfrak{L} \setminus \{(s\ t)\}$ satisfies Conditions 1 to 6 in the definition of a term, which contradicts the definition of $\mathfrak{L}$ as being the smallest set satisfying Conditions 1 to 6. Conversely, if s and t are terms, and s has type of the form $\alpha \to \beta$ and t has type α, then $(s\ t)$ is a term, since $\mathfrak{L}$ satisfies Condition 4 of the definition of a term.

3. Suppose that $(t_1, \ldots, t_n)$ is a term. Then $t_1, \ldots, t_n$ are terms by Part 5 of Proposition 2.1. Conversely, if $t_1, \ldots, t_n$ are terms, then $(t_1, \ldots, t_n)$ is a term, since $\mathfrak{L}$ satisfies Condition 5 of the definition of a term.

4. Suppose that $\Box_i t$ is a term. Then t is a term by Part 6 of Proposition 2.1. If t does not have type Ω, then $\mathfrak{L} \setminus \{\Box_i t\}$ satisfies Conditions 1 to 6 in the definition of a term, which contradicts the definition of $\mathfrak{L}$ as being the smallest set satisfying Conditions 1 to 6. Conversely, if t is a term of type Ω, then $\Box_i t$ is a term, since $\mathfrak{L}$ satisfies Condition 6 of the definition of a term. □

Proposition 2.3. *The type of each term is unique.*

Proof. Suppose that some term t has types α and β, where $\alpha \neq \beta$. If t is neither a variable nor a constant, by repeated use of Proposition 2.2, t must contain a term s that is either a variable or a constant with at least two distinct types. If s is a variable, one of these types must be distinct from the type of the variable (as a variable). If t is a constant, one of these types must be distinct from the signature of the constant. In either case, let this type be γ. Consider now the set $\mathfrak{L}$ with all those terms removed that contain s with type γ. This set is strictly smaller than $\mathfrak{L}$ and satisfies Conditions 1 to 6 of the definition of a term, which gives a contradiction. □

Notation. For each $\alpha \in \mathfrak{S}$, $\mathfrak{L}_\alpha$ denotes the set of all terms of type α. Thus $\mathfrak{L} = \bigcup_{\alpha \in \mathfrak{S}} \mathfrak{L}_\alpha$.

Terms of the form $(\Sigma_\alpha\ \lambda x.t)$ are written as $\exists_\alpha x.t$ and terms of the form $(\Pi_\alpha\ \lambda x.t)$ are written as $\forall_\alpha x.t$ (in accord with the intended meaning of Σ_α and Π_α). Furthermore, equality is nearly always written infix.

In a higher-order logic, one may identify sets and predicates – the actual identification is between a set and its characteristic function which is a predicate. Thus, if t is of type Ω, the abstraction $\lambda x.t$ may be written as $\{x \mid t\}$ if it is intended to emphasise that its intended meaning is a set. The notation $\{\}$ means $\{x \mid \bot\}$. The notation $t \in s$ means $(s\ t)$, where s has type $\alpha \to \Omega$, for some α. Furthermore, notwithstanding the fact that sets are mathematically identified with predicates, it is sometimes convenient to maintain an informal distinction between sets (as 'collections of objects') and predicates. For this reason, the notation $\{\alpha\}$ is introduced as a synonym for the type $\alpha \to \Omega$. The term $(s\ t)$ is often written as simply $s\ t$, using juxtaposition to denote application. Juxtaposition is left associative, so that $r\ s\ t$ means $((r\ s)\ t)$.

To prove properties of terms, one can employ the following *principle of induction on the structure of terms.*

Proposition 2.4. *Let $\mathfrak{X}$ be a subset of $\mathfrak{L}$ satisfying the following conditions.*

1. $\mathfrak{V} \subseteq \mathfrak{X}$.
2. $\mathfrak{C} \subseteq \mathfrak{X}$.
3. *If* $t \in \mathfrak{X}$ *and* $x \in \mathfrak{V}$, *then* $\lambda x.t \in \mathfrak{X}$.
4. *If* $s, t \in \mathfrak{X}$, s *has type* $\alpha \to \beta$ *and* t *has type* α, *then* $(s\ t) \in \mathfrak{X}$.
5. *If* $t_1, \ldots, t_n \in \mathfrak{X}$, *then* $(t_1, \ldots, t_n) \in \mathfrak{X}$.
6. *If* $t \in \mathfrak{X}$ *and* t *has type* Ω, *then* $\Box_i t \in \mathfrak{X}$.

Then $\mathfrak{X} = \mathfrak{L}$.

Proof. Clearly $\mathfrak{X}$ satisfies Conditions 1 to 6 of Definition 2.4. Thus, since $\mathfrak{L}$ is the intersection of all such sets, it follows immediately that $\mathfrak{L} \subseteq \mathfrak{X}$. Thus $\mathfrak{X} = \mathfrak{L}$. □

In later inductive proofs about terms, Proposition 2.4 will be the basis of the induction argument, but I will never explicitly state the appropriate set $\mathfrak{X}$ – in all cases, this should be immediately clear.

To prove a property of the set of non-modal terms, Proposition 2.4 can be used, omitting Condition 6 and taking $\mathfrak{X}$ to be a subset of non-modal terms. To see this, suppose one wants to prove the property P of the set of non-modal terms. Consider the set

$$\mathfrak{X} = \{t \in \mathfrak{L} \mid t \text{ is non-modal } \longrightarrow P(t)\}.$$

Then Condition 6 is automatically satisfied, since $\Box_i t$ is modal.

Definition 2.6. The *free variables* of a term are defined inductively as follows.

1. The variable x is free in x.
2. A constant contains no free variables.
3. A variable other than x is free in $\lambda x.t$ if the variable is free in t.
4. A variable is free in $(s\ t)$ if the variable is free in s or t.
5. A variable is free in $(t_1, \ldots, t_n)$ if the variable is free in t_j, for some $j \in \{1, \ldots, n\}$.
6. A variable is free in $\Box_i t$ if the variable is free in t.

Definition 2.7. A term is *closed* if it contains no free variables.

Definition 2.8. A term of type Ω is called a *formula*.

Formulas are usually denoted by φ, ψ, and so on. The possibility operators $\Diamond_i$ are introduced by the definition $\Diamond_i \varphi \equiv \neg\Box_i\neg\varphi$, for $i = 1, \ldots, m$.

Definition 2.9. A *theory* is a collection of formulas.

I will need the concept of an occurrence of a subterm of a term. Let $\mathbb{Z}$ denote the set of integers, $\mathbb{Z}^+$ the set of positive integers, and $(\mathbb{Z}^+)^*$ the set of all strings over the alphabet of positive integers, with ε denoting the empty string.

Definition 2.10. The *occurrence set* of a term t, denoted $\mathcal{O}(t)$, is the set of strings in $(\mathbb{Z}^+)^*$ defined inductively as follows.

1. If t is a variable, then $\mathcal{O}(t) = \{\varepsilon\}$.
2. If t is a constant, then $\mathcal{O}(t) = \{\varepsilon\}$.
3. If t has the form $\lambda x.s$, then $\mathcal{O}(t) = \{\varepsilon\} \cup \{1o \mid o \in \mathcal{O}(s)\}$.
4. If t has the form $(u\ v)$, then $\mathcal{O}(t) = \{\varepsilon\} \cup \{1o \mid o \in \mathcal{O}(u)\} \cup \{2o' \mid o' \in \mathcal{O}(v)\}$.
5. If t has the form $(t_1, \ldots, t_n)$, then $\mathcal{O}(t) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{io_i \mid o_i \in \mathcal{O}(t_i)\}$.
6. If t has the form $\Box_i s$, then $\mathcal{O}(t) = \{\varepsilon\} \cup \{1o \mid o \in \mathcal{O}(s)\}$.

Each $o \in \mathcal{O}(t)$ is called an *occurrence* in t.

More precisely, $\mathcal{O}$ is a function $\mathcal{O} : \mathfrak{L} \to 2^{(\mathbb{Z}^+)^*}$ from the set of terms into the powerset of the set of all strings of positive integers. The existence and uniqueness of $\mathcal{O}$ depends upon that fact that $\mathfrak{L}$ is well founded under the substring relation and hence Proposition A.3 applies: $\mathcal{O}$ is defined directly on the minimal elements (that is, constants and variables) and is uniquely determined by the rules in the definition for abstractions, applications, tuples, and box terms.

Definition 2.11. If t is a term and $o \in \mathcal{O}(t)$, then the *subterm of t at occurrence o*, denoted $t|_o$, is defined inductively on the length of o as follows.

1. If $o = \varepsilon$, then $t|_o = t$.
2. If $o = 1o'$, for some o', and t has the form $\lambda x.s$, then $t|_o = s|_{o'}$.
   If $o = 1o'$, for some o', and t has the form $(u\ v)$, then $t|_o = u|_{o'}$.
   If $o = 2o'$, for some o', and t has the form $(u\ v)$, then $t|_o = v|_{o'}$.
   If $o = io'$, for some o', and t has the form $(t_1, \ldots, t_n)$, then $t|_o = t_i|_{o'}$, for $i = 1, \ldots, n$.
   If $o = 1o'$, for some o', and t has the form $\Box_i s$, then $t|_o = s|_{o'}$.

A *subterm* is a subterm of a term at some occurrence. A subterm is *proper* if it is not at occurrence ε.

Note that a variable appearing immediately after a λ in a term is *not* a subterm since the variable appearing there is not at an occurrence of the term.

An induction argument shows that each subterm is a term.

Definition 2.12. An occurrence of a variable x in a term is *bound* if it occurs within a subterm of the form $\lambda x.t$.

A variable in a term is *bound* if it has a bound occurrence.

An occurrence of a variable in a term is *free* if it is not a bound occurrence.

For a particular occurrence of a subterm $\lambda x.t$ in a term, the occurrence of t is called the *scope* of the λx.

An induction argument shows that a variable is free in a term iff it has a free occurrence in the term.

Next substitutions are introduced.

Definition 2.13. A *substitution* is a finite set of the form $\{x_1/t_1, \ldots, x_n/t_n\}$, where each x_i is a variable, each t_i is a term distinct from x_i and having the same type as x_i, and $x_1, \ldots, x_n$ are distinct. Each element x_i/t_i is called a *binding*.

The empty substitution containing no bindings is denoted by $\{\}$.

Let θ be a term substitution and t a term. Then $\theta|_t$ is the term substitution obtained from θ by restricting θ to just the free variables appearing in t.

Intuitively, the concept of instantiating a term t by a substitution $\theta \equiv \{x_1/t_1, \ldots, x_n/t_n\}$, is simple – each free occurrence of a variable x_i in t is replaced by t_i. But there is a technical complication in that there may be a free variable y, say, in some t_i that is 'captured' in this process because, after instantiation, it occurs in the scope of a subterm of the form $\lambda y.s$ and therefore becomes bound in $t\theta$. Free variable capture spoils the intended meaning of instantiation and hence it is necessary to avoid it. There are two approaches to this: one can disallow instantiation if free variable capture would occur or one can rename bound variables in the term t to avoid free variable capture altogether. The latter approach is adopted here.

Definition 2.14. Let t be a term and $\theta \equiv \{x_1/t_1, \ldots, x_n/t_n\}$ a substitution. The *instance* $t\theta$ of t by θ is defined as follows.

1. If t is a variable x_i, for some $i \in \{1, \ldots, n\}$, then $x_i\theta = t_i$.
   If t is a variable y distinct from all the x_i, then $y\theta = y$.

2. If t is a constant C, then $C\theta = C$.

3. (a) If t is an abstraction $\lambda x.s$ such that, for all $i \in \{1, \ldots, n\}$, x_i is free in $\lambda x.s$ implies x is not free in t_i, then

   $$(\lambda x.s)\theta = \lambda x.(s\theta|_{\lambda x.s}).$$

   (b) If t is an abstraction $\lambda x.s$ such that, for some $i \in \{1, \ldots, n\}$, x_i is free in $\lambda x.s$ and x is free in t_i, then

   $$(\lambda x.s)\theta = \lambda y.(s(\{x/y\} \cup \theta|_{\lambda x.s})).$$

   (Here y is chosen to be the first variable of the same type as x that does not appear in $\lambda x.s$ or $\theta|_{\lambda x.s}$.)

4. If t is an application $(u\ v)$, then $(u\ v)\theta = (u\theta\ v\theta)$.

5. If t is a tuple $(t_1, \ldots, t_n)$, then $(t_1, \ldots, t_n)\theta = (t_1\theta, \ldots, t_n\theta)$.

6. If t has the form $\Box_i s$, then $(\Box_i s)\theta = \Box_i(s\theta)$.

More precisely, what is being defined in Definition 2.14 is a function

$$T : \mathfrak{L} \times \Theta \to \mathfrak{L}$$

such that $T(t, \theta) = t\theta$, for all $t \in \mathfrak{L}$ and $\theta \in \Theta$, where $\mathfrak{L}$ is the set of all terms and Θ is the set of all substitutions. To establish the existence of T, the principle of inductive construction on well-founded sets given by Proposition A.4 is employed. The difficulty that has to be coped with is that, for example, for $(u\theta\ v\theta)$ to be well-defined, it is not only necessary that $u\theta$ and $v\theta$

be terms, but that they have appropriate types. To set up the application of Proposition A.4, consider first the substring relation $\prec$ on $\mathfrak{L}$ and extend this to a relation $\prec_2$ on $\mathfrak{L} \times \Theta$ defined as follows: $(s, \theta) \prec_2 (t, \psi)$ if $s \prec t$. It is easy to see that $\prec_2$ is a well-founded order on $\mathfrak{L} \times \Theta$. The minimal elements of $\mathfrak{L} \times \Theta$ are tuples of the form (t, θ), where t is either a variable or a constant. Second, two partitions are defined: $\{\mathfrak{L}_\alpha \times \Theta\}_{\alpha \in \mathfrak{S}}$ is a partition of $\mathfrak{L} \times \Theta$ and $\{\mathfrak{L}_\alpha\}_{\alpha \in \mathfrak{S}}$ is a partition of $\mathfrak{L}$. Then it needs to be checked that the two conditions concerning consistency in Proposition A.4 are satisfied. First, each minimal element is consistent since each instance of a variable or constant is defined to be a term of the same type. Second, by considering the last four cases in Definition 2.14, it is clear that the rule defining T has the property that if (s, ψ) is consistent, for each $(s, \psi) \prec_2 (t, \theta)$, then (t, θ) is consistent. Thus, by Proposition A.4, the function T exists, is unique, and satisfies the condition $T(\mathfrak{L}_\alpha \times \Theta) \subseteq \mathfrak{L}_\alpha$, for all $\alpha \in \mathfrak{S}$. The latter condition states exactly that if t is a term of type α and θ a substitution, then $t\theta$ is a term of type α.

Proposition 2.5. *Let t be a term of type α and $\theta \equiv \{x_1/t_1, \ldots, x_n/t_n\}$ a substitution. Then a variable x is free in $t\theta$ iff x is free in t and distinct from all x_i or, for some $i \in \{1, \ldots, n\}$, x is free in t_i and x_i is free in t.*

Proof. The proof is by induction on the structure of t.

Let t be a variable x_i, for some $i \in \{1, \ldots, n\}$. Then $x_i\theta = t_i$, and the result follows. Let t be a variable y distinct from all the x_i. Then $y\theta = y$, and the result follows.

Let t be a constant C. Then $C\theta = C$, and the result follows.

Let t be an abstraction $\lambda x.s$ such that, for all $i \in \{1, \ldots, n\}$, x_i is free in $\lambda x.s$ implies x is not free in t_i. Then $(\lambda x.s)\theta = \lambda x.(s\theta|_{\lambda x.s})$. Suppose that $\{x_{i_1}, \ldots, x_{i_k}\}$ is the set of x_i that are free in $\lambda x.s$. Then

z is free in $(\lambda x.s)\theta$
iff $z \neq x$ and z is free in $s\theta|_{\lambda x.s}$ $[(\lambda x.s)\theta = \lambda x.(s\theta|_{\lambda x.s})]$
iff $z \neq x$ and (z is free in s and distinct from $x_{i_1}, \ldots, x_{i_k}$ or,
for some $i \in \{i_1, \ldots, i_k\}$, z is free in t_i and x_i is free in s) [induction hypothesis]
iff z is free in $\lambda x.s$ and distinct from all x_i or,
for some $i \in \{1, \ldots, n\}$, z is free in t_i and x_i is free in $\lambda x.s$
[for all $i \in \{1, \ldots, n\}$, x_i is free in $\lambda x.s$ implies x is not free in t_i].

Let t be an abstraction $\lambda x.s$ such that, for some $i \in \{1, \ldots, n\}$, x_i is free in $\lambda x.s$ and x is free in t_i. Then $(\lambda x.s)\theta = \lambda y.(s(\{x/y\} \cup \theta|_{\lambda x.s}))$. Suppose that $\{x_{i_1}, \ldots, x_{i_k}\}$ is the set of x_i that are free in $\lambda x.s$. Then

z is free in $(\lambda x.s)\theta$
iff $z \neq y$ and z is free in $s(\{x/y\} \cup \theta|_{\lambda x.s})$ $[(\lambda x.s)\theta = \lambda y.(s(\{x/y\} \cup \theta|_{\lambda x.s}))]$
iff $z \neq y$ and (z is free in s and distinct from x and $x_{i_1}, \ldots, x_{i_k}$ or,
for some $i \in \{i_1, \ldots, i_k\}$, z is free in t_i and x_i is free in s or $z = y$ and x is free in s)
[induction hypothesis]
iff z is free in $\lambda x.s$ and distinct from all x_i or,
for some $i \in \{1, \ldots, n\}$, z is free in t_i and x_i is free in $\lambda x.s$.

Let t be an application $(u\ v)$. Thus $(u\ v)\theta = (u\theta\ v\theta)$. Then

x is free in $(u\ v)\theta$
iff x is free in $u\theta$ or x is free in $v\theta$ $[(u\ v)\theta = (u\theta\ v\theta)]$
iff (x is free in u and distinct from all x_i or,
for some $i \in \{1, \ldots, n\}$, x is free in t_i and x_i is free in u) or
(x is free in v and distinct from all x_i or,
for some $i \in \{1, \ldots, n\}$, x is free in t_i and x_i is free in v)
[induction hypothesis]
iff x is free in $(u\ v)$ and distinct from all x_i or,
for some $i \in \{1, \ldots, n\}$, x is free in t_i and x_i is free in $(u\ v)$.

If t is a tuple $(t_1, \ldots, t_n)$, then the proof is similar to the previous part.
Let t have the form $\Box_i s$. Thus $(\Box_i s)\theta = \Box_i(s\theta)$. Then

x is free in $(\Box_i s)\theta$
iff x is free in $s\theta$ $[(\Box_i s)\theta = \Box_i(s\theta)]$
iff x is free in s and distinct from all x_i or,
for some $i \in \{1, \ldots, n\}$, x is free in t_i and x_i is free in s
[induction hypothesis]
iff x is free in $\Box_i s$ and distinct from all x_i or,
for some $i \in \{1, \ldots, n\}$, x is free in t_i and x_i is free in $\Box_i s$.

□

Proposition 2.6. *Let t be a term and θ a substitution. Then $t\theta = t\theta|_t$.*

Proof. If t is a variable or a constant, the result is obvious.

Suppose t is an abstraction $\lambda x.s$ such that, for all $i \in \{1, \ldots, n\}$, x_i is free in $\lambda x.s$ implies x is not free in t_i. Then $(\lambda x.s)\theta = \lambda x.(s\theta|_{\lambda x.s}) = (\lambda x.s)\theta|_{\lambda x.s}$.

Suppose t is an abstraction $\lambda x.s$ such that, for some $i \in \{1, \ldots, n\}$, x_i is free in $\lambda x.s$ and x is free in t_i. Then $(\lambda x.s)\theta = \lambda y.(s(\{x/y\} \cup \theta|_{\lambda x.s})) = (\lambda x.s)\theta|_{\lambda x.s}$. (Note that, for both $(\lambda x.s)\theta$ and $(\lambda x.s)\theta|_{\lambda x.s}$, the same variable y is chosen.)

Suppose t is an application $(u\ v)$. Then

$(u\ v)\theta$
$= (u\theta\ v\theta)$ [Part 4 of Definition 2.14]
$= (u\theta|_u\ \ v\theta|_v)$ [induction hypothesis]
$= (u\theta|_{(u\ v)}\ \ v\theta|_{(u\ v)})$ [induction hypothesis]
$=(u\ v)\theta|_{(u\ v)}$ [Part 4 of Definition 2.14].

If t is a tuple or has the form $\Box_i s$, the proof is similar to the previous case. □

Proposition 2.7. *Let t be a term and $\theta \equiv \{x_1/t_1, \ldots, x_n/t_n\}$ a substitution such that, for $i = 1, \ldots, n$, x_i is not free in t. Then $t\theta = t$.*

Proof. By Proposition 2.6, $t\theta = t\{\}$. An easy induction argument on the structure of t then shows that $t\{\} = t$. □

Proposition 2.8. *Let $\lambda x.s$ be a term and $\theta \equiv \{x_1/t_1, \ldots, x_n/t_n\}$ a substitution such that, for $i = 1, \ldots, n$, x_i is free in $\lambda x.s$. Then the following hold.*

1. *If, for all i, x is not free in t_i, then*

$$(\lambda x.s)\theta = \lambda x.(s\theta).$$

2. *If, for some i, x is free in t_i, then*

$$(\lambda y.s)\theta = \lambda y.(s(\{x/y\} \cup \theta)),$$

where y is the first variable of the same type as x that does not appear in $\lambda x.s$ or θ.

Proof. The result follows easily from Definition 2.14, since $\theta|_{\lambda x.s} = \theta$. □

Propositions 2.6 and 2.8 can simplify proofs of results involving substitutions. Typically, one can show that without loss of generality a substitution can be restricted to the free variables appearing in some term, by using Proposition 2.6. Then Proposition 2.8 shows that for application of the substitution to an abstraction only the simpler cases in that proposition need be considered. For an illustration of this approach, see the proof of Proposition 3.4 below.

For the equational reasoning introduced in Section 5, it will be necessary to replace subterms of terms by other terms.

Definition 2.15. Let t be a term, s a subterm of t at occurrence o, and r a term. Then the expression obtained by *replacing s in t by r*, denoted $t[s/r]_o$, is defined by induction on the length of o as follows.

If the length of o is 0, then $t[s/r]_o = r$.

For the inductive step, suppose the length of o is $n + 1$ ($n \geq 0$). There are several cases to consider.

If $o = 1o'$, for some o', and t has the form $\lambda x.w$, then $(\lambda x.w)[s/r]_o = \lambda x.(w[s/r]_{o'})$.

If $o = 1o'$, for some o', and t has the form $(u\ v)$, then $(u\ v)[s/r]_o = (u[s/r]_{o'}\ v)$.

If $o = 2o'$, for some o', and t has the form $(u\ v)$, then $(u\ v)[s/r]_o = (u\ v[s/r]_{o'})$.

