Knowledge Representation and Reasoning in Modal Higher-order Logic

J.W. Lloyd

Computer Sciences Laboratory
College of Engineering and Computer Science
The Australian National University

August 23, 2007

Abstract

This paper studies knowledge representation and reasoning in a polymorphically-typed, multi-modal, probabilistic, higher-order logic. A detailed account of the syntax and semantics of the logic is given. A reasoning system that combines computation and proof is presented, and the soundness of the reasoning system is proved. The main ideas are illustrated by a variety of examples and some applications to agents.

Contents

1 Introduction

This paper is intended primarily as a contribution to knowledge representation and reasoning. I argue in the paper that it is highly desirable that a logic intended for general-purpose knowledge representation and reasoning be typed, modal, probabilistic, and higher-order: types generally provide accurate modelling of applications and support important software engineering requirements, modalities allow the modelling of epistemic and temporal aspects, probabilistic reasoning is needed to handle uncertainty, and higher-orderness provides higher-order functions for computation and also sets, multisets, and similar abstractions for modelling individuals. Fundamentally, the issue is about the *expressive power* of the logic, that is, the degree to which it enables designers to easily and accurately model applications; each of the above four characteristics contributes greatly to increasing expressive power. In addition, it is important that the reasoning mechanism of the logic be 'manageable', in the sense that, even if the logic is not decidable, it is at least possible to easily prove that for each application the associated reasoning tasks will terminate, perhaps with some guaranteed time and/or space complexity.

The combinations of modal and higher-order, polymorphically-typed and higher-order, and first-order and probabilistic have been studied extensively before. (See Section 8.) However, it appears that the combination of all four features is new; certainly, polymorphically-typed, multi-modal, probabilistic, higher-order logic is rather more expressive than the logics that are commonly used in computer science. I argue that this expressive power is extremely useful in applications since it makes it possible to model applications in a language that is intuitive and comprehensible, and that the computational cost one pays for working in such an expressive logic (which is, of course, undecidable) is well worth the cost. In fact, I argue that the logic is no more difficult to work with than, say, first-order logic, and that its expressive power gives it plenty of advantages: it is rich enough to include the majority of applications in logic programming, functional programming, HOL-like applications, model checking and other applications of modal logic, symbolic machine learning, and so on.

While a whole array of applications does come within the purview of the logic presented here, one particular class of applications strongly motivated the development and largely explains the choice of topics in the paper. This broad class of applications concerns agent systems. The concept of an agent in many respects has become the fundamental unifying concept that underlies modern developments in artificial intelligence [RN02] and is surely going to provide the foundation for more sophisticated forms of software engineering in the future [Woo97]. Since almost any computer system doing something interesting can be usefully thought of as an agent, I confine the following motivation to that application domain.

I now motivate the modal aspects of the logic. While modalities can have a variety of meanings, what are of most concern here are epistemic and temporal modalities. For each (sub)agent i in a system, there is a corresponding epistemic operator typically either $\boldsymbol{K}_i$ or $\boldsymbol{B}_i$. If φ is a formula, then the informal meaning of $\boldsymbol{K}_i\varphi$ is 'agent i knows φ', while the informal meaning of $\boldsymbol{B}_i\varphi$ is 'agent i believes φ'. These modalities allow one to capture quite subtle facts about the knowledge and beliefs of agents. For example,

$$\boldsymbol{B}_i\varphi \longrightarrow \boldsymbol{B}_i\boldsymbol{B}_j\varphi$$

means that 'if agent i believes φ, then agent i believes that agent j believes φ'. Temporal modalities are also useful. For example, here are some typical (past) temporal modalities: $\bullet$ ('last'), $\blacksquare$ ('always in the past'), $\blacklozenge$ ('sometime in the past'), and $\boldsymbol{S}$ ('since'). (There are

also future versions of these.) Thus $\bullet\varphi$ means informally that 'φ held at the last (that is, immediately preceding) time'. Putting the two kinds of modalities together, there are formulas such as

$$\bullet \boldsymbol{B}_i\varphi \longrightarrow \boldsymbol{B}_i\bullet\varphi$$

whose informal meaning is 'if, at the last time, agent i believed φ, then agent i believes (now) that φ held at the last time'. As part of an agent's belief base, this kind of belief is genuinely useful for determining which action to perform next, as I show later and has been discussed widely in the literature, in [FHMV95], for example.

Furthermore, the logic allows modalities to be applied to arbitrary terms (not just formulas). So, for example, $\bullet f$, where f is a function, and $\boldsymbol{B}_i 42$ are legitimate terms. Such terms are called *modal terms*, in contrast with the conventional modal formulas. This extension beyond the standard meaning of modalities will turn out to be very useful. For example, in machine learning applications, there is a need for grammars to construct predicates for hypothesis languages. Suitable grammars, called predicate rewrite systems, were studied in the case of (non-modal) higher-order logic in [Llo03]. The setting there is extended here to the modal case by allowing modalities to be applied to predicates. As another example, for belief acquisition for agents, it is useful to allow modalities to be applied to functions. Examples to illustrate both of these application areas will be given later.

Higher-order logic admits functions that can have other functions as arguments. This feature is heavily exploited by modern declarative programming languages such as Haskell [Jon03], the prelude for which largely consists of higher-order functions. For agent applications, two consequences of the use of higher-order logic are noteworthy. One is that there is a class of terms which can be used systematically to represent individuals in applications and this class neatly models (finite) sets and multisets as abstractions (therefore, the functions that process sets and multisets are higher-order). Furthermore, a key idea here that a set is a predicate allows extensional and intensional sets to be processed uniformly by higher-order functions. The other consequence is that the higher-order nature of the logic provides the basis of predicate rewrite systems which are a convenient grammar formalism for expressing hypothesis languages for the learning components of adaptive agents. These two uses of higher-order logic are studied to some extent in this paper and, in much more detail, in [Llo03].

Another feature is the polymorphic type system. It isn't necessary to say much about the need for a type system. Apart from other considerations, good software engineering practice demands a type system to accurately model applications and avoid many typical programming errors. (Parametric) polymorphism, in particular, is also needed for many common functions that operate in a similar way on a variety of data structures. A programming language such as Haskell provides a good illustration of the value of having a sophisticated type system.

As well as representing knowledge, it is necessary to reason with it. The reasoning system introduced in this paper combines a theorem prover and an equational reasoning system. The theorem prover is a fairly conventional tableau theorem prover for modal higher-order logic similar to what is proposed in [Fit02]. The equational reasoning system is, in effect, a computational system that significantly extends existing declarative programming languages by adding facilities for computing with modalities. The proof component and the computational component are tightly integrated, in the sense that either can call the other. Furthermore, this synergy between the two is shown to make possible all kinds of interesting reasoning tasks.

It turns out that, for agent applications, the most common reasoning task is a computational one, that of evaluating a function call. In this case, the theorem-prover plays a subsidiary role, usually that of performing some rather straightforward modal theorem-proving tasks. However, in other applications it can just as easily be the other way around with the computational system performing subsidiary equational reasoning tasks for the theorem prover. The theoretical foundations for the reasoning system that are developed in the paper fully support both cases.

In many applications, it is necessary to deal with uncertainty. One of the great advantages of working in a higher-order logic is that it is expressive enough to easily encompass uncertainty without any additional machinery. The key idea is to represent uncertainty by densities (of measures). These are non-negative functions whose integral is one. Densities can conveniently be manipulated by higher-order functions and can be included in theories. In this way, it is straightforward to represent directly in a theory the probability that a particular assumption holds and also compute the probability that a theorem proved from such assumptions holds. In the paper, the theoretical foundations of this approach to uncertainty are developed in detail and illustrated by applications to agents.

Thus the paper introduces a polymorphically-typed, multi-modal, probabilistic, higher-order logic for representing knowledge and reasoning about this knowledge in a large variety of applications. The syntax and semantics of the logic, and a reasoning system that combines computation and proof for the logic are presented in detail. The main ideas are illustrated by a variety of examples and some applications to agents.

In the next section, the syntax of the logic is presented. Section 3 gives its semantics. In Section 4, the logic is generalised to the polymorphic case. Section 5 contains a discussion of grammars for generating predicates in hypothesis languages for learning applications. Section 6 considers a probabilistic reasoning system that combines computation and proof, and illustrates this with examples. Section 7 briefly discusses an approach to agent architecture that exploits the logic and gives several agent applications. In Section 8, related work is discussed. The last section gives some conclusions and future work.