If $o = io'$, for some $i \in \{1, \ldots, n\}$ and o', and t has the form $(t_1, \ldots, t_n)$, then $(t_1, \ldots, t_n)[s/r]_o = (t_1, \ldots, t_i[s/r]_{o'}, \ldots, t_n)$.

If $o = 1o'$, for some o', and t has the form $\Box_i q$, then $(\Box_i q)[s/r]_o = \Box_i(q[s/r]_{o'})$.

Proposition 2.9. *Let t be a term, s a subterm of t at occurrence o, and r a term. Suppose that s and r have the same type. Then $t[s/r]_o$ is a term of the same type as t.*

Proof. The proof is by induction on the length n of o.

Suppose first that $n = 0$. Thus t is s and $t[s/r]_o$ is r, so that $t[s/r]_o$ is a term of the same type as t.

Suppose next that the result holds for occurrences of length n and o has length $n + 1$. Thus t has the form $\lambda x.w$, $(u\ v)$, $(t_1, \ldots, t_n)$ or $\Box_i s$.

Consider the case when t has the form $\lambda x.w$ and $o = 1o'$, for some o'. Then $(\lambda x.w)[s/r]_o = \lambda x.(w[s/r]_{o'})$. By the induction hypothesis, $w[s/r]_{o'}$ is a term of the same type as w. Hence $(\lambda x.w)[s/r]_o$ is a term of the same type as $\lambda x.w$.

The other cases are similar. □

Replacing a subterm s in a term t by a term r is quite different to applying a substitution: subterms are replaced, not just free variables; and it is common, even desirable, for free variables in the replacement term r to be captured in $t[s/r]_o$ after replacement. Usually, all free variables in r are free variables in s, so that there is no *new* free variable that could be captured, but there is no reason to insist on this here.

3 Semantics

This section gives the model theory for the logic.

Definition 3.1. A *domain set* for an alphabet is a collection $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$ of non-empty sets satisfying the following conditions.

1. If α has the form $T\ \alpha_1 \dots \alpha_k$, then $\mathcal{D}_\alpha = \{C\ d_1 \dots d_n \mid C$ is a data constructor having signature $\sigma_1 \to \dots \to \sigma_n \to (T\ \alpha_1 \dots \alpha_k)$, and $d_i \in \mathcal{D}_{\sigma_i}$, for $i = 1, \dots, n\}$.
2. If α has the form $\beta \to \gamma$, then $\mathcal{D}_\alpha$ is the collection of mappings from $\mathcal{D}_\beta$ to $\mathcal{D}_\gamma$.
3. If α has the form $\alpha_1 \times \dots \times \alpha_n$, for some $n \geq 0$, then $\mathcal{D}_\alpha$ is the cartesian product $\mathcal{D}_{\alpha_1} \times \dots \times \mathcal{D}_{\alpha_n}$. In particular, $\mathcal{D}_1$ is the distinguished singleton set.

For each $\alpha \in \mathfrak{S}$, $\mathcal{D}_\alpha$ is called a *domain*.

Definition 3.2. A *frame* is a pair $\langle W, \{R_i\}_{i=1}^m\rangle$ consisting of a non-empty set W and a set of binary relations $\{R_i\}_{i=1}^m$ on W.

Each element of W is called a *world* and each relation R_i is called an *accessibility* relation.

Definition 3.3. An *augmented frame* for an alphabet is a triple $\langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}\rangle$, where $\langle W, \{R_i\}_{i=1}^m\rangle$ is a frame and $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$ is a domain set for the alphabet.

Definition 3.4. An *interpretation* for an alphabet is a quadruple $\langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}, V\rangle$, where $\langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}\rangle$ is an augmented frame for the alphabet and V is a mapping that maps each pair consisting of a constant having signature α and a world to an element of $\mathcal{D}_\alpha$ (called the *denotation* of the constant in the world) such that the following conditions are satisfied.

1. If C is a data constructor having signature $\sigma_1 \to \dots \to \sigma_n \to (T\ \alpha_1 \dots \alpha_k)$, then $V(C, w)$ is the element of $\mathcal{D}_{\sigma_1\to\dots\to\sigma_n\to(T\ \alpha_1\dots\alpha_k)}$ defined by $V(C, w)\ d_1 \dots d_n = C\ d_1 \dots d_n$, where $d_i \in \mathcal{D}_{\sigma_i}$ $(i = 1, \dots, n)$.
2. For $=_\alpha$ with signature $\alpha \to \alpha \to \Omega$, $V(=_\alpha, w)$ is the mapping from $\mathcal{D}_\alpha$ into $\mathcal{D}_{\alpha\to\Omega}$ defined by
$$V(=_\alpha, w)\ x\ y = \begin{cases} \top & \text{if } x = y \\ \bot & \text{otherwise.} \end{cases}$$
3. $V(\neg, w)$ is the mapping from $\mathcal{D}_\Omega$ into $\mathcal{D}_\Omega$ given by the following table.

x	$V(\neg, w)\ x$
$\top$	$\bot$
$\bot$	$\top$

4. $V(\wedge, w)$, $V(\vee, w)$, $V(\longrightarrow, w)$, and $V(\longleftrightarrow, w)$ are the mappings from $\mathcal{D}_\Omega$ into $\mathcal{D}_{\Omega\to\Omega}$ given by the following table.

x	y	$V(\wedge,w)\ x\ y$	$V(\vee,w)\ x\ y$	$V(\longrightarrow,w)\ x\ y$	$V(\longleftrightarrow,w)\ x\ y$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$

5. For Σ_α with signature $(\alpha \to \Omega) \to \Omega$, $V(\Sigma_\alpha, w)$ is the mapping from $\mathcal{D}_{\alpha\to\Omega}$ to $\mathcal{D}_\Omega$ which maps an element f of $\mathcal{D}_{\alpha\to\Omega}$ to $\top$ if f maps at least one element of $\mathcal{D}_\alpha$ to $\top$; otherwise, it maps f to $\bot$.

6. For Π_α with signature $(\alpha \to \Omega) \to \Omega$, $V(\Pi_\alpha, w)$ is the mapping from $\mathcal{D}_{\alpha\to\Omega}$ to $\mathcal{D}_\Omega$ which maps an element f of $\mathcal{D}_{\alpha\to\Omega}$ to $\top$ if f maps every element of $\mathcal{D}_\alpha$ to $\top$; otherwise, it maps f to $\bot$.

In effect, the connectives and quantifiers are given their usual fixed meanings in each world. Similarly, the data constructors have their usual fixed free interpretations in each world. For other constants, their meaning may change from world to world. Note that I am using here the constant domain semantics, in contrast to the varying domain semantics [FM98].

Definition 3.5. A *variable assignment* with respect to a domain set $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$ is a mapping that maps each variable of type α to an element of $\mathcal{D}_\alpha$.

Definition 3.6. Let t be a term, $I \equiv \langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}},\ V\rangle$ an interpretation, ν a variable assignment with respect to $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, and w a world in W. Then the *denotation* $\mathcal{V}(t, I, w, \nu)$ of t with respect to I, w and ν is defined inductively as follows.

1. $\mathcal{V}(x, I, w, \nu) = \nu(x)$, where x is a variable.
2. $\mathcal{V}(C, I, w, \nu) = V(C, w)$, where C is a constant.
3. $\mathcal{V}(\lambda x.s, I, w, \nu) =$ the function whose value for each $d \in \mathcal{D}_\alpha$ is $\mathcal{V}(s, I, w, \nu')$, where x has type α and ν' is ν except $\nu'(x) = d$.
4. $\mathcal{V}((s\ r), I, w, \nu) = \mathcal{V}(s, I, w, \nu)(\mathcal{V}(r, I, w, \nu))$.
5. $\mathcal{V}((t_1, \ldots, t_n), I, w, \nu) = (\mathcal{V}(t_1, I, w, \nu), \ldots, \mathcal{V}(t_n, I, w, \nu))$.
6. $\mathcal{V}(\Box_i\varphi, I, w, \nu) = \begin{cases} \top & \text{if, for each } w' \text{ such that } w\,R_i\,w',\ \mathcal{V}(\varphi, I, w', \nu) = \top \\ \bot & \text{otherwise.} \end{cases}$

Definition 3.6 is an inductive construction employing Proposition A.4. To set this up, let the interpretation $I \equiv \langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}},\ V\rangle$ be fixed throughout. Then Definition 3.6 defines a function

$$T : \mathfrak{L} \times W \times \mathcal{A} \to \mathfrak{D}$$

such that $T(t, w, \nu) = \mathcal{V}(t, I, w, \nu)$, for all $t \in \mathfrak{L}$, $w \in W$, and $\nu \in \mathcal{A}$, where $\mathcal{A}$ is the set of all variable assignments and $\mathfrak{D}$ is the (disjoint) union of the $\mathfrak{D}_\alpha$, for all $\alpha \in \mathfrak{S}$. Consider now the

substring relation $\prec$ on $\mathfrak{L}$ and extend this to a relation $\prec_2$ on $\mathfrak{L} \times W \times \mathcal{A}$ defined as follows: $(s, v, \mu) \prec_2 (t, w, \nu)$ if $s \prec t$. It is easy to see that $\prec_2$ is a well-founded order on $\mathfrak{L} \times W \times \mathcal{A}$. The minimal elements of $\mathfrak{L} \times W \times \mathcal{A}$ are triples of the form (t, w, ν), where t is either a variable or a constant. Also needed are two partitions: $\{\mathfrak{L}_\alpha \times W \times \mathcal{A}\}_{\alpha \in \mathfrak{S}}$ of $\mathfrak{L} \times W \times \mathcal{A}$ and $\{\mathfrak{D}_\alpha\}_{\alpha \in \mathfrak{S}}$ of $\mathfrak{D}$. Then it needs to be checked that the two conditions concerning consistency in Proposition A.4 are satisfied. First, each minimal element is consistent since each denotation of a variable or constant of type α is defined to be an element of $\mathfrak{D}_\alpha$. Second, by considering the last four cases in Definition 3.6, it is clear that the rule defining T has the property that if (s, v, μ) is consistent, for each $(s, v, \mu) \prec_2 (t, w, \nu)$, then (t, w, ν) is consistent. Thus, by Proposition A.4, the function T exists, is unique, and satisfies the condition $T(\mathfrak{L}_\alpha \times W \times \mathcal{A}) \subseteq \mathfrak{D}_\alpha$, for all $\alpha \in \mathfrak{S}$. The latter condition states exactly that if t is a term of type α, $I \equiv \langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha \in \mathfrak{S}}, V\rangle$ an interpretation, ν a variable assignment with respect to $\{\mathcal{D}_\alpha\}_{\alpha \in \mathfrak{S}}$, and w a world in W, then $\mathcal{V}(t, I, w, \nu) \in \mathcal{D}_\alpha$.

Definition 3.7. The variable assignments ν and ν' are said to be *x-variants* if they agree on all variables except possibly the variable x.

Proposition 3.1. *Let $I \equiv \langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha \in \mathfrak{S}}, V\rangle$ be an interpretation, ν a variable assignment with respect to $\{\mathcal{D}_\alpha\}_{\alpha \in \mathfrak{S}}$, and $w \in W$. Then the following hold.*

1. $\mathcal{V}(\varphi \wedge \psi, I, w, \nu) = \top$ *iff* $\mathcal{V}(\varphi, I, w, \nu) = \top$ *and* $\mathcal{V}(\psi, I, w, \nu) = \top$.
2. $\mathcal{V}(\varphi \vee \psi, I, w, \nu) = \top$ *iff* $\mathcal{V}(\varphi, I, w, \nu) = \top$ *or* $\mathcal{V}(\psi, I, w, \nu) = \top$.
3. $\mathcal{V}(\neg\varphi, I, w, \nu) = \top$ *iff* $\mathcal{V}(\varphi, I, w, \nu) = \bot$.
4. $\mathcal{V}(\varphi, I, w, \nu) = \top$ *iff* $\mathcal{V}(\neg\varphi, I, w, \nu) = \bot$.
5. $\mathcal{V}(\forall_\alpha x.\varphi, I, w, \nu) = \top$ *iff, for every x-variant* ν', $\mathcal{V}(\varphi, I, w, \nu') = \top$.
6. $\mathcal{V}(\exists_\alpha x.\varphi, I, w, \nu) = \top$ *iff, for some x-variant* ν', $\mathcal{V}(\varphi, I, w, \nu') = \top$.
7. $\mathcal{V}(\Diamond_i\varphi, I, w, \nu) = \top$ *iff, for some* w' *such that* $w\, R_i\, w'$, $\mathcal{V}(\varphi, I, w', \nu) = \top$.

Proof. 1.

$$
\begin{aligned}
&\mathcal{V}(\varphi \wedge \psi, I, w, \nu) = \top \\
\text{iff }\ &\mathcal{V}((\wedge\, \varphi), I, w, \nu)(\mathcal{V}(\psi, I, w, \nu)) = \top \\
\text{iff }\ &(\mathcal{V}(\wedge, I, w, \nu)(\mathcal{V}(\varphi, I, w, \nu)))(\mathcal{V}(\psi, I, w, \nu)) = \top \\
\text{iff }\ &(V(\wedge, w)(\mathcal{V}(\varphi, I, w, \nu)))(\mathcal{V}(\psi, I, w, \nu)) = \top \\
\text{iff }\ &\mathcal{V}(\varphi, I, w, \nu) = \top \text{ and } \mathcal{V}(\psi, I, w, \nu) = \top.
\end{aligned}
$$

2. This is similar to Part 1.
3.

$$
\begin{aligned}
&\mathcal{V}(\neg\varphi, I, w, \nu) = \top \\
\text{iff }\ &\mathcal{V}(\neg, I, w, \nu)(\mathcal{V}(\varphi, I, w, \nu)) = \top \\
\text{iff }\ &V(\neg, w)(\mathcal{V}(\varphi, I, w, \nu)) = \top \\
\text{iff }\ &\mathcal{V}(\varphi, I, w, \nu) = \bot.
\end{aligned}
$$

4. This is similar to Part 3.
5. Let $T : \mathfrak{D}_\alpha \to \mathfrak{D}_\Omega$ be defined by $T(x) = \top$, for all x. Then

$$\begin{aligned}
&\mathcal{V}(\forall_\alpha x.\varphi, I, w, \nu) = \top \\
\text{iff } &\mathcal{V}(\Pi_\alpha, I, w, \nu)(\mathcal{V}(\lambda x.\varphi, I, w, \nu)) = \top \\
\text{iff } &\mathcal{V}(\lambda x.\varphi, I, w, \nu) = T \\
\text{iff } &\text{for every } x\text{-variant } \nu', \mathcal{V}(\varphi, I, w, \nu') = \top.
\end{aligned}$$

6. This is similar to Part 5.
7.

$$\begin{aligned}
&\mathcal{V}(\Diamond_i \varphi, I, w, \nu) \;=\; \top \\
\text{iff } &\mathcal{V}(\neg\Box_i\neg\varphi, I, w, \nu) \;=\; \top \\
\text{iff } &\mathcal{V}(\Box_i\neg\varphi, I, w, \nu) \;=\; \bot \\
\text{iff } &\text{for some } w' \text{ such that } w\, R_i\, w',\ \mathcal{V}(\neg\varphi, I, w', \nu) \;=\; \bot \\
\text{iff } &\text{for some } w' \text{ such that } w\, R_i\, w',\ \mathcal{V}(\varphi, I, w', \nu) \;=\; \top.
\end{aligned}$$

□

Proposition 3.2. *Let I be an interpretation, w a world in I, ν a variable assignment, and φ a formula. Then the following hold.*

1. $\mathcal{V}(\varphi =_\Omega \top, I, w, \nu) = \mathcal{V}(\varphi, I, w, \nu)$.
2. $\mathcal{V}(\varphi =_\Omega \bot, I, w, \nu) = \mathcal{V}(\neg\varphi, I, w, \nu)$.

Proof. 1.

$$\begin{aligned}
&\mathcal{V}(\varphi =_\Omega \top, I, w, \nu) \\
&= \mathcal{V}((=_\Omega \varphi), I, w, \nu)\ (\mathcal{V}(\top, I, w, \nu)) \\
&= (\mathcal{V}(=_\Omega, I, w, \nu)\ (\mathcal{V}(\varphi, I, w, \nu)))\ (\mathcal{V}(\top, I, w, \nu)) \\
&= (V(=_\Omega, w)\ (\mathcal{V}(\varphi, I, w, \nu)))\ (\mathcal{V}(\top, I, w, \nu)) \\
&= (V(=_\Omega, w)\ (\mathcal{V}(\varphi, I, w, \nu)))\ (\top) \\
&= \mathcal{V}(\varphi, I, w, \nu).
\end{aligned}$$

2.

$$\begin{aligned}
&\mathcal{V}(\varphi =_\Omega \bot, I, w, \nu) \\
&= \mathcal{V}((=_\Omega \varphi), I, w, \nu)\ (\mathcal{V}(\bot, I, w, \nu)) \\
&= (\mathcal{V}(=_\Omega, I, w, \nu)\ (\mathcal{V}(\varphi, I, w, \nu)))\ (\mathcal{V}(\bot, I, w, \nu)) \\
&= (V(=_\Omega, w)\ (\mathcal{V}(\varphi, I, w, \nu)))\ (\mathcal{V}(\bot, I, w, \nu)) \\
&= (V(=_\Omega, w)\ (\mathcal{V}(\varphi, I, w, \nu)))\ (\bot) \\
&= \mathcal{V}(\neg\varphi, I, w, \nu).
\end{aligned}$$

□

The next result shows that the denotation of a *closed* term does not depend on the variable assignment.

Proposition 3.3. *Let $I \equiv \langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}, V\rangle$ be an interpretation, $w \in W$, and t a term. If ν_1 and ν_2 are variable assignments with respect to $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$ that agree on the free variables of t, then $\mathcal{V}(t, I, w, \nu_1) = \mathcal{V}(t, I, w, \nu_2)$.*

Proof. The proof is by induction on the structure of t.

If t is a variable x, then $\mathcal{V}(x, I, w, \nu_1) = \nu_1(x) = \nu_2(x) = \mathcal{V}(x, I, w, \nu_2)$.

If t is a constant C, then $\mathcal{V}(C, I, w, \nu_1) = V(C, w) = \mathcal{V}(C, I, w, \nu_2)$.

Let t be an abstraction $\lambda x.s$. Then

$$
\begin{aligned}
&\mathcal{V}(\lambda x.s, I, w, \nu_1)\\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(s, I, w, \nu_1'),\\
&\quad \text{where } x \text{ has type } \alpha \text{ and } \nu_1' \text{ is } \nu_1 \text{ except } \nu_1'(x) = d\\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(s, I, w, \nu_2'),\\
&\quad \text{where } x \text{ has type } \alpha \text{ and } \nu_2' \text{ is } \nu_2 \text{ except } \nu_2'(x) = d\\
&\quad \text{[induction hypothesis, since } \nu_1' \text{ and } \nu_2' \text{ agree on the free variables of } s\text{]}\\
&= \mathcal{V}(\lambda x.s, I, w, \nu_2).
\end{aligned}
$$

Let t be an application $(u\ v)$. Then

$$
\begin{aligned}
&\mathcal{V}((u\ v), I, w, \nu_1)\\
&= \mathcal{V}(u, I, w, \nu_1)(\mathcal{V}(u, I, w, \nu_1))\\
&= \mathcal{V}(u, I, w, \nu_2)(\mathcal{V}(u, I, w, \nu_2)) \quad \text{[induction hypothesis]}\\
&= \mathcal{V}((u\ v), I, w, \nu_2).
\end{aligned}
$$

Let t be a tuple $(t_1, \ldots, t_n)$. Then

$$
\begin{aligned}
&\mathcal{V}((t_1, \ldots, t_n), I, w, \nu_1)\\
&= (\mathcal{V}(t_1, I, w, \nu_1), \ldots, \mathcal{V}(t_n, I, w, \nu_1))\\
&= (\mathcal{V}(t_1, I, w, \nu_2), \ldots, \mathcal{V}(t_n, I, w, \nu_2)) \quad \text{[induction hypothesis]}\\
&= \mathcal{V}((t_1, \ldots, t_n), I, w, \nu_2).
\end{aligned}
$$

Let t have the form $\Box_i\varphi$. By the induction hypothesis, $\mathcal{V}(\varphi, I, w', \nu_1) = \mathcal{V}(\varphi, I, w', \nu_2)$, for each w' such that $w\ R_i\ w'$, since ν_1 and ν_2 agree on the free variables in φ. It follows that $\mathcal{V}(\Box_i\varphi, I, w, \nu_1) = \mathcal{V}(\Box_i\varphi, I, w, \nu_2)$. □

The next few results are concerned with knowing the semantics of a term after a substitution is applied or a subterm replaced. Note carefully that these results only apply to non-modal terms. This will have consequences for the generality of the tableau theorem prover introduced below.

Proposition 3.4. *Let t be a non-modal term, $\theta \equiv \{x_1/t_1, \ldots, x_n/t_n\}$ a substitution, I an interpretation, w a world in I, and ν a variable assignment. Then $\mathcal{V}(t\theta, I, w, \nu) = \mathcal{V}(t, I, w, \nu')$, where $\nu'(x_i) = \mathcal{V}(t_i, I, w, \nu)$, for $i = 1, \ldots, n$, and $\nu'(y) = \nu(y)$, for $y \notin \{x_1, \ldots, x_n\}$.*

Proof. It can be supposed without loss of generality that x_i is free in t, for $i = 1, \ldots, n$. For, suppose $x_{i_1}, \ldots, x_{i_k}$ are the variables amongst the x_i that are free in t, so that $\theta|_t = \{x_{i_1}/t_{i_1}, \ldots, x_{i_k}/t_{i_k}\}$. Then

$$\begin{aligned}
&\mathcal{V}(t\theta, I, w, \nu) \\
&= \mathcal{V}(t\theta|_t, I, w, \nu) \quad \text{[Proposition 2.6]} \\
&= \mathcal{V}(t, I, w, \nu^*), \text{ where } \nu^*(x_{i_j}) = \mathcal{V}(t_{i_j}, I, w, \nu), \text{ for } j = 1, \ldots, k, \\
&\quad \text{and } \nu^*(y) = \nu(y), \text{ for } y \notin \{x_{i_1}, \ldots, x_{i_k}\} \quad \text{[assumption]} \\
&= \mathcal{V}(t, I, w, \nu'), \text{ where } \nu'(x_i) = \mathcal{V}(t_i, I, w, \nu), \text{ for } i = 1, \ldots, n, \\
&\quad \text{and } \nu'(y) = \nu(y), \text{ for } y \notin \{x_1, \ldots, x_n\} \\
&\quad \text{[Proposition 3.3, since } \nu^* \text{ and } \nu' \text{ agree on the free variables in } t\text{]}.
\end{aligned}$$

The proof is by induction on the structure of t.