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 which is added later in the paper).

2.1 Types

Definition 2.1. An *alphabet* consists of three (mutually disjoint) 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). Constants can be declared to be *rigid*. In the semantics introduced below, rigid constants have the same meaning in every world. 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 and the sets of variables of each type are pairwise disjoint.

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.

A useful property of a type is its order.

Definition 2.3. The *order* of a type is defined by induction as follows.

1. A type has order 0 if it is a type constructor of arity 0.

2. A type has order $\max\{order(\alpha_1), \ldots, order(\alpha_k)\}$, if it has the form $T\ \alpha_1 \ldots \alpha_k$, where $order(\alpha_i)$ is the order of α_i, for $i = 1, \ldots, k$.

3. A type has order $\max\{order(\alpha_1), \ldots, order(\alpha_n)\}$, if it has the form $\alpha_1 \times \cdots \times \alpha_n$, where $order(\alpha_i)$ is the order of α_i, for $i = 1, \ldots, n$.

4. A type has order $1 + \max\{order(\alpha), order(\beta)\}$, if it has the form $\alpha \rightarrow \beta$, where $order(\alpha)$ is the order of α and $order(\beta)$ is the order of β.

The order is defined for each type. For example, the order of $(Int \rightarrow Int \rightarrow \Omega) \times Real$ is 2.

For the purposes of this paper, the rank of a type will be more much useful than its order.

Definition 2.4. The *rank* of a type is defined by induction as follows.

1. A type has rank 0 if it has the form $T\ \alpha_1 \ldots \alpha_k$ or $\alpha_1 \times \cdots \times \alpha_n$.

2. A type has rank $n+1$ if it has the form $\alpha \to \beta$, where β is a type of rank n.

The rank is defined for each type. A type has rank n if it has the form

$$\alpha_1 \to \cdots \to \alpha_n \to \beta,$$

where β has rank 0. Intuitively, the rank of a type is the number of 'top-level' arrows in the type. Note that the rank of a type is generally quite different to its order. For example, the rank of $(\mathit{Int} \to \mathit{Int} \to \Omega) \times \mathit{Real}$ is 0.

Much importance is traditionally attached to the class of terms that have type Ω, that is, the class of formulas. However, I will show that formulas can generally be replaced throughout by a bigger class of terms called biterms and that this generalisation is useful. For example, biterms can be arguments of the connectives and quantifiers. Moreover, the dual modalities will be introduced and shown to be applicable to biterms. Also theories can consist of biterms, not just formulas.

Next the type of biterms is introduced.

Definition 2.5. A *biterm type of rank* n ($n \geq 0$) is a type having the form

$$\alpha_1 \to \cdots \to \alpha_n \to \Omega.$$

A *biterm type* is a biterm type of rank n, for some $n \geq 0$.

There is just one biterm type of rank 0 and that is Ω. Note that, if $n \geq 1$, then $\alpha_2 \to \cdots \to \alpha_n \to \Omega$ is a biterm type of rank $n-1$. Also, if α is any type, then $\alpha \to \alpha_1 \to \cdots \to \alpha_n \to \Omega$ is a biterm type of rank $n+1$.

Notation. If $\alpha \equiv \alpha_1 \to \cdots \to \alpha_n \to \Omega$ is a biterm type of rank > 0, then $\mathsf{t}(\alpha)$, the *tail* of α, is defined by $\mathsf{t}(\alpha) = \alpha_2 \to \cdots \to \alpha_n \to \Omega$.

Thus $\mathsf{t}(\alpha_1 \to \Omega) = \Omega$ and $\mathsf{t}(\alpha_1 \to \alpha_2 \to \Omega) = \alpha_2 \to \Omega$.

Definition 2.6. 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, 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 constants $\sharp_\alpha$ (cons) used to construct lists (whose items have type α) are data constructors. All data constructors are rigid constants. In typical applications, many functions are not rigid. For example, a function whose definition is believed by agents may have a different definition for different agents and/or a different definition for the same agent at different times (and therefore its meaning may vary from world to world).

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

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

1. (), having signature *1*.

2. $=_\alpha$, having signature $\alpha \to \alpha \to \Omega$, for each type α.

3. $\top$ and $\bot$, having signature Ω.

4. $\neg_\alpha$, having signature $\alpha \to \alpha$, for each biterm type α.

5. $\wedge_\alpha$, $\vee_\alpha$, $\longrightarrow_\alpha$, and $\longleftarrow_\alpha$, having signature $\alpha \to \alpha \to \alpha$, for each biterm type α.

6. Σ_α and Π_α, having signature $\alpha \to \Omega$, for each biterm type α.

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, and the intended meaning of $\bot$ is false. Usually, $=_\alpha$ is written as an infix operator.

The intended meanings of the connectives $\neg_\Omega$, $\wedge_\Omega$, $\vee_\Omega$, $\longrightarrow_\Omega$, and $\longleftarrow_\Omega$ that operate on formulas are as usual. The intended meanings of the connectives when their arguments have biterm type of rank > 0 are given in Section 2.2. I usually write $\wedge_\alpha$, $\vee_\alpha$, $\longrightarrow_\alpha$, and $\longleftarrow_\alpha$ as infix operators.

The intended meanings of both Π_Ω and Σ_Ω are the identity mapping on the booleans. When the biterm type α has rank 1, 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 $\top$. The intended meanings of Π_α and Σ_α, when α is a biterm type of rank > 1, are given in Section 2.2.

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.

2.2 Terms

The next task is to define the central concept of a term. For the modal part of the definition, I assume there are necessity modal operators $\Box_i$, for $i = 1, \ldots, m$.

Definition 2.7. 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 α and $i \in \{1, \ldots, m\}$, then $\Box_i t$ is a term of type α.

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

Example 2.2. The data constructors for constructing lists whose items have type α are $[]_\alpha : \mathit{List}\ \alpha$ and $\sharp_\alpha : \alpha \to \mathit{List}\ \alpha \to \mathit{List}\ \alpha$, where $\sharp_\alpha$ is usually written infix. The term $[]_\alpha$ represents the empty list and the term $s\,\sharp_\alpha\,t$ represents the list with head s and tail t, where the items have type α. Thus $4\,\sharp_{Int}\,5\,\sharp_{Int}\,6\,\sharp_{Int}\,[]_{Int}$ represents the list $[4,5,6]$, and $A\,\sharp_T\,B\,\sharp_T\,C\,\sharp_T\,[]_T$ represents the list $[A,B,C]$, where A, B, and C are constants of some type T, say.

The notation $[]_\alpha$ and $\sharp_\alpha$, with the subscript α indicating the type of items, is cumbersome, but is needed if one wants to explicitly distinguish the various constants. When polymorphism is introduced in Section 4, the subscripts will be done away with (at the expense of possible confusion about exactly which constant is intended).

Next the class of biterms, a class of terms that includes the class of formulas and allows many concepts to be extended from formulas to biterms, is introduced.

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

Definition 2.9. A *biterm of rank n* is a term whose type is a biterm type of rank n.

A *biterm* is a biterm of rank n, for some $n \geq 0$.

Biterms are usually denoted by Greek symbols φ, ψ, and so on.

A biterm of rank 0 is a term having type Ω, that is, a formula; a biterm of rank 1 is a term having type $\alpha_1 \to \Omega$, for some α_1; a biterm of rank 2 is a term having type $\alpha_1 \to \alpha_2 \to \Omega$, for some α_1 and α_2; and so on. If φ is a biterm of rank n, then $\lambda x.\varphi$ is a biterm of rank $n+1$. If φ is a biterm of rank n and $(\varphi\ t)$ is a term, then $(\varphi\ t)$ is a biterm of rank $n-1$.

I now give definitions of the connectives and quantifiers applied to biterms, generalising the case of formulas in [And86, p.163]. No subsequent use will be made in this paper of these definitions; nevertheless, it is interesting to see how the connectives and quantifiers can be defined using the more primitive notions of equality, $\top$ and $\bot$. (One can even define $\top$ and $\bot$ in terms of $=_\Omega$, if desired.)

Definition 2.10. The function $\neg_\alpha : \alpha \to \alpha$ is defined by induction on the rank of α by

$$\neg_\Omega\ =_{\Omega\to\Omega}\ (=_\Omega\ \bot)$$

$$\neg_\alpha\ =_{\alpha\to\alpha}\ \lambda y.\lambda x.(\neg_{\mathsf{t}(\alpha)}\ (y\ x)), \quad \text{where } \alpha \text{ has rank} > 0.$$

The function $\wedge_\alpha : \alpha \to \alpha \to \alpha$ is defined by induction on the rank of α by

$$\wedge_\Omega\ =_{\Omega\to\Omega\to\Omega}\ \lambda y.\lambda z.(\lambda g.(g\ \top\ \top) =_\Omega \lambda g.(g\ y\ z))$$

$$\wedge_\alpha\ =_{\alpha\to\alpha\to\alpha}\ \lambda y.\lambda z.\lambda x.(\wedge_{\mathsf{t}(\alpha)}\ (y\ x)\ (z\ x)), \quad \text{where } \alpha \text{ has rank} > 0.$$

The function $\vee_\alpha : \alpha \to \alpha \to \alpha$ is defined by

$$\vee_\alpha =_{\alpha\to\alpha\to\alpha} \lambda y.\lambda z.(\neg_\alpha\ (\wedge_\alpha\ (\neg_\alpha\ y)\ (\neg_\alpha\ z))).$$

The function $\longrightarrow_\alpha : \alpha \to \alpha \to \alpha$ is defined by

$$\longrightarrow_\alpha =_{\alpha\to\alpha\to\alpha} \lambda y.\lambda z.(\vee_\alpha\ (\neg_\alpha\ y)\ z).$$

The function $\longleftarrow_\alpha$ is defined similarly.

Now I consider the functions Π_α and Σ_α.

Definition 2.11. The function $\Pi_\alpha : \alpha \to \Omega$, where α is a biterm type of rank $n > 0$, is defined by

$$\Pi_\alpha =_{\alpha\to\Omega} (=_\alpha \lambda x_1. \ldots \lambda x_n.\top).$$

Definition 2.12. The function $\Sigma_\alpha : \alpha \to \Omega$, where α is a biterm type of rank $n > 0$, is defined by

$$\Sigma_\alpha =_{\alpha\to\Omega} (\neq_\alpha \lambda x_1. \ldots \lambda x_n.\bot),$$

where $\neq_\alpha : \alpha \to \alpha \to \Omega$ is defined by $\neq_\alpha =_{\alpha\to\alpha\to\Omega} \lambda x.\lambda y.\neg_\Omega(x =_\alpha y)$.

To explain informally the intended semantics of Π_α and Σ_α, a little more machinery is needed. Let $\{X_i\}_{i=1}^n$ be a family of non-empty sets. Then the function

$$\Tbbb : X_1 \to \cdots \to X_n \to \{\top, \bot\}$$

is defined by $\Tbbb\ x_1\ \ldots\ x_n = \top$, for all $x_1, \ldots, x_n$. If $n = 0$, then $\Tbbb$ is just $\top$. If $n = 1$, then $\Tbbb$ is the function that maps each $x_1 \in X_1$ to $\top$, that is, the weakest predicate on X_1.

Similarly, the function

$$\Bbb : X_1 \to \cdots \to X_n \to \{\top, \bot\}$$

is defined by $\Bbb\ x_1\ \ldots\ x_n = \bot$, for all $x_1, \ldots, x_n$. If $n = 0$, then $\Bbb$ is just $\bot$. If $n = 1$, then $\Bbb$ is the function that maps each $x_1 \in X_1$ to $\bot$, that is, the strongest predicate on X_1.

In the semantics given below, for each type α there corresponds a domain $\mathcal{D}_\alpha$. For a type of the form $\alpha \to \beta$, the domain $\mathcal{D}_{\alpha\to\beta}$ is the collection of mappings from $\mathcal{D}_\alpha$ to $\mathcal{D}_\beta$. If α is a biterm type, then $\Tbbb$ and $\Bbb$ are elements in $\mathcal{D}_\alpha$.

Now the intended semantics of Π_α and Σ_α can be given informally. The intended meaning of Π_α is the function from $\mathcal{D}_\alpha$ to $\mathcal{D}_\Omega$ that maps $\Tbbb$ in $\mathcal{D}_\alpha$ to $\top$ and maps all other functions in $\mathcal{D}_\alpha$ to $\bot$. The intended meaning of Σ_α is the function from $\mathcal{D}_\alpha$ to $\mathcal{D}_\Omega$ that maps $\Bbb$ in $\mathcal{D}_\alpha$ to $\bot$ and maps all other functions in $\mathcal{D}_\alpha$ to $\top$.

The quantifiers can be defined for biterms. Let t be a biterm of type β and x a variable of type α. The universal quantifier $\forall_\alpha$ is introduced by the abbreviation

$$\forall_\alpha x.t \equiv (\Pi_{\alpha\to\beta}\ \lambda x.t)$$

and the existential quantifier $\exists_\alpha$ is introduced by the abbreviation

$$\exists_\alpha x.t \equiv (\Sigma_{\alpha\to\beta}\ \lambda x.t).$$

Note that $\forall_\alpha x.t$ and $\exists_\alpha x.t$ are formulas (even when t is a biterm that is not a formula). Later I will show that $\exists_\alpha x.t$ and $\neg\forall_\alpha x.(\neg_\beta\ t)$ have the same meaning. Similarly, $\forall_\alpha x.t$ and $\neg\exists_\alpha x.(\neg_\beta\ t)$ have the same meaning.

The possibility modalities that are dual to the necessity modalities are now introduced. Let t be a biterm of type α. Then the possibility modalities $\Diamond_i$ are introduced by the abbreviation

$$\Diamond_i t \equiv (\neg_\alpha\ \Box_i(\neg_\alpha\ t)),$$

for $i = 1, \ldots, m$. Note that while $\Box_i$ can be applied to arbitrary terms, $\Diamond_i$ can only be applied to biterms because negation only makes sense for biterms.

Example 2.3. Condition 6 of Definition 2.7 admits some rather odd looking terms. For example, one can write $\Box_i 42$ or $\Box_i A$, where A is a data constructor, or $\Box_i f$, where f is a function, or even $\Box_i(A, 42)$. All these kinds of modal terms and many others will be useful in practice.

Notation. As will have become apparent by now, the type subscripts on equality, the connectives, and the quantifiers are intrusive. So, from now on, at the possible cost of some imprecision at times, I will discard them. Thus $=_\alpha$ (resp., $\neg_\alpha$, $\wedge_\alpha$, $\vee_\alpha$, $\longrightarrow_\alpha$, $\longleftarrow_\alpha$, Π_α, Σ_α, $\forall_\alpha$, $\exists_\alpha$) will be denoted by $=$ ($\neg$, $\wedge$, $\vee$, $\longrightarrow$, $\longleftarrow$, Π, Σ, $\forall$, $\exists$).

A term of the form $(\neg\ t)$ is usually abbreviated to $\neg t$. So, for example, $(\neg\ \Box_i(\neg\ t))$ would be shortened to $\neg\Box_i\neg t$.

Application of modalities is right associative. Thus $(\Box_{j_1}(\Box_{j_2} \cdots (\Box_{j_r} t) \cdots))$ can be written as $\Box_{j_1}\Box_{j_2} \cdots \Box_{j_r} t$.

$\forall x_1 \ldots \forall x_n.\varphi$ is an abbreviation of $(\Pi\ \lambda x_1 \ldots \lambda x_n.\varphi)$.

$\exists x_1 \ldots \exists x_n.\varphi$ is an abbreviation of $(\Sigma\ \lambda x_1 \ldots \lambda x_n.\varphi)$.

Definition 2.13. A term is *rigid* if each constant in the term is rigid.

In the semantics below, rigid terms have the same meaning in each world.

In Definition 2.7, 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 2.7 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 $(t, \alpha) \in \mathfrak{X}$ and $i \in \{1, \ldots, m\}$, then $(\Box_i t, \alpha) \in \mathfrak{X}$.

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 2.7 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.*

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. Conversely, if t is a term, 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 $\mathfrak{L} = \bigcup_{\alpha \in \mathfrak{S}} \mathfrak{L}_\alpha$.