Suppose t is a variable x. If x is distinct from all the x_i, then $\mathcal{V}(x\theta, I, w, \nu) = \mathcal{V}(x, I, w, \nu) = \mathcal{V}(x, I, w, \nu')$, since ν and ν' agree on the free variables of x. If x is x_i, for some $i \in \{1, \ldots, n\}$, then $\mathcal{V}(x_i\theta, I, w, \nu) = \mathcal{V}(t_i, I, w, \nu) = \mathcal{V}(x_i, I, w, \nu')$, since $\nu'(x_i) = \mathcal{V}(t_i, I, w, \nu)$.

Suppose t is a constant C. Then $\mathcal{V}(C\theta, I, w, \nu) = \mathcal{V}(C, I, w, \nu) = \mathcal{V}(C, I, w, \nu')$.

Suppose t is an abstraction. I can assume without loss of generality that, for $i = 1, \ldots, n$, x_i is free in t.

(a) Suppose that t is an abstraction $\lambda x.s$ of type $\alpha \to \beta$ such that, for all $i \in \{1, \ldots, n\}$, x is not free in t_i. Then

$$\begin{aligned}
&\mathcal{V}((\lambda x.s)\theta, I, w, \nu) \\
&= \mathcal{V}(\lambda x.(s\theta), I, w, \nu) \quad \text{[Proposition 2.8]} \\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(s\theta, I, w, \overline{\nu}), \\
&\quad \text{where } \overline{\nu} \text{ is } \nu \text{ except } \overline{\nu}(x) = d \\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(s, I, w, \overline{\nu}'), \text{ where } \overline{\nu}'(x_i) = \mathcal{V}(t_i, I, w, \overline{\nu}), \\
&\quad \text{for all } i, \text{ and } \overline{\nu}'(y) = \overline{\nu}(y), \text{ for } y \notin \{x_1, \ldots, x_n\} \quad \text{[induction hypothesis]} \\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(s, I, w, \overline{\nu'}), \\
&\quad \text{where } \overline{\nu'} \text{ is } \nu' \text{ except } \overline{\nu'}(x) = d, \text{ and } \nu'(x_i) = \mathcal{V}(t_i, I, w, \nu), \text{ for } i = 1, \ldots, n, \\
&\quad \text{and } \nu'(y) = \nu(y), \text{ for } y \notin \{x_1, \ldots, x_n\} \\
&\quad \text{[since } x \text{ is not free in any } t_i, \ \nu \text{ and } \overline{\nu} \text{ agree on the free variables in each } t_i; \\
&\quad \text{hence } \mathcal{V}(t_i, I, w, \nu) = \mathcal{V}(t_i, I, w, \overline{\nu}); \text{ thus } \overline{\nu}' = \overline{\nu'}\ ] \\
&= \mathcal{V}(\lambda x.s, I, w, \nu').
\end{aligned}$$

(b) Suppose that t is an abstraction $\lambda x.s$ of type $\alpha \to \beta$ such that, for some $i \in \{1, \ldots, n\}$, x is free in t_i. Then

$$\begin{aligned}
&\mathcal{V}((\lambda x.s)\theta, I, w, \nu) \\
&= \mathcal{V}(\lambda y.(s(\{x/y\} \cup \theta)), I, w, \nu) \quad \text{[Proposition 2.8]} \\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(s(\{x/y\} \cup \theta), I, w, \overline{\nu}), \\
&\quad \text{where } \overline{\nu} \text{ is } \nu \text{ except } \overline{\nu}(y) = d \\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(s, I, w, \overline{\nu}'), \text{ where } \overline{\nu}'(x) = \mathcal{V}(y, I, w, \overline{\nu}),
\end{aligned}$$

$\overline{\nu}'(x_i) = \mathcal{V}(t_i, I, w, \overline{\nu})$, for $i = 1, \ldots, n$, and $\overline{\nu}'(z) = \overline{\nu}(z)$, for $z \notin (\{x_1, \ldots, x_n\} \cup \{x\})$
[induction hypothesis]
$=$ the function whose value for each $d \in \mathcal{D}_\alpha$ is $\mathcal{V}(s, I, w, \overline{\nu'})$,
where $\overline{\nu'}$ is ν' except $\overline{\nu'}(x) = d$, and $\nu'(x_i) = \mathcal{V}(t_i, I, w, \nu)$, for $i = 1, \ldots, n$,
and $\nu'(z) = \nu(z)$, for $z \notin \{x_1, \ldots, x_n\}$
[$\overline{\nu}'$ and $\overline{\nu'}$ agree, except possibly on y]
$= \mathcal{V}(\lambda x.s, I, w, \nu')$.

Suppose t has the form $(u\ v)$. Then

$\mathcal{V}((u\ v)\theta, I, w, \nu)$
$= \mathcal{V}((u\theta\ v\theta), I, w, \nu)$
$= \mathcal{V}(u\theta, I, w, \nu)(\mathcal{V}(v\theta, I, w, \nu))$
$= \mathcal{V}(u, I, w, \nu')(\mathcal{V}(v, I, w, \nu'))$ [induction hypothesis]
$= \mathcal{V}((u\ v), I, w, \nu')$,

where $\nu'(x_i) = \mathcal{V}(t_i, I, w, \nu)$, for $i = 1, \ldots, n$, and $\nu'(y) = \nu(y)$, for $y \notin \{x_1, \ldots, x_n\}$.

Suppose that t has the form $(t_1, \ldots, t_n)$. Then

$\mathcal{V}((t_1, \ldots, t_n)\theta, I, w, \nu)$
$= \mathcal{V}((t_1\theta, \ldots, t_n\theta), I, w, \nu)$
$= (\mathcal{V}(t_1\theta, I, w, \nu), \ldots, (\mathcal{V}(t_n\theta, I, w, \nu))$
$= (\mathcal{V}(t_1, I, w, \nu'), \ldots, (\mathcal{V}(t_n, I, w, \nu'))$ [induction hypothesis]
$= \mathcal{V}((t_1, \ldots, t_n), I, w, \nu')$,

where $\nu'(x_i) = \mathcal{V}(t_i, I, w, \nu)$, for $i = 1, \ldots, n$, and $\nu'(y) = \nu(y)$, for $y \notin \{x_1, \ldots, x_n\}$. □

The next example shows that the condition that t be non-modal in Proposition 3.4 cannot be dropped. The reason for this comes from a basic asymmetry in the treatment of constants and variables – the meaning of constants can change from world to world, while each variable assignment assigns the same meaning to a variable in each world.

Example 3.1. Let α be a type, $p : \alpha \to \Omega$, $C : \alpha$, $t \equiv \Box_i(p\ x)$, and $\theta \equiv \{x/C\}$. Also let I be an interpretation with two worlds w and w', an accessibility relation R_i such that $w\ R_i\ w'$, domain $\mathfrak{D}_\alpha \equiv \{a, b\}$, and mapping V, where $V(p, w) = V(p, w') =$ relation that is true on a and false on b, and $V(C, w) = a$ and $V(C, w') = b$. Finally, let ν be any variable assignment that maps x to a.

Then $\mathcal{V}(t\theta, I, w, \nu) = \mathcal{V}(\Box_i(p\ C), I, w, \nu) = \bot$. But $\mathcal{V}(t, I, w, \nu') = \mathcal{V}(\Box_i(p\ x), I, w, \nu') = \top$, where $\nu'(x) = \mathcal{V}(C, I, w, \nu) = V(C, w) = a$ and $\nu'(y) = \nu(y)$, for $y \neq x$.

In a special case, the condition that t be non-modal in Proposition 3.4 can be dropped.

Proposition 3.5. *Let t be a term, I an interpretation, w a world in I, ν a variable assignment, and x and y variables. Then $\mathcal{V}(t\{x/y\}, I, w, \nu) = \mathcal{V}(t, I, w, \nu')$, where $\nu'(x) = \nu(y)$, and $\nu'(z) = \nu(z)$, for $z \neq x$.*

Proof. Most of the proof is the same as the proof of Proposition 3.4. There is one more case to consider.

Suppose that t has the form $\Box_i\varphi$. Then

$$\begin{aligned}
&\mathcal{V}((\Box_i\varphi)\{x/y\}, I, w, \nu) = \top \\
\text{iff } &\mathcal{V}(\Box_i(\varphi\{x/y\}), I, w, \nu) = \top \\
\text{iff } &\mathcal{V}(\varphi\{x/y\}, I, w', \nu) = \top, \text{ for each } w' \text{ such that } w\, R_i\, w' \\
\text{iff } &\mathcal{V}(\varphi, I, w', \nu') = \top, \text{ for each } w' \text{ such that } w\, R_i\, w' \quad \text{[induction hypothesis]} \\
\text{iff } &\mathcal{V}(\Box_i\varphi, I, w, \nu') = \top,
\end{aligned}$$

where $\nu'(x) = \nu(y)$, and $\nu'(z) = \nu(z)$, for $z \neq x$. □

Proposition 3.6. *Let $(\lambda x.s\ t)$ be a term, where s is non-modal, I an interpretation, w a world in I, and ν a variable assignment. Then $\mathcal{V}((\lambda x.s\ t), I, w, \nu) = \mathcal{V}(s\{x/t\}, I, w, \nu)$.*

Proof.

$$\begin{aligned}
&\mathcal{V}((\lambda x.s\ t), I, w, \nu) \\
&= \mathcal{V}(\lambda x.s, I, w, \nu)\ (\mathcal{V}(t, I, w, \nu)) \\
&= \mathcal{V}(s, I, w, \nu'), \text{where } \nu' \text{ is } \nu \text{ except that } \nu'(x) = \mathcal{V}(t, I, w, \nu) \\
&= \mathcal{V}(s\{x/t\}, I, w, \nu) \quad \text{[Proposition 3.4]}.
\end{aligned}$$

□

Proposition 3.7. *Let s and t be terms of type α, I an interpretation, w a world in I, and ν a variable assignment. Then $\mathcal{V}(s =_\alpha t, I, w, \nu) = \top$ iff $\mathcal{V}(s, I, w, \nu) = \mathcal{V}(t, I, w, \nu)$.*

Proof.

$$\begin{aligned}
&\mathcal{V}(s =_\alpha t, I, w, \nu) = \top \\
\text{iff } &\mathcal{V}((=_\alpha s), I, w, \nu)\ (\mathcal{V}(t, I, w, \nu)) = \top \\
\text{iff } &(\mathcal{V}(=_\alpha, I, w, \nu)\ (\mathcal{V}(s, I, w, \nu)))\ (\mathcal{V}(t, I, w, \nu)) = \top \\
\text{iff } &\mathcal{V}(s, I, w, \nu) = \mathcal{V}(t, I, w, \nu).
\end{aligned}$$

□

Proposition 3.8. *Let t be a non-modal term, s and r terms of type α, I an interpretation, w a world in I, and ν a variable assignment. If $\mathcal{V}(s =_\alpha r, I, w, \nu) = \top$, then $\mathcal{V}(t\{x/s\}, I, w, \nu) = \mathcal{V}(t\{x/r\}, I, w, \nu)$.*

Proof. Proposition 3.7 shows that $\mathcal{V}(s, I, w, \nu) = \mathcal{V}(r, I, w, \nu)$. Then

$$\begin{aligned}
&\mathcal{V}(t\{x/s\}, I, w, \nu) \\
&= \mathcal{V}(t, I, w, \nu'), \text{ where } \nu' \text{ is } \nu \text{ except that } \nu'(x) = \mathcal{V}(s, I, w, \nu) \quad \text{[Proposition 3.4]} \\
&= \mathcal{V}(t, I, w, \nu'), \text{ where } \nu' \text{ is } \nu \text{ except that } \nu'(x) = \mathcal{V}(r, I, w, \nu) \\
&= \mathcal{V}(t\{x/r\}, I, w, \nu) \quad \text{[Proposition 3.4]}.
\end{aligned}$$

□

The next result will be important for the equational reasoning introduced in Section 5.

Proposition 3.9. *Let t be a non-modal term of type α, s a subterm of t at occurrence o such that s has type β, r a term of type β, I an interpretation, and w a world in I. Suppose that $\mathcal{V}(s =_\beta r, I, w, \nu) = \top$, for each variable assignment ν. Then $\mathcal{V}(t =_\alpha t[s/r]_o, I, w, \nu) = \top$, for each variable assignment ν.*

Proof. The proof is by induction on the length n of o.

Suppose first that $n = 0$. Thus t is s and $t[s/r]_o$ is r. Suppose that ν is a variable assignment. Then $\mathcal{V}(t =_\alpha t[s/r]_o, I, w, \nu) = \mathcal{V}(s =_\beta r, I, w, \nu) = \top$.

Suppose next that the result holds for occurrences of length n and o has length $n+1$. Since t is non-modal, t cannot have the form $\Box_i q$. Thus t has the form $\lambda x.q$, $(u\ v)$, or $(t_1, \ldots, t_n)$.

Consider the case when t has the form $\lambda x.q$ and $o = 1o'$, for some o'. Suppose that ν is a variable assignment. Then

$$\begin{aligned}
&\mathcal{V}(\lambda x.q, I, w, \nu)\\
&= \text{the function whose value for each } d \in \mathcal{D}_\gamma \text{ is } \mathcal{V}(q, I, w, \nu'),\\
&\quad \text{where } \alpha = \gamma \to \delta \text{ and } \nu' \text{ is } \nu \text{ except } \nu'(x) = d\\
&= \text{the function whose value for each } d \in \mathcal{D}_\gamma \text{ is } \mathcal{V}(q[s/r]_{o'}, I, w, \nu')\\
&\quad \text{[induction hypothesis and Proposition 3.7]}\\
&= \mathcal{V}(\lambda x.(q[s/r]_{o'}), I, w, \nu)\\
&= \mathcal{V}((\lambda x.q)[s/r]_o, I, w, \nu).
\end{aligned}$$

By Proposition 3.7, $\mathcal{V}(\lambda x.q =_\alpha (\lambda x.q)[s/r]_o, I, w, \nu) = \top$.

Consider next the case when t has the form $(u\ v)$ and $o = 1o'$, for some o'. Suppose that ν is a variable assignment. Then

$$\begin{aligned}
&\mathcal{V}((u\ v), I, w, \nu)\\
&= \mathcal{V}(u, I, w, \nu)(\mathcal{V}(v, I, w, \nu))\\
&= \mathcal{V}(u[s/r]_{o'}, I, w, \nu)(\mathcal{V}(v, I, w, \nu)) \quad \text{[induction hypothesis and Proposition 3.7]}\\
&= \mathcal{V}((u[s/r]_{o'}\ v), I, w, \nu)\\
&= \mathcal{V}((u\ v)[s/r]_o, I, w, \nu).
\end{aligned}$$

By Proposition 3.7, $\mathcal{V}((u\ v) =_\alpha (u\ v)[s/r]_o, I, w, \nu) = \top$. The case when $o = 2o'$ is similar.

The case when t has the form $(t_1, \ldots, t_n)$ is similar to the previous one. □

Proposition 3.10. *Let φ be a non-modal formula, I an interpretation, w a world in I, and θ a substitution. Suppose that $\mathcal{V}(\varphi, I, w, \nu) = \top$, for each variable assignment ν. Then $\mathcal{V}(\varphi\theta, I, w, \nu) = \top$, for each variable assignment ν.*

Proof. Let θ be $\{x_1/t_1, \ldots, x_n/t_n\}$ and ν a variable assignment. By Proposition 3.4, $\mathcal{V}(\varphi\theta, I, w, \nu) = \mathcal{V}(\varphi, I, w, \nu')$, where $\nu'(x_i) = \mathcal{V}(t_i, I, w, \nu)$, for $i = 1, \ldots, n$, and $\nu'(y) = \nu(y)$, for $y \notin \{x_1, \ldots, x_n\}$. Hence $\mathcal{V}(\varphi\theta, I, w, \nu) = \top$. □

Before moving on to other issues, I make a remark about a restriction of the syntax of formulas that is made in [FM98] to avoid some semantic difficulties. Consider the terms $(\lambda x.\Diamond_i(p\ x)\ C)$ and $\Diamond_i(\lambda x.(p\ x)\ C)$, where p is a predicate and C is a nullary data constructor.

As pointed out in [FM98, p.195], these two terms have different meanings that are captured by the different scopings of the respective abstractions. The difference in meaning comes about essentially because in the first formula the meaning of C is given by the current world, while in the second the meaning of C is given by a world accessible to this world. In fact, the second one has exactly the same meaning as the term $\Diamond_i(p\ C)$. In the logic of [FM98], the term $\Diamond_i(p\ C)$ is not admitted, but this is an unattractive approach for genuine applications as shall be shown later; neither is it necessary. So, to avoid a heavy notational burden that would compromise comprehensibility in applications, I most definitely want to admit formulas like $\Diamond_i(p\ t)$ and $\Box_i(p\ t)$, where t is a (closed) term representing some individual.

Next come the important concepts of validity and consequence.

Definition 3.8. A formula φ is *valid at a world w in an interpretation I* if $\mathcal{V}(\varphi, I, w, \nu) = \top$, for all variable assignments ν.

Definition 3.9. A formula φ is *valid in an interpretation* if it is valid at each world in the interpretation.

Definition 3.10. An interpretation $\langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}},\ V\rangle$ is said to be *based on* the frame $\langle W, \{R_i\}_{i=1}^m\rangle$.

Definition 3.11. A formula is *valid in a frame* if it is valid in every interpretation based on the frame.

Definition 3.12. If $\mathbf{L}$ is a class of frames, then a formula is $\mathbf{L}$-*valid* if it is valid in each frame in $\mathbf{L}$.

Definition 3.13. Let $\mathbf{L}$ be a class of frames, φ a formula, and $\mathcal{G}$ and $\mathcal{L}$ sets of formulas. Then φ is an $\mathbf{L}$-*consequence* of *global assumptions* $\mathcal{G}$ and *local assumptions* $\mathcal{L}$ if the following holds: for each interpretation I based on a frame in $\mathbf{L}$ for which all members of $\mathcal{G}$ are valid in I and for each world w in I such that each member of $\mathcal{L}$ is valid at w in I, it follows that φ is valid at w in I.

Notation. Let $\mathbf{K}$ denote the class of all frames. In the following, I abbreviate '$\mathbf{K}$-valid' to 'valid' and '$\mathbf{K}$-consequence' to 'consequence'.

The definition of $\mathbf{L}$-consequence is used as follows. Consider agent i, for some $i \in \{1, \ldots, m\}$. This agent has a belief base that can be split into two components, its *global beliefs* $\mathcal{G}$ and its *local beliefs* $\mathcal{L}$. The global beliefs will be used as global assumptions and the local beliefs as local assumptions in the tableau theorem prover in Section 5.

Let us call a pair of the form (I, w), where I is an interpretation and w a world in I, a *pointed interpretation*. Then agent i has a distinguished pointed interpretation (I, w) as its *intended pointed interpretation*. This means that the agent believes w to be the actual world and I to provide the worlds accessible to w by the various accessibility relations. Agent i must check that its intended pointed interpretation (I, w) satisfies the following conditions:

- Each $\psi \in \mathcal{G}$ is valid in I.
- Each $\psi \in \mathcal{L}$ is valid at w in I.

Now suppose that φ is a formula that is an **L**-consequence of global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$. Then it follows immediately from the definition of **L**-consequence that φ is valid at w in I. Intuitively, 'φ is true in the intended pointed interpretation'. The agent can then use this information as the basis for choosing an action, for example.

It is shown in Section 5 that the tableau theorem prover there does enable the agent to show that a formula is an **L**-consequence of global and local assumptions. Thus the theorem prover there provides part of the machinery for choosing actions. Note also that the belief base of the agent is likely to be changing frequently so that the intended pointed interpretation will also be changing in a corresponding way (so as the ensure the above two conditions are satisfied at all times).

4 Representation of Individuals

In this section, the application of the logic to the representation of individuals is studied. The main idea is the identification of a class of terms, called *basic terms*, suitable for representing individuals in diverse applications, including multi-agent systems. The most interesting aspect of the class of basic terms is that it includes certain abstractions and therefore is wider than is normally considered for knowledge representation. These abstractions allow one to model sets, multisets, and data of similar types, in a direct way. To define basic terms, one first needs to define normal terms, so the discussion starts there. This section is brief since this topic is covered in detail in [Llo03].

First, some motivation. I use the term 'individual' to mean an item worth identifying and representing in an application; other terminology for the same idea includes 'object' and 'instance'. How should individuals be represented? Well there are many possible data types for this, including integers, natural numbers, floats, characters, strings, and booleans; tuples; sets; multisets; lists; trees; and graphs. Put this way, there does not appear to be much structure in the way these data types are put together to represent individuals. However, it turns out that the data types above can be grouped into three major classes, constructor-based ones, abstractions, and tuples, and this grouping is the basis of the definition of basic terms. The first and third classes of data types are already widely used; for example, in functional programming languages. It is the second class which includes sets and multisets that is the most interesting of the three, so I consider it in more detail.

How should a (finite) set or multiset be represented? First, advantage is taken of the higher-order nature of the logic to identify sets and their characteristic functions; that is, sets are viewed as predicates. With this approach, an obvious representation of sets uses the connectives, so that $\lambda x.(x = 1) \vee (x = 2)$ is the representation of the set $\{1, 2\}$. (The predicate $=$ is used generically for equality in this section.) This kind of representation works well for sets. But the connectives are, of course, not available for multisets, so something more general is needed. An alternative representation for the set $\{1, 2\}$ is the term

$$\lambda x.\mathit{if}\ x = 1\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = 2\ \mathit{then}\ \top\ \mathit{else}\ \bot,$$

and this idea generalises to multisets and similar abstractions. For example,

$$\lambda x.\mathit{if}\ x = A\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ 0$$

is the multiset with 42 occurrences of A and 21 occurrences of B (and nothing else). Thus

abstractions of the form

$$\lambda x.\mathit{if}\ x = t_1\ \mathit{then}\ s_1\ \mathit{else}\ \ldots\ \mathit{if}\ x = t_n\ \mathit{then}\ s_n\ \mathit{else}\ s_0$$

are adopted to represent (extensional) sets, multisets, and so on.

However, before giving the definition of a normal term, some attention has to be paid to the term s_0 in the previous expression. The reason is that s_0 in this abstraction is usually a very specific term. For example, for finite sets, s_0 is $\bot$ and for finite multisets, s_0 is 0. For this reason, the concept of a default term is now introduced. The intuitive idea is that, for each type, there is a (unique) default term such that each abstraction having that type as codomain takes the default term as its value for all but a finite number of points in the domain, that is, s_0 is the default value. The choice of default term depends on the particular application but, since sets and multisets are so useful, one would expect the set of default terms to include $\bot$ and 0. However, there could also be other types for which a default term is needed.

For each type constructor T and types $\alpha_1, \ldots, \alpha_k$, I assume there is chosen a unique *default data constructor* C such that C has signature of the form $\sigma_1 \to \cdots \to \sigma_n \to (T\ \alpha_1 \ldots \alpha_k)$. For example, for Ω, the default data constructor could be $\bot$, for *Int*, the default data constructor could be 0, and for *List* α, the default data constructor could be $[]$, for all α.

Definition 4.1. The set of *default terms*, $\mathfrak{D}$, is defined inductively as follows.