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 $t \in \mathfrak{X}$, then $\Box_i t \in \mathfrak{X}$.*

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

Proof. Clearly $\mathfrak{X}$ satisfies Conditions 1 to 6 of Definition 2.7. 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.

Definition 2.14. 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 2.15. A term is *closed* if it contains no free variables; otherwise, it is *open*.

Definition 2.16. Let φ be a biterm with free variables $x_1, \ldots, x_n$, where x_i has type α_i, for $i = 1, \ldots, n$, and where the leftmost free occurrences in φ of the variables are in the order $x_1, \ldots, x_n$. Then the *universal closure* of φ, denoted $\forall(\varphi)$, is the biterm $\forall x_1 \ldots \forall x_n.\varphi$.

Note that, provided φ has at least one free variable, $\forall(\varphi)$ is a formula, even when φ is not a formula. If φ has no free variables, then $\forall(\varphi)$ is just φ.

2.3 Occurrences

The concept of an occurrence of a subterm of a term will be needed. 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 2.17. 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 2.18. 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 2.19. 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.

Definition 2.20. For a particular occurrence of a subterm $\Box_i t$ in a term, the occurrence of t is called the *scope* of the $\Box_i$.

An occurrence is *modal* if it occurs within the scope of a subterm of the form $\Box_i t$.

A related concept is that of the modal path to an occurrence in a term. Intuitively, this is the sequence of indexes of the modalities that are encountered on the path from the root of the term down to the occurrence.

Definition 2.21. If t is a term and $o \in \mathcal{O}(t)$, then the *modal path to o in t* is defined inductively on the length of o as follows.

1. If $o = \varepsilon$, then the modal path is empty.

2. If $o = 1o'$, for some o', and t has the form $\lambda x.s$, then the modal path to o in t is the same as the modal path to o' in s.
   If $o = 1o'$, for some o', and t has the form $(u\ v)$, then the modal path to o in t is the same as the modal path to o' in u.
   If $o = 2o'$, for some o', and t has the form $(u\ v)$, then the modal path to o in t is the same as the modal path to o' in v.
   If $o = io'$, for some o', and t has the form $(t_1, \ldots, t_n)$, then the modal path to o in t is the same as the modal path to o' in t_i, for $i = 1, \ldots, n$.
   If $o = 1o'$, for some o', and t has the form $\Box_i s$ and the modal path to o' in s is p, then the modal path to o in t is ip.

2.4 Substitutions

Next substitutions are introduced.

Definition 2.22. 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 set $\{x_1, \ldots, x_n\}$ is called the *domain* of the substitution. The set of free variables in $\{t_1, \ldots, t_n\}$ is called the *range* of the substitution.

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

Let θ be a substitution and t a term. Then $\theta|_t$ is the substitution obtained from θ by restricting the domain of 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 2.23. 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 is a box term $\Box_i s$, then $(\Box_i s)\theta = \Box_i(s\theta)$.

More precisely, what is being defined in Definition 2.23 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 2.23, 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 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 $s\theta|_{\lambda x.s}$ $\quad [(\lambda x.s)\theta = \lambda x.(s\theta|_{\lambda x.s})]$
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) $\quad$ [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 preceding part.
Let t have the form $\Box_j s$. Thus $(\Box_j s)\theta = \Box_j(s\theta)$. Then

x is free in $(\Box_j s)\theta$
iff x is free in $s\theta$ $[(\Box_j s)\theta = \Box_j(s\theta)]$
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 $\Box_j s$ 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 $\Box_j s$.

□

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

$$\begin{aligned}
&(u\ v)\theta \\
&= (u\theta\ v\theta) \quad \text{[Part 4 of Definition 2.23]} \\
&= (u\theta|_u\ \ v\theta|_v) \quad \text{[induction hypothesis]} \\
&= (u\theta|_{(u\ v)}\ \ v\theta|_{(u\ v)}) \quad \text{[induction hypothesis]} \\
&= (u\ v)\theta|_{(u\ v)} \quad \text{[Part 4 of Definition 2.23].}
\end{aligned}$$

If t is a tuple or has the form $\Box_i s$, the proof is similar to the preceding 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 2.23, 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 3.29 below.

2.5 Term Replacement

For the equational reasoning introduced in Section 6, it will be necessary to replace subterms of terms by other terms.

Definition 2.24. 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.

2.6 α-Conversion

Sometimes it is convenient to be able to rename bound variables. In the next definition, the relation $\succ_\alpha$ corresponding to α-conversion is introduced for this purpose.

Definition 2.25. The rule of *α-conversion* is as follows: $\lambda x.t \succ_\alpha \lambda y.(t\{x/y\})$, if y is not free in t and x does not occur freely in t in a subterm of the form $\lambda y.s$.

In the preceding definition, the bound variable x in the abstraction is replaced by the bound variable y. The condition that y not be free in t ensures the meaning of $\lambda x.t$ is preserved under the renaming. The condition that x does not occur freely in t in a subterm of the form $\lambda y.s$ ensures that no further renaming will be necessary when the substitution $\{x/y\}$ is applied to t. Clearly, if y does not occur in $\lambda x.t$ at all, both these conditions are satisfied.

Definition 2.26. The relation $\longrightarrow_\alpha$ is defined by $u \longrightarrow_\alpha u[s/t]_o$ if s is a subterm of u at occurrence o and $s \succ_\alpha t$.

Let $\xrightarrow{*}_\alpha$ be the reflexive, transitive closure of $\longrightarrow_\alpha$.

Let $\xleftrightarrow{*}_\alpha$ be the reflexive, symmetric, and transitive closure of $\longrightarrow_\alpha$.

Definition 2.27. If $s \xleftrightarrow{*}_\alpha t$, then s and t are said to be *α-equivalent*.

Note that α-equivalent terms differ only in the names of (some of) their bound variables. Also α-equivalent terms must have the same type.

Proposition 2.10. *Let t and t' be terms. Then the following hold.*

1. *If* $t \succ_\alpha t'$, *then* $t' \succ_\alpha t$.

2. *If* $t \overset{*}{\longleftrightarrow}_\alpha t'$, *then* $t \overset{*}{\longrightarrow}_\alpha t'$.

Proof. 1. Suppose that $\lambda x.r \succ_\alpha \lambda y.(r\{x/y\})$. Thus y is not free in r and x does not occur freely in r in a subterm of the form $\lambda y.s$. Note that x is not free in $r\{x/y\}$ and y does not occur freely in $r\{x/y\}$ in a subterm of the form $\lambda x.s$. Also $\lambda x.((r\{x/y\})\{y/x\}) = \lambda x.r$. Hence $\lambda y.(r\{x/y\}) \succ_\alpha \lambda x.r$.

2. It suffices to show that if $t \longrightarrow_\alpha t'$, then $t' \longrightarrow_\alpha t$. But this follows immediately from Part 1. □

2.7 Composition of Substitutions

The definition of the composition of two substitutions is now given. In preparation for this, three useful results involving α-equivalence and substitutions are proved.

Proposition 2.11. *Let* $\lambda x.s$ *be a term and* $\theta \equiv \{x_1/t_1, \ldots, x_n/t_n\}$ *a substitution 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 there exists a variable* y *such that* $\lambda x.s \succ_\alpha \lambda y.(s\{x/y\})$, $(\lambda x.s)\theta = (\lambda y.(s\{x/y\}))\theta$ *and, for all* $i \in \{1, \ldots, n\}$, x_i *is free in* $\lambda y.(s\{x/y\})$ *implies* y *is not free in* t_i.

Proof. Choose y to be the first variable of the same type as x that does not appear in $\lambda x.s$ or $\theta|_{\lambda x.s}$. Then y is not free in s and x does not occur freely in s in a subterm of the form $\lambda y.r$. Hence $\lambda x.s \succ_\alpha \lambda y.(s\{x/y\})$.

Clearly, for all $i \in \{1, \ldots, n\}$, x_i is free in $\lambda y.(s\{x/y\})$ implies y is not free in t_i. Thus $(\lambda x.s)\theta = \lambda y.(s(\{x/y\} \cup \theta|_{\lambda x.s})) = \lambda y.((s\{x/y\})\theta|_{\lambda y.(s\{x/y\})}) = (\lambda y.(s\{x/y\}))\theta$. □

Proposition 2.12. *Let* t *and* t' *be terms that are* α*-equivalent and* θ *a substitution. Then* $t\theta$ *is* α*-equivalent to* $t'\theta$.

Proof. Suppose that θ is $\{x_1/t_1, \ldots, x_n/t_n\}$. The proof is by induction on the structure of t.

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

Suppose that t has the form $\lambda x.s$. Then t' has the form $\lambda y.r$, where $\lambda x.s \overset{*}{\longleftrightarrow}_\alpha \lambda y.r$. By Proposition 2.11, it can be assumed without loss of generality that, for all $i \in \{1, \ldots, n\}$, x_i is free in $\lambda x.s$ implies x is not free in t_i, and x_i is free in $\lambda y.r$ implies y is not free in t_i.

There are two cases to consider.

(a) t' has the form $\lambda x.r$, where $s \overset{*}{\longleftrightarrow}_\alpha r$.

By the induction hypothesis, $s\theta|_{\lambda x.s} \overset{*}{\longleftrightarrow}_\alpha r\theta|_{\lambda x.s}$. Then

$$\begin{aligned} & (\lambda x.s)\theta \\ = \ & \lambda x.(s\theta|_{\lambda x.s}) \\ \overset{*}{\longleftrightarrow}_\alpha \ & \lambda x.(r\theta|_{\lambda x.r}) \quad \text{[induction hypothesis, since } \theta|_{\lambda x.s} = \theta|_{\lambda x.r}\text{]} \\ = \ & (\lambda x.r)\theta. \end{aligned}$$

(b) t' has the form $\lambda y.r$, where $x \neq y$.

Since $\lambda x.s \overset{*}{\longleftrightarrow}_\alpha \lambda y.r$, it follows that $\lambda x.s \overset{*}{\longleftrightarrow}_\alpha \lambda x.w \succ_\alpha \lambda y.(w\{x/y\}) \overset{*}{\longleftrightarrow}_\alpha \lambda y.r$, for some term w. Thus, by Part (a), it suffices to show that $(\lambda x.w)\theta \overset{*}{\longleftrightarrow}_\alpha (\lambda y.(w\{x/y\}))\theta$. For this,

note that y is not free in $w\theta|_{\lambda x.w}$ and x does not occur freely in $w\theta|_{\lambda x.w}$ in a subterm of the form $\lambda y.v$. Hence

$$\begin{aligned}
& (\lambda x.w)\theta \\
= \ & \lambda x.(w\theta|_{\lambda x.w}) \\
\succ_\alpha \ & \lambda y.((w\theta|_{\lambda x.w})\{x/y\}) \\
= \ & \lambda y.((w\{x/y\})\theta|_{\lambda y.(w\{x/y\})}) \\
= \ & (\lambda y.(w\{x/y\}))\theta.
\end{aligned}$$

Suppose that t has the form $(t_1, \ldots, t_n)$. Let t' be a term such that $t \overset{*}{\longleftrightarrow}_\alpha t'$. Thus $t' = (t'_1, \ldots, t'_n)$, where $t_i \overset{*}{\longleftrightarrow}_\alpha t'_i$, for $i = 1, \ldots, n$. By the induction hypothesis, $t_i\theta \overset{*}{\longleftrightarrow}_\alpha t'_i\theta$, for $i = 1, \ldots, n$. Then

$$\begin{aligned}
& (t_1, \ldots, t_n)\theta \\
= \ & (t_1\theta, \ldots, t_n\theta) \\
\overset{*}{\longleftrightarrow}_\alpha \ & (t'_1\theta, \ldots, t'_n\theta) \quad \text{[induction hypothesis]} \\
= \ & (t'_1, \ldots, t'_n)\theta.
\end{aligned}$$

If t has the form $(u\ v)$ or $\Box_i s$, the proof is similar to the preceding case.

□

Proposition 2.13. *Let θ and φ be substitutions. Then the following hold.*

1. *There are only finitely many variables $x_1, \ldots, x_m$ such that $(x_i\theta)\varphi \neq x_i$, for $i = 1, \ldots, m$.*
2. *$t\{x_1/(x_1\theta)\varphi, \ldots, x_m/(x_m\theta)\varphi\}$ is α-equivalent to $(t\theta)\varphi$, for all terms t.*

Proof. 1. For any variable x that does not occur in a binding of the form x/t in θ or φ, it is clear that $(x\theta)\varphi = x$.

2. Using Proposition 2.12, it can be assumed without loss of generality that no bound variable in t appears in θ or φ. The proof then proceeds by induction on the structure of the term t.

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

Suppose that t has the form $\lambda x.s$. Let $x_{i_1}, \ldots, x_{i_p}$ be those x_i that are free in $\lambda x.s$. Put $\theta' = \theta|_{\lambda x.s}$ and $\varphi' = \varphi|_{\lambda x.(s\theta')}$. Let $x_{j_1}, \ldots, x_{j_q}$ be those x_i such that $(x_i\theta')\varphi' \neq x_i$. Then

$$\begin{aligned}
& (\lambda x.s)\{x_1/(x_1\theta)\varphi, \ldots, x_m/(x_m\theta)\varphi\} \\
= \ & \lambda x.(s\{x_1/(x_1\theta)\varphi, \ldots, x_m/(x_m\theta)\varphi\}|_{\lambda x.s}) \\
= \ & \lambda x.(s\{x_{i_1}/(x_{i_1}\theta)\varphi, \ldots, x_{i_p}/(x_{i_p}\theta)\varphi\}) \\
= \ & \lambda x.(s\{x_{i_1}/(x_{i_1}\theta')\varphi', \ldots, x_{i_p}/(x_{i_p}\theta')\varphi'\}) \\
= \ & \lambda x.(s\{x_{j_1}/(x_{j_1}\theta')\varphi', \ldots, x_{j_q}/(x_{j_q}\theta')\varphi'\}) \\
\overset{*}{\longleftrightarrow}_\alpha \ & \lambda x.((s\theta')\varphi') \quad \text{[induction hypothesis]} \\
= \ & (\lambda x.(s\theta'))\varphi \\
= \ & ((\lambda x.s)\theta)\varphi.
\end{aligned}$$

If t has the form $(t_1, \ldots, t_n)$, then

$$\begin{aligned}
&(t_1, \ldots, t_n)\{x_1/(x_1\theta)\varphi, \ldots, x_m/(x_m\theta)\varphi\} \\
=\ &(t_1\{x_1/(x_1\theta)\varphi, \ldots, x_m/(x_m\theta)\varphi\}, \ldots, t_n\{x_1/(x_1\theta)\varphi, \ldots, x_m/(x_m\theta)\varphi\}) \\
\overset{*}{\longleftrightarrow}_\alpha\ &((t_1\theta)\varphi), \ldots, (t_n\theta)\varphi) \quad \text{[induction hypothesis]} \\
=\ &(t_1\theta, \ldots, t_n\theta)\varphi \\
=\ &((t_1, \ldots, t_n)\theta)\varphi.
\end{aligned}$$

If t has the form $(u\ v)$ or $\Box_i s$, the proof is similar to the preceding case. □

Here is an example to show that α-equivalence cannot be replaced by identity in Part 2 of Proposition 2.13. This is because bound variables may need to be renamed when a substitution is applied.

Example 2.4. Let t be $\lambda x.y$, θ be $\{y/x\}$, and φ be $\{x/z\}$. Then $t\theta = \lambda w.x$, say, where w is a fresh variable. Thus $(t\theta)\varphi = \lambda w.z$. Now x_1 is y and x_2 is x. Thus $t\{x_1/(x_1\theta)\varphi, x_2/(x_2\theta)\varphi\} = (\lambda x.y)\{y/z, x/z\} = \lambda x.z \neq (t\theta)\varphi$.

Now the definition of composition can be given.

Definition 2.28. Let θ and φ be substitutions. Then the *composition* $\theta \circ \varphi$ of θ and φ is the substitution $\{x_1/(x_1\theta)\varphi, \ldots, x_m/(x_m\theta)\varphi\}$, where $x_1, \ldots, x_m$ are the finitely many variables such that $(x_i\theta)\varphi \neq x_i$, for $i = 1, \ldots, m$.

According to Proposition 2.13, if α-equivalence rather than identity is sufficient for the particular situation (and it nearly always is), one can calculate $t(\theta \circ \varphi)$ by calculating $(t\theta)\varphi$ instead.