1. If C is a default data constructor of arity n and $t_1, \ldots, t_n \in \mathfrak{D}$ $(n \geq 0)$ such that $C\ t_1 \ldots t_n \in \mathfrak{L}$, then $C\ t_1 \ldots t_n \in \mathfrak{D}$.
2. If $t \in \mathfrak{D}$ and $x \in \mathfrak{V}$, then $\lambda x.t \in \mathfrak{D}$.
3. If $t_1, \ldots, t_n \in \mathfrak{D}$ $(n \geq 0)$ and $(t_1, \ldots, t_n) \in \mathfrak{L}$, then $(t_1, \ldots, t_n) \in \mathfrak{D}$.

Now normal terms can be defined. (Let $\mathbb{N}$ denote the set of natural numbers.)

Definition 4.2. The set of *normal terms*, $\mathfrak{N}$, is defined inductively as follows.

1. If C is a data constructor of arity n and $t_1, \ldots, t_n \in \mathfrak{N}$ $(n \in \mathbb{N})$ such that $C\ t_1 \ldots t_n \in \mathfrak{L}$, then $C\ t_1 \ldots t_n \in \mathfrak{N}$.
2. If $t_1, \ldots, t_n \in \mathfrak{N}$, $s_1, \ldots, s_n \in \mathfrak{N}$ $(n \in \mathbb{N})$, $s_0 \in \mathfrak{D}$ and

   $$\lambda x.\mathit{if}\ x = t_1\ \mathit{then}\ s_1\ \mathit{else}\ \ldots\ \mathit{if}\ x = t_n\ \mathit{then}\ s_n\ \mathit{else}\ s_0 \in \mathfrak{L},$$

   then

   $$\lambda x.\mathit{if}\ x = t_1\ \mathit{then}\ s_1\ \mathit{else}\ \ldots\ \mathit{if}\ x = t_n\ \mathit{then}\ s_n\ \mathit{else}\ s_0 \in \mathfrak{N}.$$

3. If $t_1, \ldots, t_n \in \mathfrak{N}$ $(n \in \mathbb{N})$ and $(t_1, \ldots, t_n) \in \mathfrak{L}$, then $(t_1, \ldots, t_n) \in \mathfrak{N}$.

Part 1 of the definition of the set of normal terms states, in particular, that individual natural numbers, integers, and so on, are normal terms. Also a term formed by applying a data constructor to (all of) its arguments, each of which is a normal term, is a normal term.

As an example of this, consider the following declarations of the data constructors *Circle* and *Rectangle*.

$$\mathit{Circle} : \mathit{Float} \to \mathit{Shape}$$
$$\mathit{Rectangle} : \mathit{Float} \to \mathit{Float} \to \mathit{Shape}.$$

Then (*Circle* 7.5) and (*Rectangle* 42.0 21.3) are normal terms of type *Shape*. However, (*Rectangle* 42.0) is not a normal term as not all arguments to *Rectangle* are given. Normal terms coming from Part 1 of the definition are called *normal structures* and always have a type of the form $T\ \alpha_1 \dots \alpha_k$.

The abstractions formed in Part 2 of the definition are 'almost constant' abstractions since they take the default term s_0 as value for all except a finite number of points in the domain. (The term s_0 is called the *default value* for the abstraction.) They are called *normal abstractions* and always have a type of the form $\beta \to \gamma$. This class of abstractions includes useful data types such as (finite) sets and multisets. More generally, normal abstractions can be regarded as lookup tables, with s_0 as the value for items not in the table.

Part 3 of the definition of normal terms just states that one can form a tuple from normal terms and obtain a normal term. These terms are called *normal tuples* and always have a type of the form $\alpha_1 \times \dots \times \alpha_n$.

Normal terms are not quite what is wanted because, for example, they do not give a unique representation of sets. The problem is that the items of the set can appear in any order in a normal term representing the set. The obvious solution is to define a total order on normal terms and then use this order to give a unique representation of sets and other abstractions. The details of the definition of this are given in [Llo03]. Assuming that this total order $<$ is available, basic terms can now be defined.

Definition 4.3. The set of *basic terms*, $\mathfrak{B}$, is defined inductively as follows.

1. If C is a data constructor of arity n and $t_1, \dots, t_n \in \mathfrak{B}$ $(n \in \mathbb{N})$ such that $C\ t_1 \dots t_n \in \mathfrak{L}$, then $C\ t_1 \dots t_n \in \mathfrak{B}$.

2. If $t_1, \dots, t_n \in \mathfrak{B}$, $s_1, \dots, s_n \in \mathfrak{B}$, $t_1 < \dots < t_n$, $s_i \notin \mathfrak{D}$, for $1 \leq i \leq n$ $(n \in \mathbb{N})$, $s_0 \in \mathfrak{D}$ and

   $$\lambda x.\mathit{if}\ x = t_1\ \mathit{then}\ s_1\ \mathit{else}\ \dots\ \mathit{if}\ x = t_n\ \mathit{then}\ s_n\ \mathit{else}\ s_0 \in \mathfrak{L},$$

   then

   $$\lambda x.\mathit{if}\ x = t_1\ \mathit{then}\ s_1\ \mathit{else}\ \dots\ \mathit{if}\ x = t_n\ \mathit{then}\ s_n\ \mathit{else}\ s_0 \in \mathfrak{B}.$$

3. If $t_1, \dots, t_n \in \mathfrak{B}$ $(n \in \mathbb{N})$ and $(t_1, \dots, t_n) \in \mathfrak{L}$, then $(t_1, \dots, t_n) \in \mathfrak{B}$.

The basic terms from Part 1 of the definition are called *basic structures*, those from Part 2 are called *basic abstractions*, and those from Part 3 are called *basic tuples*.

Definition 4.4. For each $\alpha \in \mathfrak{S}$, define $\mathfrak{B}_\alpha = \{t \in \mathfrak{B} \mid t \text{ has type } \alpha\}$.

The sets $\{\mathfrak{B}_\alpha\}_{\alpha \in \mathfrak{S}}$ play an important role in knowledge representation. For example, for a particular application, the representation space of each class of individuals would be $\mathfrak{B}_\alpha$, for some choice of $\alpha \in \mathfrak{S}$. This approach is systematically followed in the remainder of the paper.

5 Reasoning

This section contains a tableau proof system for the logic that is used to establish whether a formula is an **L**-consequence of global and local assumptions (where **L** is some class of frames).

The tableau proof system employs prefixed formulas as is often the case for modal logics.

Definition 5.1. A *prefix* is a finite sequence of positive integers. A *prefixed formula* is an expression of the form $\sigma\ \varphi$, where σ is a prefix and φ is a formula.

The tableau system works as usual for modal logics. Given that the aim is to show that a formula φ is an **L**-consequence of sets of global and local assumptions, the initial node of the tableau is $1\ \neg\varphi$. At any stage in the construction of the tableau, one of its branches is chosen and the branch extended by one of the tableau rules. Figure 1 gives the basic tableau rules. These rules handle the connectives, modalities, quantifiers, abstractions, and equality. Figure 2 gives the two rules for handling the global and local assumptions. These rules are similar to the corresponding ones given in [FM98] and [Fit02]. The proof is completed when each branch is closed, that is, contains contradictory formulas.

Before getting down to the details of the tableau proof system, I make a few remarks about the tableau rules in Figure 1. While they are, in general terms, familiar, the rules have some unusual aspects. These mainly concern the universal, abstraction, and substitutivity rules that all have the assumption that the formula φ appearing in them must be non-modal. Apparently this assumption cannot simply be dropped; for example, [FM98] and [Fit02] have side conditions in the universal rule that involve adding subscripts to constants in tableau proofs. Here I have taken advantage of the particular form of belief bases that appear in typical agent applications. What is common in these applications is for formulas to have a single modality at the front of the formula and for the remainder of the formula to be non-modal. (Examples of this are given below.) This means that in a tableau proof the modality is quickly stripped off leaving the non-modal part exposed to which the universal and other related rules now apply without restriction. While admittedly heavy-handed, this approach does seem appropriate for multi-agent applications, at least.

I now turn to the technical details of the tableau proof system.

Definition 5.2. A tableau branch is *closed* if it contains both $\sigma\ \varphi$ and $\sigma\ \neg\varphi$, for some prefix σ and formula φ. A branch that is not closed is *open*. A tableau is *closed* if every branch is closed.

Definition 5.3. A *tableau proof* for a formula φ is a closed tableau for $1\ \neg\varphi$.

In this paper, I concentrate on the (multi-modal) logic K_m (where the m refers to the number of modalities) which has the tableau system given by the rules in Figures 1 and 2 and for which the corresponding set of frames is **K**.

The next task is to prove the soundness of the tableau system. The key result that needs to be established is that, if a formula has a tableau proof using the rules in Figures 1 and 2, then the formula is a **K**-consequence of the global and local assumptions. To this end, the concept of a satisfiable set of prefixed formulas is introduced.

Definition 5.4. Let **L** be a class of frames. A set S of prefixed formulas is **L**-*satisfiable* with respect to a set $\mathcal{G}$ of global assumptions and a set $\mathcal{L}$ of local assumptions if there exists an interpretation $I \equiv \langle W, \{R_i\}_{i=1}^{m}, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}},\ V\rangle$ based on a frame in **L**, a variable assignment ν, and a mapping F of prefixes to worlds such that the following hold.

(Conjunctive rules) For any prefix σ,

$$\frac{\sigma\ \ \varphi \wedge \psi}{\begin{array}{c}\sigma\ \ \varphi\\ \sigma\ \ \psi\end{array}} \qquad \frac{\sigma\ \ \neg(\varphi \vee \psi)}{\begin{array}{c}\sigma\ \ \neg\varphi\\ \sigma\ \ \neg\psi\end{array}} \qquad \frac{\sigma\ \ \neg(\varphi \to \psi)}{\begin{array}{c}\sigma\ \ \varphi\\ \sigma\ \ \neg\psi\end{array}}$$

(Disjunctive rules) For any prefix σ,

$$\frac{\sigma\ \ \varphi \vee \psi}{\sigma\ \ \varphi \mid \sigma\ \ \psi} \qquad \frac{\sigma\ \ \neg(\varphi \wedge \psi)}{\sigma\ \ \neg\varphi \mid \sigma\ \ \neg\psi} \qquad \frac{\sigma\ \ \varphi \to \psi}{\sigma\ \ \neg\varphi \mid \sigma\ \ \psi}$$

(Double negation rule) For any prefix σ,

$$\frac{\sigma\ \ \neg\neg\varphi}{\sigma\ \ \varphi}$$

(Possibility rules) If the prefix $\sigma.n_i$ is new to the branch, where $i \in \{1, \ldots, m\}$,

$$\frac{\sigma\ \ \Diamond_i\varphi}{\sigma.n_i\ \ \varphi} \qquad \frac{\sigma\ \ \neg\Box_i\varphi}{\sigma.n_i\ \ \neg\varphi}$$

(Necessity rules) If the prefix $\sigma.n_i$ already occurs on the branch, where $i \in \{1, \ldots, m\}$,

$$\frac{\sigma\ \ \Box_i\varphi}{\sigma.n_i\ \ \varphi} \qquad \frac{\sigma\ \ \neg\Diamond_i\varphi}{\sigma.n_i\ \ \neg\varphi}$$

(Existential rules) If y is a variable of type α new to the branch,

$$\frac{\sigma\ \ \exists_\alpha x.\varphi}{\sigma\ \ \varphi\{x/y\}} \qquad \frac{\sigma\ \ \neg\forall_\alpha x.\varphi}{\sigma\ \ \neg\varphi\{x/y\}}$$

(Universal rules) If φ is a non-modal formula and t is a term of type α,

$$\frac{\sigma\ \ \forall_\alpha x.\varphi}{\sigma\ \ \varphi\{x/t\}} \qquad \frac{\sigma\ \ \neg\exists_\alpha x.\varphi}{\sigma\ \ \neg\varphi\{x/t\}}$$

(Abstraction rules) If φ is a non-modal formula,

$$\frac{\sigma\ \ (\lambda x.\varphi\ \ t)}{\sigma\ \ \varphi\{x/t\}} \qquad \frac{\sigma\ \ \neg(\lambda x.\varphi\ \ t)}{\sigma\ \ \neg\varphi\{x/t\}}$$

(Reflexivity rule) If t is a term of type α and the prefix σ already occurs on the branch,

$$\frac{}{\sigma\ \ t =_\alpha t}$$

(Substitutivity rule) If φ is a non-modal formula containing the free variable x of type α,

$$\frac{\begin{array}{c}\sigma\ \ s =_\alpha t\\ \sigma\ \ \varphi\{x/s\}\end{array}}{\sigma\ \ \varphi\{x/t\}}$$

Figure 1: Basic tableau rules

(Global assumption rule) If ψ is a global assumption and the prefix σ already occurs on the branch,

$$\frac{}{\sigma \ \psi}$$

(Local assumption rule) If ψ is a local assumption,

$$\frac{}{1 \ \psi}$$

Figure 2: Tableau rules for global and local assumptions

1. Each $\psi \in \mathcal{G}$ is valid in I.
2. Each $\psi \in \mathcal{L}$ is valid at $F(1)$ in I.
3. If the prefixes σ and $\sigma.n_i$ both occur in S, then $F(\sigma)\ R_i\ F(\sigma.n_i)$.
4. If $\sigma\ \varphi$ is in S, then $\mathcal{V}(\varphi, I, F(\sigma), \nu) = \top$.

A tableau branch is said to be **L**-*satisfiable* if the set of prefixed formulas on it is **L**-satisfiable. A tableau is said to be **L**-*satisfiable* if it has a branch that is **L**-satisfiable.

Notation. In the following, 'satisfiable' will mean '**K**-satisfiable'.

Proposition 5.1. *A closed tableau is not satisfiable.*

Proof. Suppose some closed tableau is satisfiable. Thus some branch of it must be satisfiable. Let S be the set of prefixed formulas on this branch. Suppose S is satisfiable using the interpretation I, variable assignment ν, and mapping F. Since the tableau is closed, the branch is closed and there exist prefixed formulas $\sigma\ \varphi$ and $\sigma\ \neg\varphi$ on the branch, for some prefix σ and formula φ. Thus $\mathcal{V}(\varphi, I, F(\sigma), \nu) = \top$ and $\mathcal{V}(\neg\varphi, I, F(\sigma), \nu) = \top$. But $\mathcal{V}(\neg\varphi, I, F(\sigma), \nu) = \top$ iff $\mathcal{V}(\varphi, I, F(\sigma), \nu) = \bot$, by Proposition 3.1, which gives a contradiction. Thus the closed tableau is not satisfiable. □

Since, for any class **L** of frames, an **L**-satisfiable set of prefixed formulas is satisfiable, it follows from Proposition 5.1 that a closed tableau is not **L**-satisfiable.

Proposition 5.2. *If a tableau rule (from Figures 1 and 2) is applied to a satisfiable tableau, then the resulting tableau is satisfiable.*

Proof. Suppose that $\mathcal{T}$ is a satisfiable tableau and some tableau rule is applied to it on branch $\mathcal{B}$. Now some branch of $\mathcal{T}$ is satisfiable. If this branch is not $\mathcal{B}$, then the tableau rule has no effect on $\mathcal{B}$ and this branch, and hence the new tableau, is satisfiable after the application of the tableau rule.

Thus I can assume that $\mathcal{B}$ is satisfiable. Suppose the set of prefixed formulas on $\mathcal{B}$ is satisfiable via the interpretation I, variable assignment ν, and mapping F. Each of the tableau rules is now examined in turn.

Conjunctive rules Consider the first of these. To show satisfiability of the tableau extended by this rule, it suffices to show that $\mathcal{V}(\varphi, I, F(\sigma), \nu) = \top$ and $\mathcal{V}(\psi, I, F(\sigma), \nu) = \top$. Since the original tableau is satisfiable, it follows that $\mathcal{V}(\varphi \wedge \psi, I, F(\sigma), \nu) = \top$. The result now follows from Proposition 3.1. The proofs of the other conjunctive rules are similar.

Disjunctive rules This is similar to the case for the conjunctive rules.

Double negation rule Since the original tableau is satisfiable, it follows that $\mathcal{V}(\neg\neg\varphi, I, F(\sigma), \nu) = \top$. By two applications of Proposition 3.1, $\mathcal{V}(\varphi, I, F(\sigma), \nu) = \top$. Thus the extended tableau is also satisfiable.

Possibility rules Consider the first of these. Since the original tableau is satisfiable, it follows that $\mathcal{V}(\Diamond_i\varphi, I, F(\sigma), \nu) = \top$. By Proposition 3.1, for some w' such that $F(\sigma)\ R_i\ w'$, $\mathcal{V}(\varphi, I, w', \nu) = \top$. Now extend the definition of F so that $F(\sigma.n_i) = w'$. Thus $F(\sigma)\ R_i\ F(\sigma.n_i)$ and $\mathcal{V}(\varphi, I, F(\sigma.n_i), \nu) = \top$, and so the extended tableau is satisfiable. The proof of the other possibility rule is similar.

Necessity rules This is similar to the case for the possibility rules.

Existential rules Consider the first of these. Since the original tableau is satisfiable, it follows that $\mathcal{V}(\exists_\alpha x.\varphi, I, F(\sigma), \nu) = \top$. By Proposition 3.1, $\mathcal{V}(\varphi, I, F(\sigma), \nu') = \top$, for some x-variant ν' of ν. Define the variable assignment ν^* by $\nu^*(y) = \nu'(x)$ and $\nu^*(z) = \nu(z)$, for $z \neq y$. By Proposition 3.5, $\mathcal{V}(\varphi\{x/y\}, I, F(\sigma), \nu^*) = \mathcal{V}(\varphi, I, F(\sigma), \nu^{*\prime})$, where $\nu^{*\prime}(x) = \nu^*(y)$ and $\nu^{*\prime}(z) = \nu^*(z)$, for $z \neq x$. I claim $\nu^{*\prime}$ and ν' are y-variants. To see this, note that $\nu^{*\prime}(x) = \nu^*(y) = \nu'(x)$ and, for $z \neq x$ and $z \neq y$, $\nu^{*\prime}(z) = \nu^*(z) = \nu'(z)$. Thus, since $\nu^{*\prime}$ and ν' are y-variants and y does not occur in φ, it follows from Proposition 3.3 that $\mathcal{V}(\varphi, I, F(\sigma), \nu^{*\prime}) = \mathcal{V}(\varphi, I, F(\sigma), \nu') = \top$. Hence $\mathcal{V}(\varphi\{x/y\}, I, F(\sigma), \nu^*) = \top$. Now, since ν and ν^* are y-variants and y is new to the branch, it follows that the extended branch is satisfiable via the interpretation I, variable assignment ν^*, and mapping F. The proof of the other existential rule is similar.

Universal rules Consider the first of these. To show satisfiability of the tableau extended by this rule, it suffices to show that $\mathcal{V}(\varphi\{x/t\}, I, F(\sigma), \nu) = \top$. Since the original tableau is satisfiable, it follows that $\mathcal{V}(\forall_\alpha x.\varphi, I, F(\sigma), \nu) = \top$. By Proposition 3.1, $\mathcal{V}(\forall_\alpha x.\varphi, I, F(\sigma), \nu) = \top$ iff, for every x-variant ν', $\mathcal{V}(\varphi, I, F(\sigma), \nu') = \top$. Define the variable assignment ν' by $\nu'(x) = \mathcal{V}(t, I, F(\sigma), \nu)$, and $\nu'(y) = \nu(y)$, for $y \neq x$. By Proposition 3.4, $\mathcal{V}(\varphi\{x/t\}, I, F(\sigma), \nu) = \mathcal{V}(\varphi, I, F(\sigma), \nu')$. Thus $\mathcal{V}(\varphi\{x/t\}, I, F(\sigma), \nu) = \top$. The proof for the second of the universal rules is similar.

Abstraction rules Consider the first of these. To show satisfiability of the tableau extended by this rule, it suffices to show that $\mathcal{V}(\varphi\{x/t\}, I, F(\sigma), \nu) = \top$. Since the original tableau is satisfiable, it follows that $\mathcal{V}((\lambda x.\varphi\ t), I, F(\sigma), \nu) = \top$. The result now follows immediately, since by Proposition 3.6, $\mathcal{V}((\lambda x.\varphi\ t), I, F(\sigma), \nu) = \mathcal{V}(\varphi\{x/t\}, I, F(\sigma), \nu)$. The proof for the second of the abstraction rules is similar.

Reflexivity rule To show satisfiability of the tableau extended by this rule, it suffices to show that $\mathcal{V}((t =_\alpha t), I, F(\sigma), \nu) = \top$. But this follows immediately since the meaning of $=_\alpha$ is the identity mapping.

Substitutivity rule To show satisfiability of the tableau extended by this rule, it suffices to show that $\mathcal{V}(\varphi\{x/t\}, I, F(\sigma), \nu) = \top$. Since the original tableau is satisfiable, it follows that $\mathcal{V}(s =_\alpha t, I, F(\sigma), \nu) = \top$ and $\mathcal{V}(\varphi\{x/s\}, I, F(\sigma), \nu) = \top$. The result now follows immediately from Proposition 3.8.

Global assumption rule Since the (original) tableau is satisfiable and ψ is a global assumption, ψ is valid in I. Thus $\mathcal{V}(\psi, I, F(\sigma), \nu) = \top$. Hence the extended tableau is satisfiable.

Local assumption rule Since the (original) tableau is satisfiable and ψ is a local assumption, ψ is valid at $F(1)$ in I. Thus $\mathcal{V}(\psi, I, F(1), \nu) = \top$. Hence the extended tableau is satisfiable. □

Now the soundness result can be proved. (Recall that 'consequence' means '**K**-consequence'.)

Proposition 5.3. *If a formula φ has a tableau proof from global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$, then φ is a consequence of global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$.*

Proof. Suppose φ has a tableau proof, but is not a consequence of global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$. Since φ has a tableau proof, there is a closed tableau $\mathcal{T}$ beginning with the prefixed formula $1\ \neg\varphi$. Let $\mathcal{T}_0$ be the tableau consisting of the single node $1\ \neg\varphi$. Thus $\mathcal{T}$ results from $\mathcal{T}_0$ by applying various tableau rules in Figures 1 and 2.

Since φ is not a consequence, there is an interpretation I, a world w in I, and a variable assignment ν such that each $\psi \in \mathcal{G}$ is valid in I, each $\psi \in \mathcal{L}$ is valid at w in I, and $\mathcal{V}(\varphi, I, w, \nu) = \bot$. Define the mapping F by $F(1) = w$. Clearly, $\{1\ \neg\varphi\}$ is satisfiable, using I, ν, and F. Thus $\mathcal{T}_0$ is satisfiable.

Since $\mathcal{T}_0$ is satisfiable, so is any tableau obtained from $\mathcal{T}_0$ by applying tableau rules, by Proposition 5.2. Thus $\mathcal{T}$ is satisfiable. But this gives a contradiction, by Proposition 5.1, since $\mathcal{T}$ is closed. Thus φ is a consequence of global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$. □

The following result is an immediate corollary of Proposition 5.3. (Recall that 'valid' means '**K**-valid'.)