2.8 Matching

For the computation system introduced below, given terms s and t, there will be a need to determine whether or not there is a substitution θ such that $s\theta$ is (α-equivalent to) t. This motivates the next definition.

Definition 2.29. Let s and t be terms of the same type. Then a substitution θ is a *matcher* of s to t if $s\theta$ is α-equivalent to t. In this case, s is said to be *matchable* to t.

It follows immediately from the definition and the fact that $\overset{*}{\longleftrightarrow}_\alpha$ is an equivalence relation that, if θ and θ' are matchers of s to t, then $s\theta \overset{*}{\longleftrightarrow}_\alpha s\theta'$.

Here is a useful alternative formulation of the definition of matcher.

Proposition 2.14. *Let s and t be terms of the same type. Then a substitution θ is a matcher of s to t iff there exist a sequence of terms $s_0 = s, s_1, \ldots, s_n$ and an increasing subsequence $i_1, \ldots, i_m$ of the sequence $1, \ldots, n$ such that*

1. *there exists a substitution θ_{i_j} such that $s_{i_j-1}\theta_{i_j} = s_{i_j}$, for $j = 1, \ldots, m$,*
2. *$s_{i-1} \overset{*}{\longleftrightarrow}_\alpha s_i$, for $i \in \{1, \ldots, n\} \setminus \{i_1, \ldots, i_m\}$,*
3. *$s_n \overset{*}{\longleftrightarrow}_\alpha t$, and*

4. $\theta = \theta_{i_1} \circ \cdots \circ \theta_{i_m}$.

Proof. Suppose first that θ is a matcher of s to t. In this case, put $n = 1$ and $\theta_1 = \theta$. Then Conditions 1 to 4 are clearly satisfied.

Conversely, suppose that there exist a sequence of terms $s_0 = s, s_1, \ldots, s_n$ and an increasing subsequence $i_1, \ldots, i_m$ of the sequence $1, \ldots, n$ satisfying Conditions 1 to 4. By repeated application of Proposition 2.12, it follows that $(\ldots(s\theta_{i_1})\ldots)\theta_{i_m} \stackrel{*}{\longleftrightarrow}_\alpha s_n$. By Proposition 2.13, $s(\theta_{i_1} \circ \cdots \circ \theta_{i_m}) \stackrel{*}{\longleftrightarrow}_\alpha (\ldots(s_0\theta_{i_1})\ldots)\theta_{i_m}$. Thus $s(\theta_{i_1} \circ \cdots \circ \theta_{i_m}) \stackrel{*}{\longleftrightarrow}_\alpha t$, and so $\theta_{i_1} \circ \cdots \circ \theta_{i_m}$ is a matcher of s to t. □

The matching algorithm in Figure 1 determines whether one term is matchable with another. Note that the inputs to this algorithm are two terms that have no free variables in common. It is usual to standardise apart before applying a unification algorithm so doing this for matching as well is not out of the ordinary. If this condition were to be dropped, the algorithm in Figure 1 would need modification.

function *Match*(s, t) **returns** matcher θ, if s is matchable to t
failure, otherwise;

inputs: s and t, terms of the same type with no free variables in common;

$\theta := \{\}$;
while $s \neq t$ **do**

$o :=$ occurrence of innermost subterm containing symbol at leftmost point of disagreement between s and t;

if $s|_o$ has form $\lambda x.v$ and $t|_o$ has form $\lambda y.w$ % where $x \neq y$

then

$s := s[\lambda x.v/\lambda z.(v\{x/z\})]_o$; % where z is a variable not in s or t
$t := t[\lambda y.w/\lambda z.(w\{y/z\})]_o$;

else if $s|_o$ is a free occurrence of a variable x and there is no free occurrence of x in s to the left of o and each free occurrence of a variable in $t|_o$ is a free occurrence in t

then

$\theta := \theta \circ \{x/t|_o\}$;
$s := s\{x/t|_o\}$;

else return failure;

return θ;

Figure 1: Algorithm for finding a matching substitution

I now turn to establishing the correctness and termination of this algorithm.

Proposition 2.15. *Let s and t be terms of the same type with no free variables in common. If s is matchable to t, then the algorithm in Figure 1 terminates and returns a matcher of s*

to t. Otherwise, the algorithm terminates and returns failure.

Proof. The algorithm does terminate since all terms obtained from t by changing the names of bound variables have the same fixed length as t and, in each iteration of the loop, the point of disagreement moves strictly to the right.

Suppose first that s is not matchable to t. In this case, the algorithm cannot return successfully, since this would imply that s was matchable to t, by Proposition 2.14. Thus, since the algorithm terminates, it must return failure.

Suppose now that s is matchable to t. Let φ be a matcher of s to t, so that $s\varphi$ is α-equivalent to t. Now, at each step of the algorithm, either a further change of a bound variable is made to s and t or else a further binding of the form $\{x/t|_o\}$ is applied to s. Let the sequence of terms obtained by the algorithm starting from s by either changing a bound variable or else by applying a binding at each step be $s_0(= s), s_1, s_2, \ldots$. Similarly, let the sequence of terms obtained starting from t by either changing a bound variable or else doing nothing be $t_0(= t), t_1, t_2, \ldots$. I will show by induction on the number n of steps so far of the algorithm that there exists a substitution π_n such that the domain of π_n is a subset of the free variables in s, the range of π_n is a subset of the free variables in t, and $s_n\pi_n \overset{*}{\longleftrightarrow}_\alpha t_n$, for $n = 0, 1, \ldots$.

For $n = 0$, put $\pi_0 = \varphi|_s$. Then $s_o\pi_o \overset{*}{\longleftrightarrow}_\alpha t_0$, since $s\varphi \overset{*}{\longleftrightarrow}_\alpha t$. Since s and t have no free variables in common, the domain of π_0 is the set of free variables in s and the range is the set of free variables in t.

Next suppose that the result holds for n so that the domain of π_n is a subset of the free variables in s, the range of π_n is a subset of the free variables in t, and $s_n\pi_n \overset{*}{\longleftrightarrow}_\alpha t_n$. For the $(n + 1)$th step, there are two cases to consider.

If the disagreement is such that the $(n + 1)$th step of the algorithm is a change of bound variables, put $\pi_{n+1} = \pi_n$. Then $s_{n+1} \overset{*}{\longleftrightarrow}_\alpha s_n$, so that $s_{n+1}\pi_{n+1} \overset{*}{\longleftrightarrow}_\alpha s_n\pi_{n+1}$, by Proposition 2.12. Now $s_n\pi_{n+1} = s_n\pi_n$, $s_n\pi_n \overset{*}{\longleftrightarrow}_\alpha t_n$ (by the induction hypothesis), and $t_n \overset{*}{\longleftrightarrow}_\alpha t_{n+1}$. Hence $s_{n+1}\pi_{n+1} \overset{*}{\longleftrightarrow}_\alpha t_{n+1}$.

Suppose next that the disagreement does not involve a change of bound variables. Then the condition immediately following the **else if** must hold. That is, the disagreement must be at a free occurrence o of a variable x and there must be no free occurrence of x in s_n to the left of o and each free occurrence of a variable in $t_n|_o$ must be a free occurrence in t_n; otherwise, it would not be true that $s_n\pi_n \overset{*}{\longleftrightarrow}_\alpha t_n$. Thus π_n must contain a binding of the form $\{x/r\}$, where r is α-equivalent to $t_n|_o$. Put $\pi_{n+1} = \pi_n \setminus \{x/r\}$. Clearly the domain of π_{n+1} is a subset of the free variables in s and the range of π_{n+1} is a subset of the free variables in t. Thus $\pi_n = \{x/r\} \circ \pi_{n+1}$. Also

$$
\begin{aligned}
& s_{n+1}\pi_{n+1} \\
&= (s_n\{x/t_n|_o\})\pi_{n+1} \\
&\overset{*}{\longleftrightarrow}_\alpha (s_n\{x/r\})\pi_{n+1} \quad \text{[Proposition 2.12, since } r \text{ is } \alpha\text{-equivalent to } t_n|_o\text{]} \\
&\overset{*}{\longleftrightarrow}_\alpha s_n(\{x/r\} \circ \pi_{n+1}) \quad \text{[Proposition 2.13]} \\
&= s_n\pi_n \\
&\overset{*}{\longleftrightarrow}_\alpha t_n \quad \text{[induction hypothesis]} \\
&= t_{n+1}.
\end{aligned}
$$

This completes the induction argument.