Proposition 5.4. *If a formula φ has a tableau proof from empty sets of global and local assumptions, then φ is valid.*

It is convenient to allow the use of lemmas in proofs. These are handled by the lemma introduction rule in Figure 3. The assumption of this rule is that the proof of the lemma uses the same global and local assumptions as the proof of the main result, of course.

(Lemma introduction rule) If there is a tableau proof of φ,

$$\frac{}{1\ \varphi}$$

Figure 3: Lemma introduction rule

I now establish the soundness of the lemma introduction rule. The complication is that the rule can be applied recursively, that is, one can use the lemma introduction rule to establish a lemma that could then be introduced into the main proof, and so on. This leads to the following definition of the rank of a tableau proof.

Definition 5.5. The *rank* of a tableau proof is defined inductively as follows.

- A tableau proof has rank 0 if it does not use the lemma introduction rule.
- A tableau proof has rank $n + 1$ if the maximum of the ranks of the tableau proofs of the introduced lemmas is n.

Proposition 5.5. *If a formula φ has a tableau proof using the rules in Figures 1, 2 and 3 from global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$, then φ is a consequence of global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$.*

Proof. The proof is by induction on the rank of the proof. If the rank is 0, then this is just Proposition 5.3.

Suppose now that the rank of the proof is $n + 1$. The first task is to show that an application of the lemma introduction rule to a satisfiable tableau results in a satisfiable

tableau. In the same way as in the proof of Proposition 5.2, I can assume that the rule is applied to a satisfiable branch $\mathcal{B}$. Suppose that the set of prefixed formulas on $\mathcal{B}$ is satisfiable via the interpretation I, variable assignment ν, and mapping F.

Let $1\ \varphi$ be the prefixed formula that is added to the branch $\mathcal{B}$ by the application of the lemma introduction rule. Since the rank of the proof of the introduced lemma φ is less than or equal to n, φ is a consequence of global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$, by the induction hypothesis. Consequently, $\mathcal{V}(\varphi, I, F(1), \nu) = \top$. Thus the extended branch is also satisfiable.

Having shown that the lemma introduction rule preserves satisfiability, the remainder of the proof now proceeds in the same way as the proof of Proposition 5.3. □

Many formulas in belief bases are equations, so to increase the usefulness of the tableau theorem prover, it will be convenient to introduce some specialised machinery for equational reasoning. This kind of equational reasoning is commonly used as the foundation for declarative programming languages, as in [Llo03].

If s and t differ only in the names of bound variables, they are said to be *renaming equivalent*. (This is the same as the λ-calculus concept of α-equivalence.)

Definition 5.6. Let t be a non-modal term, $\Box_j(u =_\alpha v)$ a formula, where u and v are non-modal, s a subterm of t at occurrence o, and θ a substitution such that s is renaming equivalent to $u\theta$. Then $t[s/v\theta]_o$ is said to be obtained from t by a *computation step for agent j using input equation $\Box_j(u =_\alpha v)$, redex s at occurrence o*, and *matching substitution θ*.

By Proposition 2.9 and the fact that an instance of a term is a term of the same type, $t[s/v\theta]_o$ is a term of the same type as t.

Definition 5.7. A *computation for agent j* from a term t is a sequence $\{t_i\}_{i=1}^n$ of terms such that the following conditions are satisfied.

1. $t = t_1$.
2. t_{i+1} is obtained from t_i by a computation step for agent j, for $i = 1, \ldots, n-1$.

The term t_1 is called the *goal* of the computation and t_n is called the *answer*.

Proposition 5.6. *Suppose that $\{t_i\}_{i=1}^n$ is a computation for agent j such that $\Box_j(u_i =_{\alpha_i} v_i)$ is the input equation at the ith step, for $i = 1, \ldots, n-1$, and t_1 has type α. Then $\Box_j(t_1 =_\alpha t_n)$ is a consequence of local assumptions $\{\Box_j(u_i =_{\alpha_i} v_i)\}_{i=1}^{n-1}$.*

Proof. Let I be an interpretation and w a world in I such that $\mathcal{V}(\Box_j(u_i =_{\alpha_i} v_i), I, w, \nu) = \top$, for $i = 1, \ldots, n-1$ and each variable assignment ν. It has to be shown that $\mathcal{V}(\Box_j(t_1 =_\alpha t_n), I, w, \nu) = \top$, for each variable assignment ν.

The proof is by induction on n. If $n = 1$, then the result is obvious.

Suppose now that the result holds for computations of length n and consider a computation $\{t_i\}_{i=1}^{n+1}$ of length $n+1$ for which the redex is s_n at occurrence o_n and the matching substitution is θ_n at the nth step. By the induction hypothesis, $\mathcal{V}(\Box_j(t_1 =_\alpha t_n), I, w, \nu) = \top$, for each variable assignment ν. Thus $\mathcal{V}(t_1 =_\alpha t_n, I, w', \nu) = \top$, for each variable assignment ν and each w' such that $w\ R_j\ w'$.

Since $\mathcal{V}(\Box_j(u_n =_{\alpha_n} v_n), I, w, \nu) = \top$, for each variable assignment ν, it follows that $\mathcal{V}(u_n =_{\alpha_n} v_n, I, w', \nu) = \top$, for each variable assignment ν and each w' such that $w\ R_j\ w'$.

Then, by Proposition 3.10, it follows that $\mathcal{V}(u_n\theta_n =_{\alpha_n} v_n\theta_n, I, w', \nu) = \top$, for each variable assignment ν and each w' such that $w\ R_j\ w'$.

Now $t_{n+1} = t_n[s_n/v_n\theta_n]_{o_n}$. Hence, by Proposition 3.9, $\mathcal{V}(t_n =_\alpha t_{n+1}, I, w', \nu) = \top$, for each variable assignment ν and each w' such that $w\ R_j\ w'$. By Proposition 3.7, $\mathcal{V}(t_1 =_\alpha t_{n+1}, I, w', v)$, for each variable assignment ν and each w' such that $w\ R_j\ w'$. Thus $\mathcal{V}(\Box_j(t_1 =_\alpha t_{n+1}), I, w, \nu) = \top$, for each variable assignment ν. □

A common application of Proposition 5.6 is when t_1 is a formula. In this case, the next result is useful.

Proposition 5.7. *Let* **L** *be a class of frames, φ a formula, $\mathcal{G}$ and $\mathcal{L}$ sets of formulas, and $j \in \{1, \ldots, m\}$ Then the following hold.*

1. *$\Box_j(\varphi =_\Omega \top)$ is an* **L***-consequence of global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$ iff $\Box_j\varphi$ is an* **L***-consequence of global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$.*

2. *$\Box_j(\varphi =_\Omega \bot)$ is an* **L***-consequence of global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$ iff $\Box_j\neg\varphi$ is an* **L***-consequence of global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$.*

Proof. The result follows immediately from Proposition 3.2. □

Thus if φ is a formula and a computation has established that $\Box_j(\varphi =_\alpha \top)$ is a consequence of some local assumptions as in Proposition 5.6, then $\Box_j(\varphi =_\alpha \top)$ can be replaced by $\Box_j\varphi$. Similarly, $\Box_j(\varphi =_\alpha \bot)$ can be replaced by $\Box_j\neg\varphi$.

To make use of Proposition 5.6, another tableau rule given in Figure 4 is needed.

(Consequence introduction rule) If φ is a consequence of the local and global assumptions,

$$\frac{}{1\ \varphi}$$

Figure 4: Consequence introduction rule

The next result shows that the consequence introduction rule is sound.

Proposition 5.8. *If a formula φ has a tableau proof using the rules in Figures 1, 2, 3 and 4 from global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$, then φ is a consequence of global assumptions $\mathcal{G}$ and local assumptions $\mathcal{L}$.*

Proof. It suffices to show that an application of the consequence introduction rule to a satisfiable tableau results in a satisfiable tableau. In the same way as in the proof of Proposition 5.2, I can assume that the rule is applied to a satisfiable branch $\mathcal{B}$. Suppose that the set of prefixed formulas on $\mathcal{B}$ is satisfiable via the interpretation I, variable assignment ν, and mapping F.

Let $1\ \varphi$ be the prefixed formula that is added to the branch $\mathcal{B}$ by the application of the consequence introduction rule. Thus φ is a consequence of the global and local assumptions. It now follows immediately from the definitions of satisfiable and consequence that $\mathcal{V}(\varphi, I, F(1), \nu) = \top$, and so the extended tableau is satisfiable. □

It is worth noting that if φ is a consequence of the local and global assumptions, then there is a tableau proof for φ as follows.

1 $\neg\varphi$ 1.

1 φ 2.

Item 1 is the negation of the formula to be proved; 2 is by consequence introduction; now the branch closes by 1 and 2. This means that instead of introducing a formula by consequence introduction, the formula could just as easily be introduced by lemma introduction. In the extended example of Section 6, I will systematically use the lemma introduction rule instead of the consequence introduction rule, even when the formula has been obtained from a computation.

It is common in applications for the global assumptions to include particular kinds of formulas that can be more directly handled by specialised tableau rules. As a typical example, consider the schema

$$\Box_i\varphi \longrightarrow \Box_j\varphi.$$

This is a schema (rather than a formula) since the intention is that it should hold for *any* formula φ and so the φ should be understood as a syntactical variable standing for any formula. I call this kind of schema an *interaction axiom*, since it gives a relationship between the belief states of two agents. Its intuitive meaning is that 'if agent i believes φ, then agent j believes φ'. A second example is the global assumption

$$\Box_i\varphi \longrightarrow \varphi,$$

which states that the beliefs of agent i are true.

The following result provides a systematic method for replacing implicational global assumptions by tableau rules.

Proposition 5.9.

1. *Use of the global assumption $\varphi \longrightarrow \psi$ is equivalent to use of the tableau rule:*

$$\frac{\sigma \;\; \varphi}{\sigma \;\; \psi}$$

2. *Use of the global assumption $\varphi \longrightarrow \Box_j\psi$ is equivalent to use of the tableau rule:*
   If the prefix $\sigma.m_j$ already occurs on the branch,

$$\frac{\sigma \;\; \varphi}{\sigma.m_j \;\; \psi}$$

3. *Use of the global assumption $\Diamond_i\varphi \longrightarrow \psi$ is equivalent to use of the tableau rule:*

$$\frac{\sigma.n_i \;\; \varphi}{\sigma \;\; \psi}$$

4. *Use of the global assumption $\Diamond_i\varphi \longrightarrow \Box_j\psi$ is equivalent to use of the tableau rule:*
   If the prefix $\sigma.m_j$ already occurs on the branch,

$$\frac{\sigma.n_i \;\; \varphi}{\sigma.m_j \;\; \psi}$$

Proof. 1. Suppose first that the tableau rule is available. It is shown that this makes $\varphi \longrightarrow \psi$ available as a global assumption. Suppose that σ is a prefix already occurring on some branch. Consider the prefixed formula

$$\sigma \ \ (\varphi \longrightarrow \psi) \vee \neg(\varphi \longrightarrow \psi).$$

Since the formula is equivalent to $\top$, this could be added (correctly) to the end of the branch. A disjunctive rule can then be used to produce two children

$$\sigma \ \ \varphi \longrightarrow \psi \quad \text{and} \quad \sigma \ \ \neg(\varphi \longrightarrow \psi).$$

From the second of these, the tableau proof can proceed as follows.

$\sigma \ \ \neg(\varphi \longrightarrow \psi) \ \ 1.$
$\sigma \ \ \varphi \ \ 2.$
$\sigma \ \ \neg\psi \ \ 3.$
$\sigma \ \ \psi \ \ 4.$

Items 2 and 3 are from 1 by a conjunctive rule; 4 is from 2 by the proposed rule. Now this branch closes by 3 and 4. What remains in the tableau is the prefixed formula $\sigma \ \ \varphi \longrightarrow \psi$, which is exactly what would result from the use of $\varphi \longrightarrow \psi$ as a global assumption.

For the converse, suppose that $\varphi \longrightarrow \psi$ is a global assumption. It has to be shown that the use of the proposed rule is sound, that is, if the rule is applied to a satisfiable tableau, then the resulting tableau is satisfiable. Suppose the rule is applied to a branch $\mathcal{B}$. As in the proof of Proposition 5.2, I can assume that $\mathcal{B}$ is satisfiable. Suppose the set of prefixed formulas on $\mathcal{B}$ is satisfiable via the interpretation I, variable assignment ν, and mapping F. Since the (original) tableau is satisfiable, $\mathcal{V}(\varphi, I, F(\sigma), \nu) = \top$. Also $\mathcal{V}(\varphi \longrightarrow \psi, I, F(\sigma), \nu) = \top$, since $\varphi \longrightarrow \psi$ is a global assumption. Thus $\mathcal{V}(\psi, I, F(\sigma), \nu) = \top$, by Proposition 3.1, as required.

2. and 3. The proofs of these are similar to the other parts.

4. Suppose first that the tableau rule is available. It is shown that this makes $\Diamond_i\varphi \longrightarrow \Box_j\psi$ available as a global assumption. Suppose that σ is a prefix already occurring on some branch. Consider the prefixed formula

$$\sigma \ \ (\Diamond_i\varphi \longrightarrow \Box_j\psi) \vee \neg(\Diamond_i\varphi \longrightarrow \Box_j\psi).$$

Since the formula is equivalent to $\top$, this could be added (correctly) to the end of the branch. A disjunctive rule can then be used to produce two children

$$\sigma \ \ \Diamond_i\varphi \longrightarrow \Box_j\psi \quad \text{and} \quad \sigma \ \ \neg(\Diamond_i\varphi \longrightarrow \Box_j\psi).$$

From the second of these, the tableau proof can proceed as follows.

$\sigma \ \ \neg(\Diamond_i\varphi \longrightarrow \Box_j\psi) \ \ 1.$
$\sigma \ \ \Diamond_i\varphi \ \ 2.$
$\sigma \ \ \neg\Box_j\psi \ \ 3.$
$\sigma.n_i \ \ \varphi \ \ 4.$
$\sigma.m_j \ \ \neg\psi \ \ 5.$
$\sigma.m_j \ \ \psi \ \ 6.$

Items 2 and 3 are from 1 by a conjunctive rule; 4 is from 2 by possibility rule; 5 is from 3 by possibility rule; 6 is from 4 by the proposed rule. Now this branch closes by 5 and 6. What remains in the tableau is the prefixed formula $\sigma \ \Diamond_i\varphi \longrightarrow \Box_j\psi$, which is exactly what would result from the use of $\Diamond_i\varphi \longrightarrow \Box_j\psi$ as a global assumption.

For the converse, suppose that $\Diamond_i\varphi \longrightarrow \Box_j\psi$ is a global assumption. It has to be shown that the use of the proposed rule is sound, that is, if the rule is applied to a satisfiable tableau, then the resulting tableau is satisfiable. Suppose the rule is applied to a branch $\mathcal{B}$. As in the proof of Proposition 5.2, I can assume that $\mathcal{B}$ is satisfiable. Suppose the set of prefixed formulas on $\mathcal{B}$ is satisfiable via the interpretation I, variable assignment ν, and mapping F. Since the (original) tableau is satisfiable, $\mathcal{V}(\varphi, I, F(\sigma.n_i), \nu) = \top$ and $F(\sigma)\ R_i\ F(\sigma.n_i)$. Hence, by Proposition 3.1, $\mathcal{V}(\Diamond_i\varphi, I, F(\sigma), \nu) = \top$. Also $\mathcal{V}(\Diamond_i\varphi \longrightarrow \Box_j\psi, I, F(\sigma), \nu) = \top$, since $\Diamond_i\varphi \longrightarrow \Box_j\psi$ is a global assumption. Thus $\mathcal{V}(\Box_j\psi, I, F(\sigma), \nu) = \top$, by Proposition 3.1. Since $F(\sigma)\ R_j\ F(\sigma.m_j)$, it follows from this that $\mathcal{V}(\psi, I, F(\sigma.m_j), \nu) = \top$, as required. □

As an alternative to Proposition 5.9, one can replace the $\Box_j$ modality in front of ψ by $\Diamond_j$. In this case, the side condition on the prefix changes.

Proposition 5.10.

1. *Use of the global assumption $\varphi \longrightarrow \Diamond_j\psi$ is equivalent to use of the tableau rule:*
   If the prefix $\sigma.m_j$ is new to the branch,

$$\frac{\sigma \ \ \varphi}{\sigma.m_j \ \ \psi}$$

2. *Use of the global assumption $\Diamond_i\varphi \longrightarrow \Diamond_j\psi$ is equivalent to use of the tableau rule:*
   If the prefix $\sigma.m_j$ is new to the branch,

$$\frac{\sigma.n_i \ \ \varphi}{\sigma.m_j \ \ \psi}$$

Proof. 1. Suppose first that the tableau rule is available. It is shown that this makes $\varphi \longrightarrow \Diamond_j\psi$ available as a global assumption. Suppose that σ is a prefix already occurring on some branch. Consider the prefixed formula

$$\sigma \ \ (\varphi \longrightarrow \Diamond_j\psi) \vee \neg(\varphi \longrightarrow \Diamond_j\psi).$$

Since the formula is equivalent to $\top$, this could be added (correctly) to the end of the branch. A disjunctive rule can then be used to produce two children

$$\sigma \ \ \varphi \longrightarrow \Diamond_j\psi \quad \text{and} \quad \sigma \ \ \neg(\varphi \longrightarrow \Diamond_j\psi).$$

From the second of these, the tableau proof can proceed as follows.

$\sigma \ \ \neg(\varphi \longrightarrow \Diamond_j\psi)$ 1.
$\sigma \ \ \varphi$ 2.
$\sigma \ \ \neg\Diamond_j\psi$ 3.
$\sigma.m_j \ \ \psi$ 4.
$\sigma.m_j \ \ \neg\psi$ 5.

Items 2 and 3 are from 1 by a conjunctive rule; 4 is from 2 by the proposed rule; 5 is from 3 by necessity rule; Now this branch closes by 4 and 5. What remains in the tableau is the prefixed formula $\sigma\ \ \varphi \longrightarrow \Diamond_j\psi$, which is exactly what would result from the use of $\varphi \longrightarrow \Diamond_j\psi$ as a global assumption.

For the converse, suppose that $\varphi \longrightarrow \Diamond_j\psi$ is a global assumption. It has to be shown that the use of the proposed rule is sound, that is, if the rule is applied to a satisfiable tableau, then the resulting tableau is satisfiable. Suppose the rule is applied to a branch $\mathcal{B}$. As in the proof of Proposition 5.2, I can assume that $\mathcal{B}$ is satisfiable. Suppose the set of prefixed formulas on $\mathcal{B}$ is satisfiable via the interpretation I, variable assignment ν, and mapping F. Since the (original) tableau is satisfiable, $\mathcal{V}(\varphi, I, F(\sigma), \nu) = \top$. Also $\mathcal{V}(\varphi \longrightarrow \Diamond_j\psi, I, F(\sigma), \nu) = \top$, since $\varphi \longrightarrow \Diamond_j\psi$ is a global assumption. Thus $\mathcal{V}(\Diamond_j\psi, I, F(\sigma), \nu) = \top$, by Proposition 3.1. Hence, for some w' such that $F(\sigma)\ R_j\ w'$, $\mathcal{V}(\psi, I, w', \nu) = \top$, again by Proposition 3.1. Now define $F(\sigma.m_j) = w'$, so that $F(\sigma)\ R_j\ F(\sigma.m_j)$. It follows that $\mathcal{V}(\psi, I, F(\sigma.m_j), \nu) = \top$, as required, and the resulting tableau is satisfiable.

2. This is similar to Part 1. □

Some global assumptions (usually called *axioms*) arise often in agent applications. Using their traditional names, these axioms are as follows (where $i \in \{1, \ldots, m\}$).

- T_i: $\Box_i\varphi \longrightarrow \varphi$.
- D_i: $\Box_i\varphi \longrightarrow \Diamond_i\varphi$.
- B_i: $\varphi \longrightarrow \Box_i\Diamond_i\varphi$.
- 4_i: $\Box_i\varphi \longrightarrow \Box_i\Box_i\varphi$.
- 5_i: $\Diamond_i\varphi \longrightarrow \Box_i\Diamond_i\varphi$.

It is well known that axiom T_i corresponds to the accessibility relation R_i being reflexive, axiom D_i corresponds to seriality, axiom B_i corresponds to symmetry, axiom 4_i corresponds to transitivity, and axiom 5_i corresponds to euclideanness. Figure 5 gives the tableau rules corresponding to these axioms, together with the rules for the interaction axioms. The correctness of these rules follows immediately from Proposition 5.9. Note that all except the axiom D_i have a third rule that comes from considering the dual of the axiom. For example, the dual of axiom B_i is $\Diamond_i\Box_i\varphi \longrightarrow \varphi$, which is equivalent to the original axiom. The third of the B_i rules is obtained by applying Proposition 5.9 to this dual. Axiom D_i is self dual so nothing new is obtained. The second rule in each case is a variation of the first when a negation appears, and the fourth is a similar variation of the third.

Then, for example, rules T_i, B_i, and 4_i, for $i = 1, \ldots, m$, (together with the rules in Figures 1 and 2) give a tableau system for the logic $S5_m$ which is characterised by the axioms T_i, B_i, and 4_i, for $i = 1, \ldots, m$, and for which the corresponding accessibility relation is an equivalence relation for each agent. Similarly, rules T_i and 4_i, for $i = 1, \ldots, m$, give a tableau system for the logic $S4_m$ which is characterised by the axioms T_i and 4_i, for $i = 1, \ldots, m$, and for which the corresponding accessibility relation is reflexive and transitive for each agent. Hybrid logics which mix modalities that satisfy different axioms are also possible.