In effect, it has been shown that in the case that s is matchable to t, the algorithm cannot terminate with failure.

Now suppose that s is matchable to t and the algorithm terminates at the nth step with $s_n = t_n$. Let the sequence of bindings obtained by the algorithm be $\theta_{i_1}, \ldots, \theta_{i_m}$. Put $\theta = \theta_{i_1} \circ \cdots \circ \theta_{i_m}$. Then Conditions 1 to 4 of Proposition 2.14 are satisfied, so that θ is a matcher of s to t. Finally, it is clear from the algorithm that the domain of θ is a subset of the free variables of s. □

Here are three examples to illustrate the matching algorithm.

Example 2.5. Let s be $\lambda x.(f\ x\ (g\ y\ z))$ and t be $\lambda z.(f\ z\ (g\ A\ B))$, where f, g, A, and B are constants with suitable signatures. Then the successive steps of the algorithm are as follows.

0.	$\lambda\underset{\uparrow}{x}.(f\ x\ (g\ y\ z))$	$\lambda\underset{\uparrow}{z}.(f\ z\ (g\ A\ B))$	
1.	$\lambda w.(f\ w\ (g\ \underset{\uparrow}{y}\ z))$	$\lambda w.(f\ w\ (g\ \underset{\uparrow}{A}\ B))$	$\{y/A\}$
2.	$\lambda w.(f\ w\ (g\ A\ \underset{\uparrow}{z}))$	$\lambda w.(f\ w\ (g\ A\ \underset{\uparrow}{B}))$	$\{z/B\}$
3.	$\lambda w.(f\ w\ (g\ A\ B))$	$\lambda w.(f\ w\ (g\ A\ B))$	

(The arrows indicate the points of disagreement and the substitutions in the last column are the substitutions applied at that step in the algorithm.) Thus $\lambda x.(f\ x\ (g\ y\ z))$ is matchable to $\lambda z.(f\ z\ (g\ A\ B))$ with matcher $\{y/A\} \circ \{z/B\}$.

Example 2.6. Let s be $(f\ x\ (g\ x))$ and t be $(f\ y\ (g\ A))$. Then the successive steps of the algorithm are as follows.

0.	$(f\ \underset{\uparrow}{x}\ (g\ x))$	$(f\ \underset{\uparrow}{y}\ (g\ A))$	$\{x/y\}$
1.	$(f\ y\ (g\ \underset{\uparrow}{y}))$	$(f\ y\ (g\ \underset{\uparrow}{A}))$	

Thus $(f\ x\ (g\ x))$ is not matchable to $(f\ y\ (g\ A))$, since there is a free occurrence of y in s to the left of the point of disagreement. Note that, in contrast, s and t are unifiable.

Example 2.7. Let s be $\lambda x.(f\ x\ y\ z)$ and t be $\lambda x.(f\ x\ A\ (g\ x))$. Then the successive steps of the algorithm are as follows.

0.	$\lambda x.(f\ x\ \underset{\uparrow}{y}\ z)$	$\lambda x.(f\ x\ \underset{\uparrow}{A}\ (g\ x))$	$\{y/A\}$
1.	$\lambda x.(f\ x\ A\ \underset{\uparrow}{z})$	$\lambda x.(f\ x\ A\ \underset{\uparrow}{(g\ x)})$	

Thus $\lambda x.(f\ x\ y\ z)$ is not matchable to $\lambda x.(f\ x\ A\ (g\ x))$, since x has a free occurrence in $(g\ x)$ but this occurrence is not free in $\lambda x.(f\ x\ A\ (g\ x))$.

2.9 Representation of Individuals

In this subsection, 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. 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 $\{1, 2\}$. 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 $[]_\alpha$, for all α.

In the following, let $\mathbb{N}$ denote the set of natural numbers.

Definition 2.30. The set of *default terms*, $\mathfrak{D}$, is defined inductively as follows.

1. If C is a default data constructor of arity n and $t_1, \ldots, t_n \in \mathfrak{D}$ $(n \in \mathbb{N})$ 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 $t_1, \ldots, t_n \in \mathfrak{D}$ $(n \in \mathbb{N})$, then $(t_1, \ldots, t_n) \in \mathfrak{D}$.

Definition 2.31. For each $\alpha \in \mathfrak{S}$, define $\mathfrak{D}_\alpha = \{t \in \mathfrak{D} \mid t \text{ has type } \alpha\}$.

Now normal terms can be defined.

Definition 2.32. 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 $t_1, \ldots, t_n \in \mathfrak{N}$ $(n \in \mathbb{N})$, 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 $T\ \alpha_1 \ldots \alpha_k$.

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 $\alpha_1 \times \cdots \times \alpha_n$.

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

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

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

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

   then

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

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

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 2.34. 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. Further illustrations of the use of basic terms are in [Llo03].

3 Semantics

This section gives the model theory for the logic.

3.1 Interpretations

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 $T\ \alpha_1 \ldots \alpha_k$, then $\mathcal{D}_\alpha = \{C\ d_1 \ldots d_n \mid C$ is a data constructor having signature $\sigma_1 \to \cdots \to \sigma_n \to (T\ \alpha_1 \ldots \alpha_k)$, and $d_i \in \mathcal{D}_{\sigma_i}$, for $i = 1, \ldots, n\}$.

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 $\alpha_1 \times \cdots \times \alpha_n$, for some $n \geq 0$, then $\mathcal{D}_\alpha$ is the cartesian product $\mathcal{D}_{\alpha_1} \times \cdots \times \mathcal{D}_{\alpha_n}$. 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. A *valuation* with respect to an augmented frame $\langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha \in \mathfrak{S}} \rangle$ is a mapping V that maps each pair consisting of a constant having signature α and a world in W to an element of $\mathcal{D}_\alpha$ (called the *denotation* of the constant in the world) such that if a constant C is rigid, then $V(C, w) = V(C, w')$, for each $w, w' \in W$.

Definition 3.5. 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 valuation with respect to this augmented frame such that the following conditions are satisfied.

1. For a data constructor $C : \sigma_1 \to \cdots \to \sigma_n \to (T\ \alpha_1 \ldots \alpha_k)$ and $w \in W$, $V(C, w)$ is the element of $\mathcal{D}_{\sigma_1 \to \cdots \to \sigma_n \to (T\ \alpha_1 \ldots \alpha_k)}$ defined by

   $$V(C, w)\ d_1 \ldots d_n = C\ d_1 \ldots d_n,$$

   where $d_i \in \mathcal{D}_{\sigma_i}$ $(i = 1, \ldots, n)$.

2. For $= : \alpha \to \alpha \to \Omega$ and $w \in W$, $V(=, w)$ is the element of $\mathcal{D}_{\alpha \to \alpha \to \Omega}$ defined by

   $$V(=, w)\ x\ y = \begin{cases} \top & \text{if } x = y \\ \bot & \text{otherwise,} \end{cases}$$

   for all $x, y \in \mathcal{D}_\alpha$.

3. For $\neg : \alpha \to \alpha$ and $w \in W$, $V(\neg, w)$ is the element of $\mathcal{D}_{\alpha \to \alpha}$ defined by

   $$V(\neg, w)\ f\ d_1\ \ldots\ d_n = \begin{cases} \top & \text{if } f\ d_1\ \ldots\ d_n = \bot \\ \bot & \text{otherwise,} \end{cases}$$

   for all $f \in \mathcal{D}_\alpha$ and $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$, where $\alpha \equiv \alpha_1 \to \cdots \to \alpha_n \to \Omega$, and $n \geq 0$.

4. For $\wedge : \alpha \to \alpha \to \alpha$ and $w \in W$, $V(\wedge, w)$ is the element of $\mathcal{D}_{\alpha \to \alpha \to \alpha}$ defined by

   $$V(\wedge, w)\ f\ g\ d_1\ \ldots\ d_n = \begin{cases} \top & \text{if } f\ d_1\ \ldots\ d_n = \top \text{ and } g\ d_1\ \ldots\ d_n = \top \\ \bot & \text{otherwise,} \end{cases}$$

   for all $f, g \in \mathcal{D}_\alpha$ and $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$, where $\alpha \equiv \alpha_1 \to \cdots \to \alpha_n \to \Omega$, and $n \geq 0$.