Actually, as is well known, finding a suitable set of tableau rules is much more complicated than the preceding paragraph might suggest. The point is that soundness is only one requirement; one also wants completeness and, at least for the propositional subset of the logic,

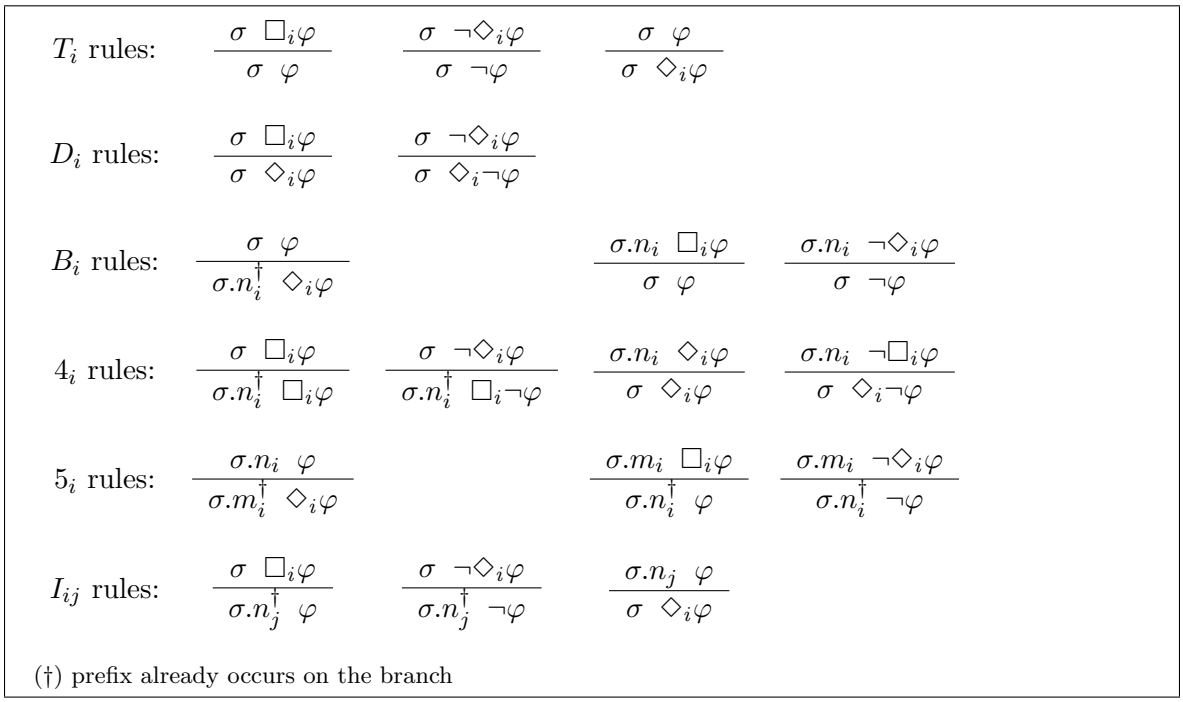

Figure 5: Tableau rules corresponding to various global assumptions

decidability. This means that from amongst the large collection of potential rules, many of which are listed above, a clever choice must be made of a 'minimal' subset that achieves completeness and, for the propositional subset, decidability. For this, desirable rules are ones that are analytic, that is, the formula in the denominator of a rule is a strict subformula of the formula in the numerator. If all rules are analytic, no branch of a tableau can be infinite and decidability is achieved. Unfortunately, the universal rules, for example, are far from being analytic, but for first-order and higher-order logics this is to be expected. In the propositional case, the analytic requirement suggests that the third of the T_i rules in Figure 5 is not a good rule; it can be applied endlessly to make an infinite branch. Similar criticisms can be made of some other rules in Figure 5. For some of the cases I have considered here (for example, the S_{pj} rules in Figure 6 below), the problem of finding a good choice of rules is currently under investigation by modal logicians. Future work involves making a suitable choice of tableau rules for each of the logics to meet the above objectives.

The following example taken from [Bal98] illustrates the use of these rules.

Example 5.1. Consider the agents, Peter, John, and Wendy, with belief modalities $\Box_p$, $\Box_j$, and $\Box_w$. Suppose that Peter believes that *time* is true, Peter believes that John believes that *place* is true, Wendy believes that if Peter believes that *time* is true, then John believes that *time* is true, and Peter believes that John believes that if *time* and *place* are true, then *appointment* is true. These beliefs are captured in the following local assumptions of the belief base.

$\Box_p time$
$\Box_p \Box_j place$
$\Box_w(\Box_p time \longrightarrow \Box_j time)$

$$\Box_p \Box_j (place \wedge time \longrightarrow appointment)$$

Suppose, in addition, that Peter and John satisfy the axioms T and 4, everything believed by Wendy is believed by Peter, and if Peter believes that John believes something, then John believes that Peter believes the same thing. The global assumptions that capture all this are as follows.

$$T_p : \Box_p \varphi \longrightarrow \varphi$$
$$4_p : \Box_p \varphi \longrightarrow \Box_p \Box_p \varphi$$
$$T_j : \Box_j \varphi \longrightarrow \varphi$$
$$4_j : \Box_j \varphi \longrightarrow \Box_j \Box_j \varphi$$
$$I_{wp} : \Box_w \varphi \longrightarrow \Box_p \varphi$$
$$S_{pj} : \Box_p \Box_j \varphi \longrightarrow \Box_j \Box_p \varphi$$

The tableau rules corresponding to these axioms are given in Figure 6.

Suppose now that one wants to show that John believes that Peter believes *appointment* is true, that is,

$$\Box_j \Box_p appointment,$$

is a theorem of the belief base. The tableau proof of this formula is given in Figure 7.

An explanation of this proof follows. Item 1 is the negation of the formula to be proved; 2 to 5 are local assumptions; 6 is from 1 by a possibility rule; 7 is from 6 by an S_{pj} rule; 8 is from 7 by a possibility rule; 9 is from 8 by a possibility rule; 10 is from 5 by a necessity rule; 11 is from 10 by a necessity rule; 12 and 24 are from 11 by a disjunctive rule; 13 and 16 are from 12 by a disjunctive rule; 14 is from 3 by a necessity rule; 15 is from 14 by a necessity rule; this branch now closes by 13 and 15; 17 is from 4 by an I_{wp} rule; 18 and 22 are from 17 by a disjunctive rule; 19 is from 18 by a possibility rule; 20 is from 2 by a 4_p rule; 21 is from 20 by a necessity rule; this branch now closes by 19 and 21; 23 is from 22 by a necessity rule; this branch now closes by 16 and 23; and the remaining branch closes by 9 and 24.

Here is another example, quite different to the preceding one, that arises in learning applications. When learning a function on some set of individuals, the need can arise to show that some predicate which acts on the individuals is stronger than some other predicate. The next example illustrates this using individuals that are sets of items and two predicates on sets.

Example 5.2. Let α be some type and consider the functions [Llo03]

$$setExists_1 : (\alpha \to \Omega) \to \{\alpha\} \to \Omega$$

defined by

$$setExists_1\ p\ t =_\Omega \exists x.((p\ x) \wedge (x \in t)),$$

and

$$\wedge_2 : (\alpha \to \Omega) \to (\alpha \to \Omega) \to \alpha \to \Omega$$

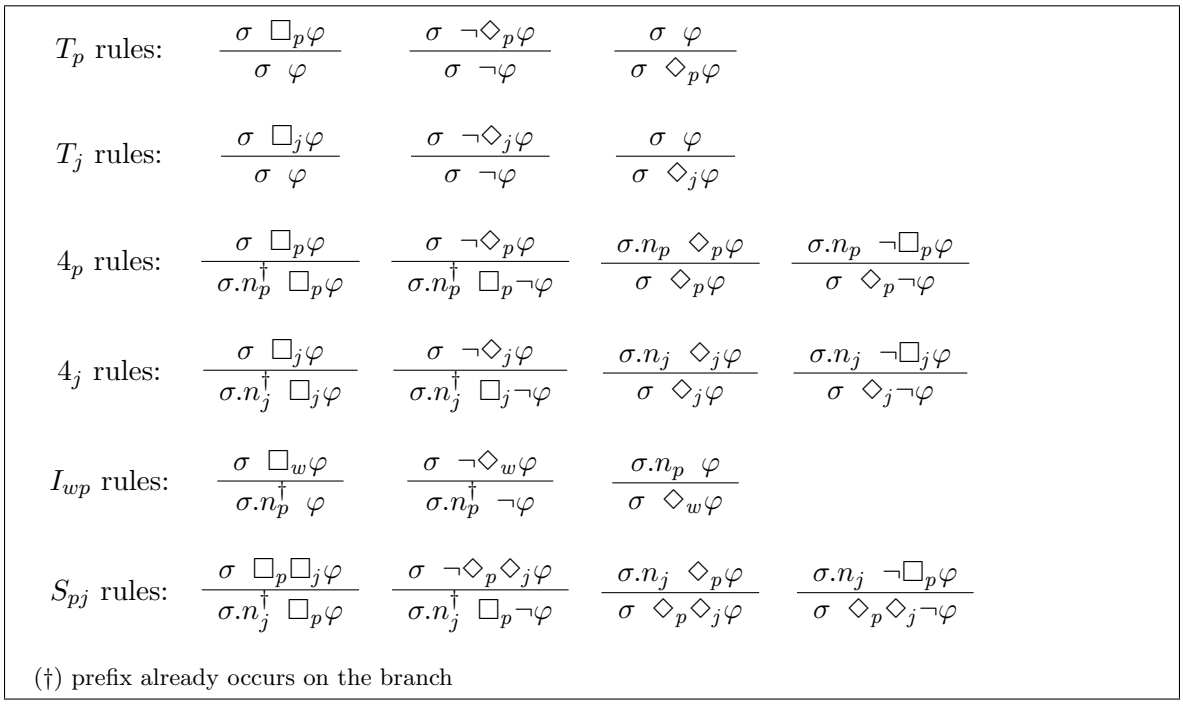

Figure 6: Tableau rules corresponding to the global assumptions of Example 5.1

defined by

$$\wedge_2\ p_1\ p_2\ x =_\Omega (p_1\ x) \wedge (p_2\ x).$$

(In this example, I omit some brackets by use of the convention that function application is left associative.) Let p and q be specific predicates on items of type α and $top : \alpha \to \Omega$ be defined by $top\ x =_\Omega \top$, for all x. Consider now the two predicates

$$(setExists_1\ (\wedge_2\ p\ q))$$

and

$$(setExists_1\ (\wedge_2\ p\ top)).$$

I will show the first predicate is stronger than the second, that is,

$$\Box_j \forall x.((setExists_1\ (\wedge_2\ p\ q)\ x) \longrightarrow (setExists_1\ (\wedge_2\ p\ top)\ x))$$

is a theorem of the background theory that includes the definitions of the functions just introduced and where the agent trying to establish this belief is j. The tableau proof of this formula is given in Figure 8.

An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 is from 1 by a possibility rule; 3 is from 2 by an existential rule; 4 and 5 are from 3 by a conjunctive rule; 6 is by lemma introduction; 7 is from 6 by a necessity rule; 8 is from 4 and 7 by the substitutivity rule; 9 is from 8 by an existential rule; 10, 11, and 12 are from 9 by a conjunctive rule; 13 is by lemma introduction; 14 is from 13 by a necessity rule; 15 is from 5 and 14 by the substitutivity rule; 16 is from 15 by a universal rule; 17 and 18 are from 16 by a disjunctive rule; now one branch closes by 10 and 17 and the other branch closes by 12 and 18. The proof is now complete.

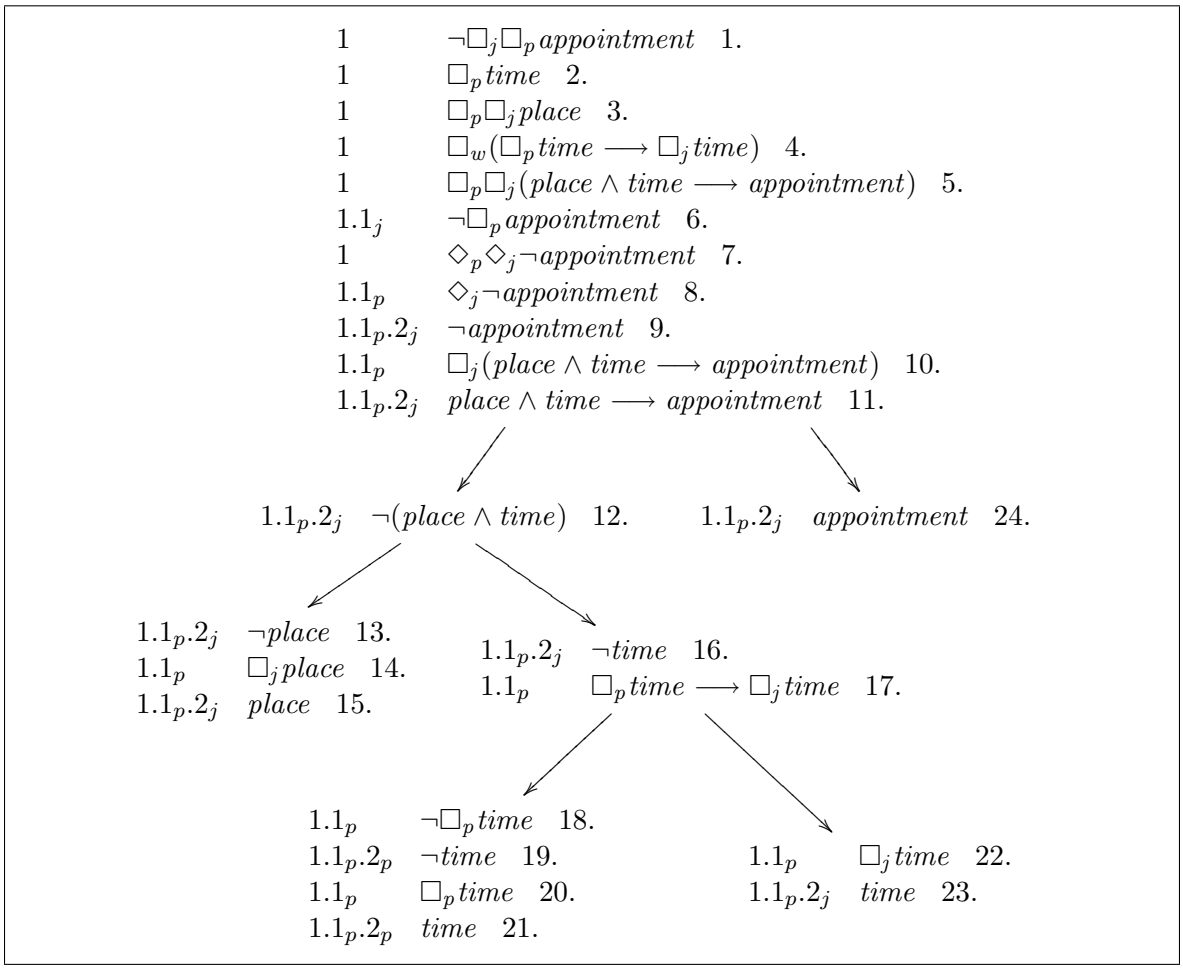

Figure 7: Proof of $\Box_j\Box_p appointment$

The next example, similar to the preceding one and also arising in learning applications, concludes this section.

Example 5.3. Let α be a type and $p : \alpha \to \Omega$ a predicate. Suppose that t is a basic term of type α and, by equational reasoning, it can be established that $\Box_j((p\ t) =_\Omega \top)$ is a theorem. I show by the tableau proof in Figure 9 that $\Box_j\forall x.(((=_\alpha\ t)\ x) \longrightarrow (p\ x))$ is a theorem, that is, $(=_\alpha\ t)$ is stronger than p.

An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 is from 1 by a possibility rule; 3 is from 2 by an existential rule; 4 and 5 are from 3 by a disjunctive rule; 6 is from 4 and 5 by the substitutivity rule; 7 is by lemma introduction; 8 is from 7 by a necessity rule; now the branch closes by 6 and 8.

6 Applications to Agents

This section studies the application of the logic to building multi-agent systems. In particular, it contains an extended example of the use of the logic for representing belief states of an agent that facilitates interaction between a user and the Internet. But before that I make

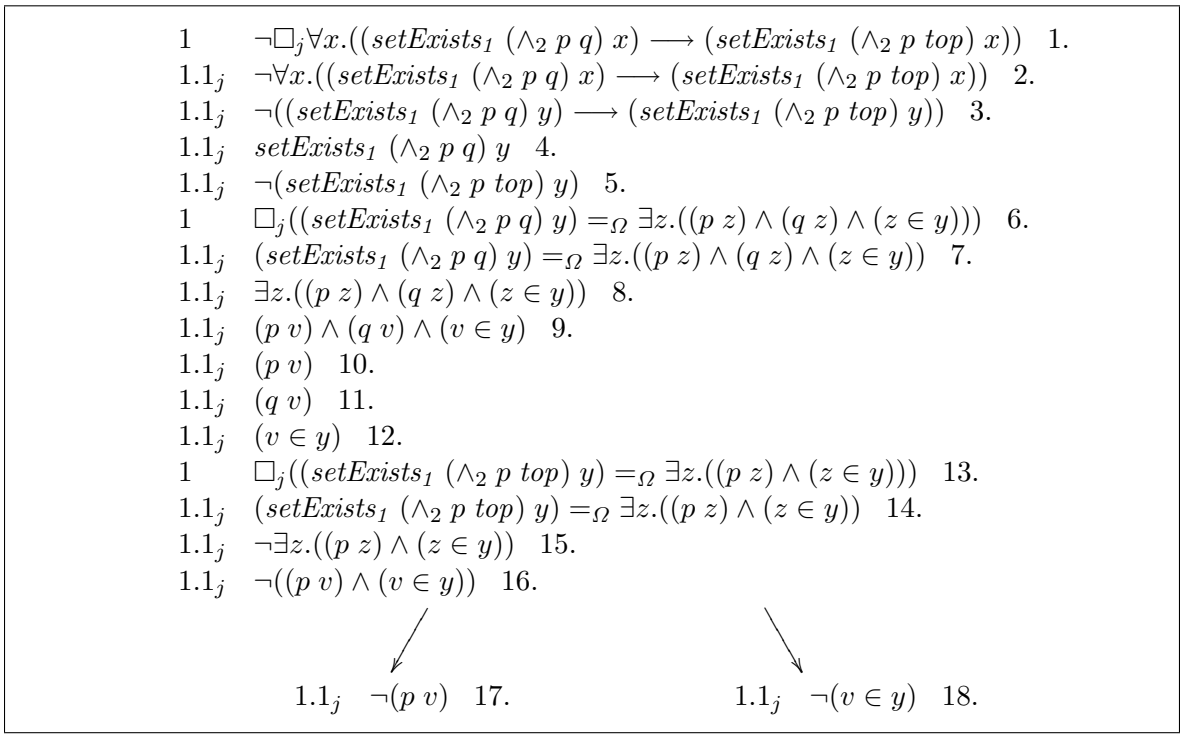

Figure 8: Proof of $\Box_j\forall x.((setExists_1\ (\wedge_2\ p\ q)\ x) \longrightarrow (setExists_1\ (\wedge_2\ p\ top)\ x))$

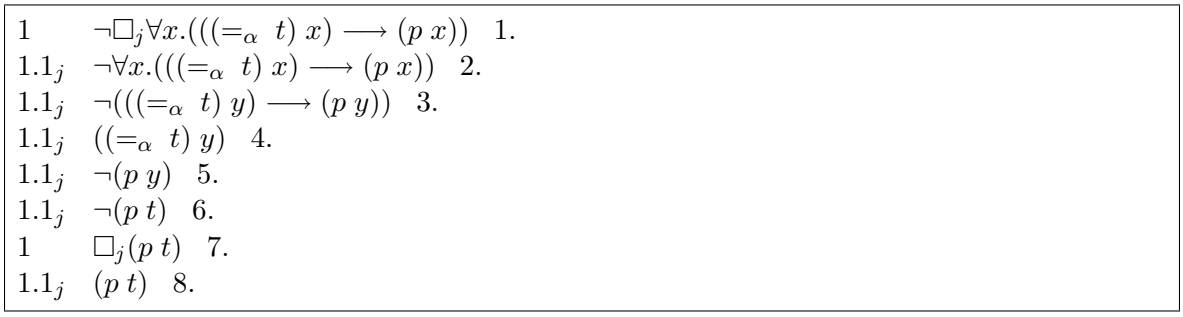

Figure 9: Proof of $\Box_j\forall x.(((=_\alpha\ t)\ x) \longrightarrow (p\ x))$

some remarks about the general approach taken here towards building agents, representation issues, and user agents for the Internet.

6.1 Approach to Agent Architectures

In general terms, I adopt the BDI approach to building agents. Thus agents are intention-based: having selected some intention to achieve, an agent reasons about its beliefs to select an action that will contribute towards achieving the intention. Here is a high-level view of the main loop of such an intention-based agent.

```
loop forever
  get percept
  revise beliefs
```

revise desires
revise intentions
select action
put action.

The aspect of this loop that is of most interest here is the action selection part. Actions are selected by a function *action* that has the signature

$$action : Beliefs \times Intention \rightarrow Action,$$

where *Beliefs* is the type of (the representation of) belief bases, *Intention* is the type of (the representation of) intentions, and *Action* is the type of (the representation of) actions. Thus with the intention selected in the previous step, the agent reasons about its beliefs (and this intention) to decide on an appropriate action.

There are many different ways the function *action* could use the agent's beliefs and the intention to make this choice, but one important one that I will concentrate on here is that of proving theorems using the belief base. Thus there is a function *prove* with signature

$$prove : Beliefs \times Term \rightarrow \Omega,$$

where *Term* is the type of terms in a meta-language representing terms in the belief state. The meaning of *prove* is that $(prove\ (b, f))$ is true iff the formula f is a theorem of the belief base b. The function *prove* implements the tableau theorem-proving system of Section 5. In Section 6.4, an example is given of the use of *prove* in the action selection function of a TV recommender.

6.2 Representation

Before moving on to the TV recommender itself, I make some remarks about the representation issue that was just touched on. The logical language in which the agent loop program (assumed to be a declarative program) is written includes a meta-language for representing beliefs, desires, intentions, and actions of the object language of the agent. Here are some details about a possible representation language for beliefs.

Let *Type* be the type of terms (in the meta-language) representing types (of the object language). Similarly, let *Constructor* be the type of terms representing type constructors. Suppose that the type constructors (in the object language) are $T_1, \ldots, T_m$. Representing these type constructors in the meta-language are m nullary data constructors, also named $T_1, \ldots, T_m$. Thus $T_1, ..., T_m : Constructor$. The following data constructors are needed.

$$ConstructorType : Constructor \times (List\ Type) \rightarrow Type$$
$$FunctionType : Type \times Type \rightarrow Type$$
$$ProductType : (List\ Type) \rightarrow Type.$$

The data constructors *ConstructorType*, *FunctionType* and *ProductType* are used to represent the forms of types given in the three parts of Definition 2.2. Now let $\ulcorner . \urcorner$ denote the function that maps a type (in the object language) to the term representing it (in the meta-language).

Then

$$\ulcorner T\ \alpha_1 \ldots \alpha_k \urcorner = (\mathit{ConstructorType}\ (T, [\ulcorner \alpha_1 \urcorner, \ldots, \ulcorner \alpha_k \urcorner]))$$
$$\ulcorner \alpha \to \beta \urcorner = (\mathit{FunctionType}\ (\ulcorner \alpha \urcorner, \ulcorner \beta \urcorner))$$
$$\ulcorner \alpha_1 \times \ldots \times \alpha_n \urcorner = (\mathit{ProductType}\ [\ulcorner \alpha_1 \urcorner, \ldots, \ulcorner \alpha_n \urcorner]).$$

Next the representation of terms is considered. Let *Term* be the type of terms (in the meta-language) representing terms (in the object language) and *Constant* the type of terms representing constants. Suppose that the constants (in the object language) are $C_1, \ldots, C_p$. Representing these constants in the meta-language are p nullary data constructors, also named $C_1, \ldots, C_p$. Thus $C_1, ..., C_p : \mathit{Constant}$.

The representation of variables is a little more complicated. For this, I assume there is a way of associating a unique natural number with each variable (in the object language). The following data constructors are needed.

$$\mathit{VariableTerm} : \mathit{Nat} \to \mathit{Term}$$
$$\mathit{ConstantTerm} : \mathit{Constant} \to \mathit{Term}$$
$$\mathit{AbstractionTerm} : \mathit{Term} \times \mathit{Term} \to \mathit{Term}$$
$$\mathit{ApplicationTerm} : \mathit{Term} \times \mathit{Term} \to \mathit{Term}$$
$$\mathit{TupleTerm} : (\mathit{List}\ \mathit{Term}) \to \mathit{Term}$$
$$\mathit{BoxTerm} : \mathit{Nat} \times \mathit{Term} \to \mathit{Term}.$$

Note that there is a data constructor corresponding to each part in Definition 2.4.

Now let $\ulcorner . \urcorner$ denote the function that maps a term (in the object language) to the term representing it (in the meta-language). Then

$\ulcorner x \urcorner = (\mathit{VariableTerm}\ n)$ (where x is the nth variable)
$$\ulcorner C \urcorner = (\mathit{ConstantTerm}\ C)$$
$$\ulcorner \lambda x.t \urcorner = (\mathit{AbstractionTerm}\ (\ulcorner x \urcorner, \ulcorner t \urcorner))$$
$$\ulcorner (s\ t) \urcorner = (\mathit{ApplicationTerm}\ (\ulcorner s \urcorner, \ulcorner t \urcorner))$$
$$\ulcorner (t_1, \ldots, t_n) \urcorner = (\mathit{TupleTerm}\ [\ulcorner t_1 \urcorner, \ldots, \ulcorner t_n \urcorner])$$
$$\ulcorner \Box_i t \urcorner = (\mathit{BoxTerm}\ (i, \ulcorner t \urcorner)).$$

Now signatures can be represented. For this, there is a type *Signature* that is the type of terms representing signatures and a data constructor

$$\mathit{Sig} : \mathit{Term} \times \mathit{Type} \to \mathit{Signature}.$$

Then

$$\ulcorner C : \alpha \urcorner = (\mathit{Sig}\ (\ulcorner C \urcorner, \ulcorner \alpha \urcorner)).$$

Finally, beliefs bases themselves can be represented. A belief base can be thought of a set of signatures (of the constants in the alphabet) and a set of formulas. Thus the representation of belief bases can be achieved using a type *Beliefs*, which is the type of terms representing belief bases, and a data constructor *BeliefBase* having signature

$$\mathit{BeliefBase} : \{\mathit{Signature}\} \times \{\mathit{Term}\} \to \mathit{Beliefs}.$$

Similar representation schemes can be set up for desires, intentions, and actions. Below, a little more detail is given about representing intentions and actions for a particular application.

6.3 User Agents for the Internet

Before moving on to the particular agent application studied in this section, I set the scene by describing briefly the general context of this application.

At The Australian National University, a team of researchers is working on a project funded by the Smart Internet Technology Cooperative Research Centre to apply machine learning techniques to build adaptive agents. The particular context is that of user agents that facilitate interaction between a user and the Internet. More particularly, the project is focussing at the moment on building an infotainment demonstrator. This multi-agent system combines a number of related functionalities concerning information search and entertainment. The agents that comprise the demonstrator include a TV recommender, a movie recommender, a news agent, a search agent, and a diary agent. Most of these are currently partially implemented, but none completely.

The aspect of adaptivity of major concern in the project is that of personalisation, that is, the adaptation of the agents' behaviour according to the interests and preferences of the user. There are a number of techniques that could be brought to bear to achieve effective personalisation; machine learning is probably the most important of these. More details of the infotainment demonstrator and its use of machine learning technology are given elsewhere [CGLN04]. Here it is enough to note that theorem proving plays a key role in the decision-making process of the agent and inside learning processes themselves. Furthermore, the full expressive power of the logic introduced here is needed in this application: multi-modalities are needed because it is a multi-agent system, the type system is heavily used in knowledge representation, and the higher-order facilities are needed to make basic terms available for knowledge representation and higher-order functions available for computation and deduction. The current implementation of the demonstrator does not make explicit use of theorem-proving capabilities; this paper is intended to provide the theoretical foundation for the next version of the demonstrator that will use a theorem prover.

6.4 TV Recommender

Now I consider the TV recommender itself. Its main objective is to make recommendations of TV programs that the user is likely to want to watch given the user's interests and preferences. To motivate the theorem-proving task presented later, I need to consider in a little more detail the function *action* for the TV recommender.

The function *action* has signature

$$action : Beliefs \times Intention \to Action.$$

Now what sort of intentions might the agent have? Well the fundamental thing it must be able to do is to decide whether or not to recommend a TV program to the user that is on a particular date, time, and channel. To represent the intention to do this, I introduce the data constructor

$$Evaluate : Date \times Time \times Channel \to Intention,$$

where *Date*, *Time* and *Channel* are types with the obvious meanings. Thus, if (D, T, C) is (the representation of) a particular date, time, and channel, then $(Evaluate\ (D, T, C))$ is (the representation of) the intention to evaluate whether the program uniquely determined by this triple should be recommended or not. Next, what about actions? To continue the

preceding theme, the agent will want to have available the actions of recommending and not recommending some particular program. To represent these actions, I introduce the data constructors

$$\begin{aligned}&\mathit{Recommend} : \mathit{Date} \times \mathit{Time} \times \mathit{Channel} \to \mathit{Action}\\ &\mathit{NotRecommend} : \mathit{Date} \times \mathit{Time} \times \mathit{Channel} \to \mathit{Action}.\end{aligned}$$

Thus $(\mathit{Recommend}\ (D, T, C))$ is (the representation of) the action to recommend to the user the program determined by the triple (D, T, C). Now suppose there is a predicate *program_recommendable* whose domain has type $\mathit{Date} \times \mathit{Time} \times \mathit{Channel}$ in the belief state of the TV agent. (Details below.) The meaning of this function is that if the program determined by some date, time, and channel is to be recommended to the user, then the value of the function is $\top$; otherwise, its value is $\bot$. Then one equation (amongst a number of other similar ones) in the definition of the function *action* is

$$\begin{aligned}&(\mathit{action}\ (b\ (\mathit{Evaluate}\ t))) =\\ &\quad \mathit{if}\ (\mathit{prove}\ (b,\\ &\qquad (\mathit{BoxTerm}\ (I, (\mathit{ApplicationTerm}\ ((\mathit{ConstantTerm}\ \mathit{program\_recommendable}), t))))))\\ &\quad \mathit{then}\ (\mathit{Recommend}\ t)\\ &\quad \mathit{else}\ (\mathit{NotRecommend}\ t),\end{aligned}$$

where I is the index of the TV agent. Here *prove* is the function with signature

$$\mathit{prove} : \mathit{Beliefs} \times \mathit{Term} \to \Omega,$$

mentioned earlier, that implements the tableau theorem prover. Thus whether or not the agent recommends a program determined by a particular triple (D, T, C) now depends on whether or not the formula

$$\Box_t(\mathit{program\_recommendable}\ (D, T, C))$$

is a theorem of the belief base of the TV agent. (Note that the belief modality for the TV agent is denoted by $\Box_t$.)

With this motivation, I now discuss this theorem proving task in considerable detail. To begin with, three nullary types are needed: *Channel*, *Genre*, and *Classification*. Here are the data constructors for these types.

$$\begin{aligned}&\mathit{ABC}, \mathit{Prime}, \mathit{WIN}, \mathit{Ten}, \mathit{SBS} : \mathit{Channel}\\ &\mathit{Animation}, \mathit{Arts\_and\_Culture}, \mathit{Business\_and\_Finance}, \mathit{Cartoon}, \mathit{Children}, \mathit{Comedy},\\ &\quad \mathit{Current\_Affairs}, \mathit{Documentary}, \mathit{Drama}, \mathit{Education}, \mathit{Entertainment}, \mathit{Family},\\ &\quad \mathit{Game\_Show}, \mathit{Infotainment}, \mathit{Lifestyle}, \mathit{Music}, \mathit{News}, \mathit{Real\_Life}, \mathit{Religion},\\ &\quad \mathit{Science\_and\_Technology}, \mathit{Sci\_Fi}, \mathit{Shopping}, \mathit{Sitcom}, \mathit{Soap\_Opera}, \mathit{Sport}, \mathit{Talk\_Show},\\ &\quad \mathit{Thriller}, \mathit{Variety}, \mathit{Weather}, \mathit{NA} : \mathit{Genre}\\ &C, G, M, \mathit{MA}, P, \mathit{PG}, \mathit{NA} : \mathit{Classification}.\end{aligned}$$

I introduce the following type synonyms.

$Date = Day \times Month \times Year$
$Time = Hour \times Minute$
$Title = String$
$Duration = Minute$
$Synopsis = String$
$Program = Title \times Duration \times Genre \times Classification \times Synopsis$
$Year = Nat$
$Month = Nat$
$Day = Nat$
$Hour = Nat$
$Minute = Nat.$

Some relevant function signatures are as follows.

$tv\_guide : Date \times Time \times Channel \to Program$
$program\_recommendable : Date \times Time \times Channel \to \Omega$
$user\_likes\_tv\_program : Program \to \Omega$
$user\_tv\_time\_acceptable : Date \times Time \times Time \to \Omega$
$user\_diary\_free : Date \times Time \times Time \to \Omega$
$proj_{Duration} : Program \to Duration$
$proj_{Hour} : Time \to Hour.$

Here is part of the belief base of the TV agent.

$$\Box_t \, \forall_{Date} d. \, \forall_{Time} t. \, \forall_{Channel} c.$$
$$((user\_likes\_tv\_program \; (tv\_guide \; (d,t,c))) \wedge$$
$$(user\_tv\_time\_acceptable \; (d,t,(add \; (t,(proj_{Duration} \; (tv\_guide \; (d,t,c))))))) \wedge$$
$$(user\_diary\_free \; (d,t,(add \; (t,(proj_{Duration} \; (tv\_guide \; (d,t,c)))))))$$
$$\to (program\_recommendable \; (d,t,c)))$$

$$\Box_t \, \forall_{Date} d. \, \forall_{Time} t. \, \forall_{Time} t'.$$
$$((weekday \; d) \wedge$$
$$((proj_{Hour} \; t) \geq 20) \wedge$$
$$((proj_{Hour} \; t') \leq 23)$$
$$\to (user\_tv\_time\_acceptable \; (d,t,t')))$$

$$\Box_t \, \forall_{Date} d. \, \forall_{Time} t. \, \forall_{Time} t'.$$
$$(\neg(weekday \; d) \wedge$$
$$((proj_{Hour} \; t) \geq 12) \wedge$$
$$((proj_{Hour} \; t') \leq 25)$$
$$\to (user\_tv\_time\_acceptable \; (d,t,t')))$$

$$\Box_t \forall_{Title} t.\ \forall_{Duration} d.\ \forall_{Genre} g.\ \forall_{Classification} c.\ \forall_{Synopsis} s.$$
$$((proj_{Duration}\ (t,d,g,c,s)) =_{Duration}\ d)$$

$$\Box_t \forall_{Hour} h.\ \forall_{Minute} m.$$
$$((proj_{Hour}\ (h,m)) =_{Hour}\ h).$$

Also included in this belief base is the TV guide that consists of (thousands of) facts like the following one.

$$\Box_t\ ((tv\_guide\ ((20,7,2004),(20,30),ABC)) =_{Program}$$
$$(\text{“}The\ Bill\text{”}, 50, Drama, M,$$
$$\text{“}Sun\ Hill\ continues\ to\ work\ at\ breaking\ the\ people\ smuggling\ operation\text{”})).$$

Note that the term on the right hand side of this equation is a basic term (representing the individual that, in this case, is a TV program).

The belief base of the TV agent also includes the function *user_likes_tv_program* that has a definition of the form

$$\Box_t\ ((user\_likes\_tv\_program\ p) =_\Omega\ if \ldots then \ldots else \ldots).$$

The function *user_likes_tv_program* is a learned function that helps to personalise the TV agent for a particular user. The decision-list learning system ALKEMY [Ng04] is used to learn the definition of this function from training examples provided by the user. The function definition is an equation and when using this function in theorem proving the specialised equational reasoning introduced earlier is employed. Typically, one wants to know whether or not a formula of the form

$$(user\_likes\_tv\_program\ P),$$

where P is a particular program, is a theorem of the belief base or not. To determine this, a computation with this as the first formula is carried out. This computation will end in $\top$ or $\bot$. Thus, by Proposition 5.6, either

$$\Box_t((user\_likes\_tv\_program\ P) =_\Omega \top)$$

or

$$\Box_t((user\_likes\_tv\_program\ P) =_\Omega \bot)$$

is a consequence of some local assumptions. Then, by Proposition 5.7, either

$$\Box_t(user\_likes\_tv\_program\ P)$$

or

$$\Box_t\neg(user\_likes\_tv\_program\ P)$$

is a consequence of these local assumptions. Thus one of these formulas can be introduced into the main proof by lemma introduction. This is illustrated below.

The function *user_likes_tv_program* depends upon a number of other functions, some of which are obtained directly from the user. A good example of this is the function *genre*. To define this, I introduce the type synonym

$$Preference = Int,$$

where it is understood that only numbers in $\{-2, -1, 0, 1, 2\}$ are to be used as constants of type *Preference*. Then

$$genre : Genre \rightarrow Preference$$

is the function that maps each genre into an integer in the range -2 to 2, depending on how strong a preference the user has for that particular genre. Thus a typical definition of *genre* could be as follows.

$$\Box_t(genre =_{Genre \rightarrow Preference} \lambda x.if\ x =_{Genre} Comedy\ then\ 1\ else\ if\ x =_{Genre} Drama\ then\ -1\ else\ 0).$$

This equation states that the user has a modest liking for comedy, a modest dislike for drama. and is neutral about the other genres. The information in this equation is obtained by an initial questionnaire completed by the user and by belief update, if the user later changes his/her preferences. Note carefully that the right hand side of the above equation is a basic term, in fact, a basic abstraction. One can think of it as a finite lookup table, for which the default term is 0. Thus whatever the belief revision/update algorithm, it must be able to deal with the internal details of basic terms, including basic abstractions. For example, if the user later changes his/her preferences to a strong liking (2) for comedy, then the belief update/revision algorithm should update the above equation to

$$\Box_t(genre =_{Genre \rightarrow Preference} \lambda x.if\ x =_{Genre} Comedy\ then\ 2\ else\ if\ x =_{Genre} Drama\ then\ -1\ else\ 0).$$

The proof below also needs the function *user_diary_free* for which there are formulas in the belief base of the diary agent of the form

$$\Box_d\ \forall_{Date} d.\ \forall_{Time} t.\ \forall_{Time} t'.(\cdots \longrightarrow (user\_diary\_free\ (d, t, t'))).$$

Here $\Box_d$ is the belief modality for the diary agent. In the proof below, theorems of the belief base of the diary agent are introduced as lemmas. The details of how these lemmas are proved are omitted. These lemmas will have the form either

$$\Box_d(user\_diary\_free\ (D, T_1, T_2))$$

or

$$\Box_d\neg(user\_diary\_free\ (D, T_1, T_2)),$$

where D is a particular date, and T_1 and T_2 are particular times.

To make use of these lemmas in the main proof, there is an interaction axiom $\Box_d\varphi \longrightarrow \Box_t\varphi$, which states that everything believed by the diary agent is also believed by the TV agent. Corresponding to this axiom are the tableau rules in Figure 10.

I_{dt} rules: $\dfrac{\sigma\ \Box_d\varphi}{\sigma.n_t^\dagger\ \varphi}$ $\dfrac{\sigma\ \neg\Diamond_d\varphi}{\sigma.n_t^\dagger\ \neg\varphi}$ $\dfrac{\sigma.n_t\ \varphi}{\sigma\ \Diamond_d\varphi}$

I_{ct} rules: $\dfrac{\sigma\ \Box\varphi}{\sigma.n_t^\dagger\ \varphi}$ $\dfrac{\sigma\ \neg\Diamond\ \varphi}{\sigma.n_t^\dagger\ \neg\varphi}$ $\dfrac{\sigma.n_t\ \varphi}{\sigma\ \Diamond\ \varphi}$

(†) prefix already occurs on the branch

Figure 10: Interaction rules for the diary and TV agents, and common beliefs

In addition, there is a common belief base of basic information with a corresponding belief modality $\Box$ and interaction axiom $\Box\varphi \longrightarrow \Box_t\varphi$, which states that common beliefs are also believed by the TV agent. Corresponding to this axiom are the tableau rules in Figure 10.

Now suppose that one wants to discover whether

$$\Box_t\ (\mathit{program\_recommendable}\ ((20,7,2004),(20,30),ABC)))$$

is a theorem of the belief base of the TV agent (and the diary agent and the common beliefs). A proof of this formula is given in Figure 12 and subsequent figures below. The overall structure of the proof is given in Figure 11.

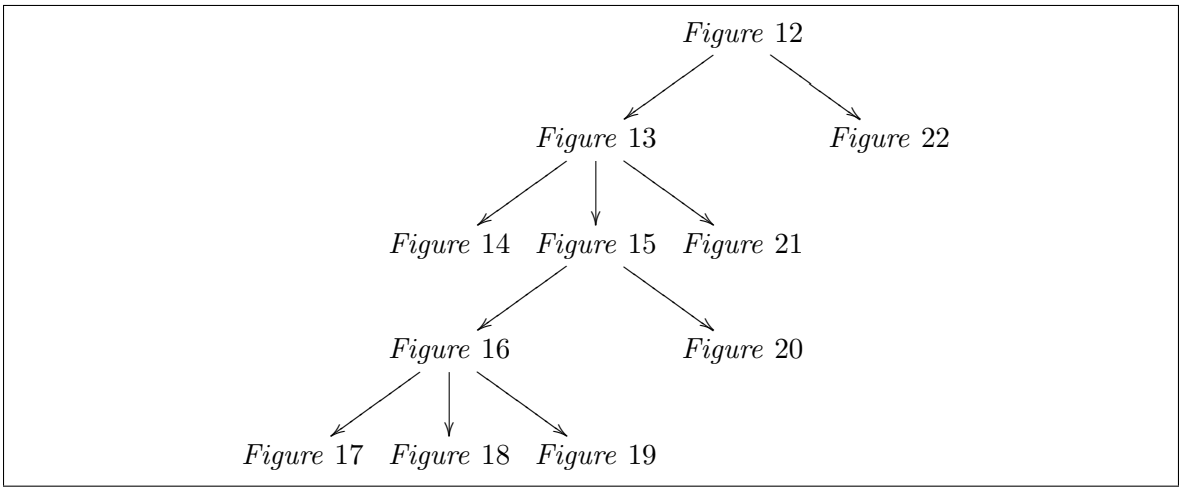

Figure 11: Overall structure of the proof

In Figure 12, item 1 is the negation of the formula to be proved; 2 to 7 are local assumptions; 8 is from 1 by a possibility rule; 9 is from 7 by a necessity rule; 10 is from 2 by a necessity rule; 11 is from 10 by a universal rule.

In Figure 13, item 12 is from 11 by a disjunctive rule.

In Figure 14, item 13 is from 12 by a disjunctive rule; 14 is from 9 and 13 by the substitutivity rule; 15 is by lemma introduction from the TV agent belief base; 16 is from 15 by a necessity rule. Now this branch closes because of 14 and 16.

In Figure 15, item 17 is from 12 by a disjunctive rule; 18 is from 9 and 17 by the substitutivity rule; 19 is from 5 by a necessity rule; 20 is from 19 by a universal rule; 21 is from 18 and 20 by the substitutivity rule; 22 is by lemma introduction from the common belief base;

1 $\neg\Box_t\ (program\_recommendable\ ((20,7,2004),(20,30),ABC))$ 1.

1 $\Box_t\ \forall_{Date}d.\ \forall_{Time}t.\ \forall_{Channel}c.$
$((user\_likes\_tv\_program\ (tv\_guide\ (d,t,c)))\ \wedge$
$(user\_tv\_time\_acceptable\ (d,t,(add\ (t,(proj_{Duration}\ (tv\_guide\ (d,t,c)))))))\ \wedge$
$(user\_diary\_free\ (d,t,(add\ (t,(proj_{Duration}\ (tv\_guide\ (d,t,c)))))))$
$\rightarrow (program\_recommendable\ (d,t,c)))$ 2.

1 $\Box_t\ \forall_{Date}d.\ \forall_{Time}t.\ \forall_{Time}t'.$
$((weekday\ d)\wedge((proj_{Hour}\ t)\geq 20)\wedge((proj_{Hour}\ t')\leq 23)$
$\rightarrow (user\_tv\_time\_acceptable\ (d,t,t')))$ 3.

1 $\Box_t\ \forall_{Date}d.\ \forall_{Time}t.\ \forall_{Time}t'.$
$(\neg(weekday\ d)\wedge((proj_{Hour}\ t)\geq 12)\wedge((proj_{Hour}\ t')\leq 25)$
$\rightarrow (user\_tv\_time\_acceptable\ (d,t,t')))$ 4.

1 $\Box_t\ \forall_{Title}t.\ \forall_{Duration}d.\ \forall_{Genre}g.\ \forall_{Classification}c.\ \forall_{Synopsis}s.$
$((proj_{Duration}\ (t,d,g,c,s))=_{Duration}\ d)$ 5.

1 $\Box_t\ \forall_{Hour}h.\ \forall_{Minute}m.$
$((proj_{Hour}\ (h,m))=_{Hour}\ h)$ 6.

1 $\Box_t\ ((tv\_guide\ ((20,7,2004),(20,30),ABC))=_{Program}$
$(\text{“}The\ Bill\text{”},50,Drama,M,\text{“}Sun\ Hill\ continues\ to\ work\ at\ breaking$
$the\ people\ smuggling\ operation\text{”}))$ 7.

1.1$_t$ $\neg(program\_recommendable\ ((20,7,2004),(20,30),ABC))$ 8.

1.1$_t$ $(tv\_guide\ ((20,7,2004),(20,30),ABC))=_{Program}$
$(\text{“}The\ Bill\text{”},50,Drama,M,\text{“}Sun\ Hill\ continues\ to\ work\ at\ breaking$
$the\ people\ smuggling\ operation\text{”})$ 9.

1.1$_t$ $\forall_{Date}d.\ \forall_{Time}t.\ \forall_{Channel}c.$
$((user\_likes\_tv\_program\ (tv\_guide\ (d,t,c)))\ \wedge$
$(user\_tv\_time\_acceptable\ (d,t,(add\ (t,(proj_{Duration}\ (tv\_guide\ (d,t,c)))))))\ \wedge$
$(user\_diary\_free\ (d,t,(add\ (t,(proj_{Duration}\ (tv\_guide\ (d,t,c)))))))$
$\rightarrow (program\_recommendable\ (d,t,c)))$ 10.

1.1$_t$ $((user\_likes\_tv\_program\ (tv\_guide\ ((20,7,2004),(20,30),ABC)))\ \wedge$
$(user\_tv\_time\_acceptable\ ((20,7,2004),(20,30),$
$(add\ ((20,30),(proj_{Duration}\ (tv\_guide\ ((20,7,2004),(20,30),ABC)))))))\ \wedge$
$(user\_diary\_free\ ((20,7,2004),(20,30),$
$(add\ ((20,30),(proj_{Duration}\ (tv\_guide\ ((20,7,2004),(20,30),ABC)))))))$
$\rightarrow (program\_recommendable\ ((20,7,2004),(20,30),ABC)))$ 11.

Figure 12: Start of proof of $\Box_t\ (program\_recommendable\ ((20,7,2004),(20,30),ABC))$ (Go next to Figures 13 and 22)

1.1_t $\neg((user\_likes\_tv\_program\ (tv\_guide\ ((20,7,2004),(20,30),ABC))) \wedge$
$(user\_tv\_time\_acceptable\ ((20,7,2004),(20,30),$
$(add\ ((20,30),(proj_{Duration}\ (tv\_guide\ ((20,7,2004),(20,30),ABC)))))))) \wedge$
$(user\_diary\_free\ ((20,7,2004),(20,30),$
$(add\ ((20,30),(proj_{Duration}\ (tv\_guide\ ((20,7,2004),(20,30),ABC))))))))$ 12.

Figure 13: Part of proof of $\Box_t\ (program\_recommendable\ ((20,7,2004),(20,30),ABC))$ (Go next to Figures 14, 15 and 21)

1.1_t $\neg(user\_likes\_tv\_program\ (tv\_guide\ ((20,7,2004),(20,30),ABC)))$ 13.

1.1_t $\neg(user\_likes\_tv\_program\ (\text{“}The\ Bill\text{”},50,Drama,M,\text{“}Sun\ Hill\ continues$
$to\ work\ at\ breaking\ the\ people\ smuggling\ `operation\text{”}))$ 14.

1 $\Box_t(user\_likes\_tv\_program\ (\text{“}The\ Bill\text{”},50,Drama,M,\text{“}Sun\ Hill\ continues$
$to\ work\ at\ breaking\ the\ people\ smuggling\ operation\text{”}))$ 15.

1.1_t $(user\_likes\_tv\_program\ (\text{“}The\ Bill\text{”},50,Drama,M,\text{“}Sun\ Hill\ continues$
$to\ work\ at\ breaking\ the\ people\ smuggling\ operation\text{”}))$ 16.

Figure 14: Part of proof of $\Box_t\ (program\_recommendable\ ((20,7,2004),(20,30),ABC))$

23 is from 22 by an interaction rule; 24 is from 21 and 23 by the substitutivity rule; 25 is from 3 by a necessity rule; 26 is from 25 by a universal rule.

In Figure 16, item 27 is from 26 by a disjunctive rule.

In Figure 17, item 28 is from 27 by a disjunctive rule; 29 is by lemma introduction from the common belief base; 30 is from 29 by an interaction rule. Now this branch closes because of 28 and 30.

In Figure 18, item 31 is from 27 by a disjunctive rule; 32 is from 6 by a necessity rule; 33 is from 32 by a universal rule; 34 is from 31 and 33 by the substitutivity rule; 35 is by lemma introduction from the common belief base; 36 is from 35 by an interaction rule. Now this branch closes because of 34 and 36.

In Figure 19, item 37 is from 27 by a disjunctive rule; 38 is from 6 by a necessity rule; 39 is from 38 by a universal rule; 40 is from 37 and 39 by the substitutivity rule; 41 is by lemma introduction from the common belief base; 42 is from 41 by an interaction rule. Now this branch closes because of 40 and 42.

In Figure 20, item 43 is from 26 by a disjunctive rule. Now this branch closes because of 24 and 43.

In Figure 21, item 44 is from 12 by a disjunctive rule; 45 is from 9 and 44 by the substitutivity rule; 46 is from 5 by a necessity rule; 47 is from 46 by a universal rule; 48 is from 45 and 47 by the substitutivity rule; 49 is by lemma introduction from the common belief base; 50 is from 49 by an interaction rule; 51 is from 48 and 50 by the substitutivity rule; 52 is by lemma introduction from the diary belief base; 53 is from 52 by an interaction rule. Now this branch closes by 51 and 53.

In Figure 22, item 54 is from 11 by a disjunctive rule. Now this branch closes by 8 and 54. The proof is now complete.

1.1_t $\neg(user\_tv\_time\_acceptable\ ((20,7,2004),(20,30),$
$(add\ ((20,30),(proj_{Duration}\ (tv\_guide\ ((20,7,2004),(20,30),ABC)))))))$ 17.

1.1_t $\neg(user\_tv\_time\_acceptable\ ((20,7,2004),(20,30),$
$(add\ ((20,30),(proj_{Duration}\ (\text{“}The\ Bill\text{”},50,Drama,M,$
$\text{“}Sun\ Hill\ continues\ to\ work\ at\ breaking\ the\ people\ smuggling\ operation\text{”})))))))$ 18.

1.1_t $\forall_{Title}t.\ \forall_{Duration}d.\ \forall_{Genre}g.\ \forall_{Classification}c.\ \forall_{Synopsis}s.$
$((proj_{Duration}\ (t,d,g,c,s) =_{Duration} d))$ 19.

1.1_t $(proj_{Duration}\ (\text{“}The\ Bill\text{”},50,Drama,M,\text{“}Sun\ Hill\ continues\ to\ work$
$at\ breaking\ the\ people\ smuggling\ operation\text{”})) =_{Duration} 50$ 20.

1.1_t $\neg(user\_tv\_time\_acceptable\ ((20,7,2004),(20,30),(add\ ((20,30),50))))$ 21.

1 $\Box((add\ ((20,30),50)) =_{Time} (21,20))$ 22.

1.1_t $(add\ ((20,30),50)) =_{Time} (21,20)$ 23.

1.1_t $\neg(user\_tv\_time\_acceptable\ ((20,7,2004),(20,30),(21,20)))$ 24.

1.1_t $\forall_{Date}d.\ \forall_{Time}t.\ \forall_{Time}t'.$
$((weekday\ d) \wedge ((proj_{Hour}\ t) \geq 20) \wedge ((proj_{Hour}\ t') \leq 23)$
$\rightarrow (user\_tv\_time\_acceptable\ (d,t,t')))$ 25.

1.1_t $((weekday\ (20,7,2004)) \wedge ((proj_{Hour}\ (20,30)) \geq 20) \wedge ((proj_{Hour}\ (21,20)) \leq 23)$
$\rightarrow (user\_tv\_time\_acceptable\ ((20,7,2004),(20,30),(21,20))))$ 26.

Figure 15: Part of proof of $\Box_t$ $(program\_recommendable\ ((20,7,2004),(20,30),ABC))$ (Go next to Figures 16 and 20)

1.1_t $\neg((weekday\ (20,7,2004)) \wedge ((proj_{Hour}\ (20,30)) \geq 20) \wedge ((proj_{Hour}\ (21,20)) \leq 23))$ 27.

Figure 16: Part of proof of $\Box_t$ $(program\_recommendable\ ((20,7,2004),(20,30),ABC))$ (Go next to Figures 17, 18 and 19)

1.1_t $\neg(weekday\ (20,7,2004))$ 28.

1 $\Box(weekday\ (20,7,2004))$ 29.

1.1_t $(weekday\ (20,7,2004))$ 30.

Figure 17: Part of proof of $\Box_t$ $(program\_recommendable\ ((20,7,2004),(20,30),ABC))$

1.1_t	$\neg((proj_{Hour}\ (20,30)) \geq 20)$	31.
1.1_t	$\forall_{Hour}h.\ \forall_{Minute}m.((proj_{Hour}\ (h,m)) =_{Hour} h)$	32.
1.1_t	$(proj_{Hour}\ (20,30)) =_{Hour} 20$	33.
1.1_t	$\neg(20 \geq 20)$	34.
1	$\Box(20 \geq 20)$	35.
1.1_t	$20 \geq 20$	36.

Figure 18: Part of proof of $\Box_t$ $(program\_recommendable\ ((20,7,2004),(20,30),ABC))$

1.1_t	$\neg((proj_{Hour}\ (21,20)) \leq 23))$	37.
1.1_t	$\forall_{Hour}h.\ \forall_{Minute}m.((proj_{Hour}\ (h,m)) =_{Hour} h)$	38.
1.1_t	$(proj_{Hour}\ (21,20)) =_{Hour} 21$	39.
1.1_t	$\neg(21 \leq 23)$	40.
1	$\Box(21 \leq 23)$	41.
1.1_t	$21 \leq 23$	42.

Figure 19: Part of proof of $\Box_t$ $(program\_recommendable\ ((20,7,2004),(20,30),ABC))$

1.1_t	$(user\_tv\_time\_acceptable\ ((20,7,2004),(20,30),(21,20)))$	43.

Figure 20: Part of proof of $\Box_t$ $(program\_recommendable\ ((20,7,2004),(20,30),ABC))$

1.1_t $\neg(user\_diary\_free\ ((20,7,2004),(20,30),$
$(add\ ((20,30),(proj_{Duration}\ (tv\_guide\ ((20,7,2004),(20,30),ABC)))))))$ 44.

1.1_t $\neg(user\_diary\_free\ ((20,7,2004),(20,30),$
$(add\ ((20,30),(proj_{Duration}\ (\text{“}The\ Bill\text{”},50,Drama,M,$
$\text{“}Sun\ Hill\ continues\ to\ work\ at\ breaking\ the\ people\ smuggling,operation\text{”})))))$ 45.

1.1_t $\forall_{Title}t.\ \forall_{Duration}d.\ \forall_{Genre}g.\ \forall_{Classification}c.\ \forall_{Synopsis}s.$
$((proj_{Duration}\ (t,d,g,c,s) =_{Duration}\ d))$ 46.

1.1_t $(proj_{Duration}\ (\text{“}The\ Bill\text{”},50,Drama,M,\text{“}Sun\ Hill\ continues\ to\ work$
$at\ breaking\ the\ people\ smuggling\ operation\text{”})) =_{Duration} 50$ 47.

1.1_t $\neg(user\_diary\_free\ ((20,7,2004),(20,30),(add\ ((20,30),50))))$ 48.

1 $\Box((add\ ((20,30),50)) =_{Time}\ (21,20))$ 49.

1.1_t $(add\ ((20,30),50)) =_{Time}\ (21,20)$ 50.

1.1_t $\neg(user\_diary\_free\ ((20,7,2004),(20,30),(21,20)))$ 51.

1 $\Box_d(user\_diary\_free\ ((20,7,2004),(20,30),(21,20)))$ 52.

1.1_t $(user\_diary\_free\ ((20,7,2004),(20,30),(21,20)))$ 53.

Figure 21: Part of proof of $\Box_t\ (program\_recommendable\ ((20,7,2004),(20,30),ABC))$

1.1_t $(program\_recommendable\ ((20,7,2004),(20,30),ABC))$ 54.

Figure 22: Part of proof of $\Box_t\ (program\_recommendable\ ((20,7,2004),(20,30),ABC))$

Next a brief illustration of equational reasoning is given. (For much more on this, see [Llo03].) Consider the function *add* that has the signature

$$add : Time \times Duration \rightarrow Time.$$

The definition of this function is

$$\Box((add\ ((h, m), d)) =_{Time} ((60 \times h + m + d)\ div\ 60, (60 \times h + m + d)\ mod\ 60)).$$

Suppose now we want to compute the value of $(add\ ((20, 30), 50))$. The computation proceeds as follows.

$$(add\ ((20, 30), 50))$$
$$((60 \times 20 + 30 + 50)\ div\ 60, (60 \times 20 + 30 + 50)\ mod\ 60)$$
$$\vdots$$
$$(1280\ div\ 60, 1280\ mod\ 60)$$
$$\vdots$$
$$(21, 20).$$

Now Proposition 5.6 shows that

$$\Box((add\ ((20, 30), 50)) =_{Time} (21, 20))$$

is a consequence of some local assumptions, including the definition of the function *add* and some other arithmetic functions in the common belief base. This formula can now be introduced into the proof by lemma introduction, as is done in item 49 in Figure 21.

I conclude this section with some remarks about the efficiency and decidability of theorem proving in the logic. It is well known that the satisfiability problem of the logics $S5_m$ and $KD45_m$ ($m \geq 2$) is PSPACE-complete, so the corresponding validity problems are also PSPACE-complete [FHMV95]. Church proved that the validity problem of (classical) first-order logic is undecidable. Furthermore, Gödel proved that the validity problem of (classical) second-order logic is not semi-decidable. Gödel's result depends upon the use of standard models. It was Henkin who realised that by extending the class of models to so-called Henkin (also called general) models the collection of valid formulas could be reduced and a completeness result could be obtained for higher-order (and therefore second-order) logic. As a consequence the validity problem became merely undecidable (instead of not semi-decidable). Furthermore, the compactness and Löwenheim-Skolem theorems can be proved in this setting [Fit02]. Note that, at this point, Lindström's (first) theorem [EFT84, Ch. XII] can be applied to show that higher-order logic (with the Henkin semantics) is essentially just a variant of first-order logic. In spite of this result, something important has been gained: instead of being forced to express certain things awkwardly in first-order logic, the greater expressive power of higher-order logic can be used. This paper contains plenty of illustrations of the advantages of the greater expressive power of higher-order logic.

In the modal higher-order case, Fitting notes that completeness still holds. (This is left a "huge exercise" in [Fit02].) The logic here and the one in [Fit02] are so close that I presume a completeness theorem (and compactness and Löwenheim-Skolem theorems) can also be proved for the logic of this paper. This is left as important future work.

However, I am advocating a move from the decidable validity problem of modal propositional logic to the undecidable validity problem of modal higher-order logic, so particular care needs to be taken when implementing a modal higher-order theorem prover to mitigate against this difficulty as much as possible. There is plenty of evidence that useful things can be done even in such a rich logic. First, the equational reasoning mechanism is only slightly more complicated than the operational mechanism of a typical functional programming language such as Haskell, so this provides us with something practically useful for the equational reasoning part. For the tableau rules themselves, the source of the difficulty is fairly obvious: how do we know which substitution to choose in an application of a universal rule? To make some progress on this, it seems necessary to make restrictions on the kind of formulas that can appear in a belief state. In fact, the above extended example already indicates a good direction.

Note that the formula about the predicate *program_recommendable* is a Horn clause. How do we know which instance of it should be chosen in step 11? Already by then the formula

$$\neg(\mathit{program\_recommendable}\ ((20,7,2004),(20,30),ABC))$$

is in the tableau. Looking now at the head $(\mathit{program\_recommendable}\ (d,t,c))$ of the Horn clause for *program_recommendable*, it is clear that the appropriate substitution is $\{d/(20,7,2004), t/(20,30), c/ABC\}$. From this, step 11 is obtained. Subsequently the formulas in steps 8 and 11 are handled by an inference mechanism in the tableau proof that exactly mimics resolution. In fact, this inference mechanism is even more specialised in that it is just the SLD-resolution of Prolog. The formulas about the predicate *user_tv_time_acceptable* are handled similarly. These formulas are all program clauses, in the logic programming terminology, and thus can be handled efficiently by the Prolog mechanism (except negation as failure is not used to handle negative literals in the body of the program clause – instead negation is handled classically). Since program clauses capture much useful knowledge that is in implicational form, one can expect this straightforward mechanism to cover many uses of the universal rules. Of course, all this (Haskell in the equational part and Prolog in the tableau part) does not make the logic decidable; but one can at least see that many practical problems will have useful solutions because of the syntactic form of the knowledge/beliefs involved.

7 Conclusion

This paper has introduced a multi-modal, typed, higher-order logic that is well suited to agent and other applications. A detailed account of the syntax and semantics of the logic, together with a tableau system for proving theorems, was given. An extended example showed the application of the logic to the representation of belief states in multi-agent systems. Heavy use of the type system, modalities, and higher-order features of the logic was made in this example and would also be in any practical agent applications.

However, there is still much that needs to be done. First, it would be desirable to have a completeness result for the tableau system. This will certainly involve extending the definition of an interpretation used here because of the need to employ Henkin models (rather than standard ones) to obtain a completeness result. Second, the tableau system needs to be implemented and then deployed in some significant agent systems. Third, an essential facility of the logic for agent applications is a belief revision algorithm. A promising approach for

this that needs to be developed is a belief acquisition algorithm based on decision lists that unifies learning and belief revision/update.

Acknowledgments

This research was partially supported by the Smart Internet Technology Cooperative Research Centre through the project entitled "Machine Learning for the Smart Personal Assistant". Many thanks to Robert Bridle, Joshua Cole, Eric McCreath, Vineet Nair, and Kee Siong Ng for numerous helpful discussions. I am particularly grateful to Rajeev Goré for expert advice on modal logic.

References

[Bal98] M. Baldoni. *Normal Multimodal Logics: Automatic Deduction and Logic Programming Extension*. PhD thesis, Università degli Studi di Torino, 1998.

[CGLN04] J. Cole, M. Gray, J.W. Lloyd, and K.S. Ng. Personalisation for smart personal assistants. In *Annual Partner Conference, Smart Internet Technology Cooperative Research Centre*, 2004.

[EFT84] H.D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Springer-Verlag, 1984.

[FHMV95] R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[Fit02] M. Fitting. *Types, Tableaus, and Gödel's God*. Kluwer Academic Publishers, 2002.

[FM98] M. Fitting and R.L. Mendelsohn. *First-order Modal Logic*. Kluwer Academic Publishers, 1998.

[Gal75] D. Gallin. *Intensional and Higher-order Modal Logic*. North-Holland, 1975.

[He] M. Hanus (ed.). Curry: An integrated functional logic language. `http://www.informatik.uni-kiel.de/~curry`.

[Je] S. Peyton Jones and J. Hughes (editors). Haskell98: A non-strict purely functional language. `http://haskell.org/`.

[Llo03] J.W. Lloyd. *Logic for Learning*. Cognitive Technologies. Springer, 2003.

[Ng04] K.S. Ng. Alkemy: A learning system based on an expressive knowledge representation formalism. submitted for publication, 2004.

[Woo00] M. Wooldridge. *Reasoning about Rational Agents*. Bradford Books, 2000.

A Appendix

This appendix contains two principles of inductive construction on well-founded sets.

A *strict partial order* on a set A is a binary relation $<$ on A such that, for each $a, b, c \in A$, $a \not< a$ (irreflexivity), $a < b$ implies $b \not< a$ (asymmetry), and $a < b$ and $b < c$ implies $a < c$ (transitivity).

Suppose now that $<$ is a strict partial order on a set A. Then $<$ is a *well-founded order* if there is no infinite sequence $a_1, a_2, \ldots$ such that $a_{i+1} < a_i$, for $i \in \mathbb{Z}^+$.

By way of an example, if Σ any alphabet, then the set Σ^* of all strings over Σ with the substring relation $\prec$ (that is, $s_1 \prec s_2$ iff s_1 is a proper substring of s_2) is a well-founded set.

Let A be a set with a strict partial order $<$ and $X \subseteq A$. An element $a \in X$ is *minimal* for X if $x \not< a$, for all $x \in X$.

Proposition A.1. *Let A be a set with a strict partial order. Then A is a well-founded set iff every nonempty subset X of A has a minimal element (in X).*

Proof. Straightforward. □

Proposition A.2. *Let A be a set with a well-founded order $<$. Let X be a subset of A satisfying the condition: for all $a \in A$, whenever $b \in X$, for all $b < a$, it follows that $a \in X$. Then $X = A$.*

Proof. Suppose that $X \neq A$. Thus $A \setminus X \neq \emptyset$ and so $A \setminus X$ has a minimal element a, say. Consider an element $b \in A$ such that $b < a$. By the minimality of a, it follows that $b \notin A \setminus X$ and thus $b \in X$. Since this is true for all $b < a$, it follows that $a \in X$, by the condition satisfied by X. This gives a contradiction and so $X = A$. □

The condition in Proposition A.2 satisfied by X implies that X must contain the minimal elements of A (since these have no predecessors).

Now here is the first principle of inductive construction on well-founded sets.

Proposition A.3. *Let A be a well-founded set and S a set. Then there exists a unique function $f : A \to S$ having arbitrary given values on the minimal elements of A and satisfying the condition that there is a rule that, for all $a \in A$, uniquely determines the value of $f(a)$ from the values $f(b)$, for $b < a$.*

Proof. Uniqueness is shown first. Suppose that there exist two distinct functions f and g having the same values on the minimal elements of A and satisfying the condition in the statement of the proposition. Let X be the set of elements of A on which f and g differ. Let a be a minimal element of X. Now a cannot be minimal in A because f and g agree on the minimal elements of A. Thus there exist elements $b \in A$ such that $b < a$. For such an element b, $f(b) = g(b)$, since $b \notin X$. By the condition satisfied by f and g, it follows that $f(a) = g(a)$, which gives a contradiction.

Next existence is demonstrated. Denote the set $\{b \in A \mid b \leq a\}$ by W_a. Let $X = \{a \in A \mid$ there exists a function f_a defined on W_a having the given values on the minimal elements of A in W_a and satisfying the uniqueness condition on $W_a\}$. I show by the induction principle of Proposition A.2 that $X = A$. Note that, if $a, b \in X$ and $b < a$, by the uniqueness part of the proof applied to W_b, it follows that $f_b(x) = f_a(x)$, for all $x \in W_b$. Suppose now that $a \in A$ and $b \in X$, for all $b < a$. By the previous remark, $a \in X$ – it suffices to define f_a by $f_a(b) = f_b(b)$, for all $b < a$, and let $f_a(a)$ be the value uniquely determined by the rule. By

Proposition A.2, $X = A$. Now define f by $f(a) = f_a(a)$, for all $a \in A$. Clearly, f has the required properties. □

Because of the type system used by the logic, there is a requirement for another principle of inductive construction that demonstrates the existence of functions that satisfy an extra condition. First a definition is needed.

Definition A.1. Let $\{A_i\}_{i\in I}$ be a partition of the set A, $\{S_i\}_{i\in I}$ a partition of the set S, and $f : A \to S$ a mapping. Then $a \in A$ is *consistent* (with respect to f, $\{A\}_{i\in I}$ and $\{S\}_{i\in I}$) if $a \in A_i$ implies $f(a) \in S_i$.

Here is the second principle of inductive construction on well-founded sets.

Proposition A.4. *Let A be a well-founded set, $\{A_i\}_{i\in I}$ a partition of A, S a set, and $\{S_i\}_{i\in I}$ a partition of S. Then there exists a unique function $f : A \to S$ such that $f(A_i) \subseteq S_i$, for all $i \in I$, satisfying the following conditions.*

1. *For all $a \in A$, if a is minimal, then a is consistent.*
2. *There is a rule that, for all $a \in A$, uniquely determines the value of $f(a)$ from the values $f(b)$, for $b < a$, in such a way that if each b is consistent, then a is consistent.*

Proof. The existence and uniqueness follows from Proposition A.3. It remains to show that $f(A_i) \subseteq S_i$, for all $i \in I$.

Let $X = \{a \in A \mid a \text{ is consistent}\}$. It suffices to show that $X = A$. Now X contains the minimal elements, by assumption. Suppose now that $a \in A$ and, for all $b < a$, $b \in X$. By assumption, $a \in X$. Thus, by Proposition A.2, $A = X$. □