5. For $\vee : \alpha \to \alpha \to \alpha$ and $w \in W$, $V(\vee, w)$ is the element of $\mathcal{D}_{\alpha \to \alpha \to \alpha}$ defined by

   $$V(\vee, w)\ f\ g\ d_1\ \ldots\ d_n = \begin{cases} \top & \text{if } f\ d_1\ \ldots\ d_n = \top \text{ or } g\ d_1\ \ldots\ d_n = \top \\ \bot & \text{otherwise,} \end{cases}$$

   for all $f, g \in \mathcal{D}_\alpha$ and $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$, where $\alpha \equiv \alpha_1 \to \cdots \to \alpha_n \to \Omega$, and $n \geq 0$.

6. For $\longrightarrow : \alpha \to \alpha \to \alpha$ and $w \in W$, $V(\longrightarrow, w)$ is the element of $\mathcal{D}_{\alpha\to\alpha\to\alpha}$ defined by

$$V(\longrightarrow, w)\ f\ g\ d_1\ \ldots\ d_n = \begin{cases} \top & \text{if } f\ d_1\ \ldots\ d_n = \bot \text{ or } g\ d_1\ \ldots\ d_n = \top \\ \bot & \text{otherwise,} \end{cases}$$

for all $f, g \in \mathcal{D}_\alpha$ and $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$, where $\alpha \equiv \alpha_1 \to \cdots \to \alpha_n \to \Omega$, and $n \geq 0$.

7. For $\Sigma : \alpha \to \Omega$ and $w \in W$, $V(\Sigma, w)$ is the element of $\mathcal{D}_{\alpha\to\Omega}$ defined by

$$V(\Sigma, w)\ f = \begin{cases} \top & \text{if } f \neq \perp\!\!\!\perp \\ \bot & \text{otherwise,} \end{cases}$$

for all $f \in \mathcal{D}_\alpha$, where α is a biterm type.

8. For $\Pi : \alpha \to \Omega$ and $w \in W$, $V(\Pi, w)$ is the element of $\mathcal{D}_{\alpha\to\Omega}$ defined by

$$V(\Pi, w)\ f = \begin{cases} \top & \text{if } f = \top\!\!\!\top \\ \bot & \text{otherwise,} \end{cases}$$

for all $f \in \mathcal{D}_\alpha$, where α is a biterm type.

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]. It may seem restrictive to employ the constant domain semantics but it turns out that, for many applications, the issues of keeping track of function definitions that vary over time and vary from agent to agent completely overshadow the subtleties of whether objects may come into, or go out of, existence. In other words, for many applications, the simpler constant domain semantics easily suffices. In any case, it is shown in [FM98, p.105-107] that each semantics can simulate the other.

Definition 3.6. A *pointed interpretation* is a pair of the form (I, w), where $I \equiv \langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}, V\rangle$ is an interpretation and w a world in W.

3.2 Denotations

The next task is to define the denotation of a term with respect to an interpretation, world, and variable assignment. This is rather standard except for the case of modal terms. I begin with some intuition that shows how to proceed in this case.

If t is a formula, then the meaning of $\Box_i t$ in a world is $\top$ if the meaning of t in all accessible worlds is $\top$, its meaning is $\bot$ if the meaning of t in all accessible worlds is $\bot$, and, in the other cases, the meaning of $\Box_i t$ is conventionally defined to be $\bot$. This suggests an obvious extension to terms t whose type has rank 0: if t has the same meaning in all accessible worlds, then the meaning of $\Box_i t$ should be this common meaning; otherwise, the meaning of $\Box_i t$ should be some default value. This definition then becomes the base case of an inductive definition on the rank of the type of t of the semantics of a modal term $\Box_i t$. Here is the machinery to do this.

Let $\{X_i\}_{i=0}^n$ $(n \geq 0)$ be a family of non-empty sets and x_0 some distinguished element in X_0 that is called the *default value*. Let F be a set of functions, where

$$f : X_n \to (X_{n-1} \to (X_{n-2} \to \cdots \to X_0) \cdots),$$

for each $f \in F$. (If $n = 0$, the meaning is that each f belongs to X_0.)

Definition 3.7. Define $\mathcal{M}(F) : X_n \to (X_{n-1} \to (X_{n-2} \to \cdots \to X_0) \cdots)$ by induction on n as follows:

(a) If $n = 0$, then $\mathcal{M}(F) \in X_0$ is defined by

$$\mathcal{M}(F) = \begin{cases} x & \text{if there exists } x \in X_0, \text{ for each } f \in F,\ f = x \\ x_0 & \text{otherwise.} \end{cases}$$

(b) If $n > 0$, then $\mathcal{M}(F) : X_n \to (X_{n-1} \to (X_{n-2} \to \cdots \to X_0) \cdots)$ is defined by

$$\mathcal{M}(F)(x) = \mathcal{M}(\{f(x)\}_{f \in F}),$$

for each $x \in X_n$.

If there exists $f \in F$ such that $f = g$, for all $g \in F$, then clearly $\mathcal{M}(F) = f$.
The function $\mathcal{M}$ will be used to give meaning to modal terms.

Definition 3.8. Let α be a type of rank 0. Then the *default value* in $\mathcal{D}_\alpha$ is some distinguished element in $\mathcal{D}_\alpha$. In the case of $\mathcal{D}_\Omega$ the default value is $\bot$.

There are no restrictions on the choice of default values, other than the default value for $\mathcal{D}_\Omega$ is $\bot$. This choice is made for consistency with the standard semantics of modalities applied to formulas.

Definition 3.9. 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 $\mathcal{D}_\alpha$.

Now everything is in place to give the denotation of a term with respect to an interpretation, world and variable assignment.

Definition 3.10. 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. $\mathcal{V}(\Box_i t, I, w, \nu) = \mathcal{M}(\{\mathcal{V}(t, I, w', \nu)\}_{w' \in W'})$, where $W' = \{w' : w\, R_i\, w'\}$.

A special case of Condition 6 of Definition 3.10 is when the type of t is Ω (that is, t is a formula). Then the semantics for $\Box_i t$ given by Condition 6 is exactly the same as the standard semantics for a modal formula.

Definition 3.10 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 3.10 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 3.10, 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 $\mathcal{V}(t, I, w, \nu) \in \mathcal{D}_\alpha$.

The next three results establish (mostly) familiar properties of the connectives and quantifiers, but in the more general setting of biterms rather than formulas.

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 $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, φ and ψ biterms of type $\alpha_1 \to \cdots \to \alpha_n \to \Omega$, $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$, and t a term of type α_1. Then the following hold.*

1. $\mathcal{V}(\varphi\wedge\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$ *iff* $\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$ *and* $\mathcal{V}(\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$.
2. $\mathcal{V}(\varphi\vee\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$ *iff* $\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$ *or* $\mathcal{V}(\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$.
3. $\mathcal{V}(\varphi \longrightarrow \psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$ *iff* $\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \bot$ *or* $\mathcal{V}(\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$.
4. $\mathcal{V}(\neg\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$ *iff* $\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \bot$.
5. $\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$ *iff* $\mathcal{V}(\neg\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \bot$.
6. $\mathcal{V}(\neg\neg\varphi, I, w, \nu) = \mathcal{V}(\varphi, I, w, \nu)$.
7. $\mathcal{V}(\neg(\varphi \wedge \psi), I, w, \nu) = \mathcal{V}(\neg\varphi \vee \neg\psi, I, w, \nu)$.
8. $\mathcal{V}(\neg(\varphi \vee \psi), I, w, \nu) = \mathcal{V}(\neg\varphi \wedge \neg\psi, I, w, \nu)$.
9. $\mathcal{V}((\neg\varphi\ t), I, w, \nu) = \mathcal{V}(\neg(\varphi\ t), I, w, \nu)$.

10. $\mathcal{V}(\lambda x.\neg\varphi, I, w, \nu) = \mathcal{V}(\neg\lambda x.\varphi, I, w, \nu)$.

11. $\mathcal{V}(\lambda x.\varphi \wedge \lambda x.\psi, I, w, \nu) = \mathcal{V}(\lambda x.(\varphi \wedge \psi), I, w, \nu)$.

12. $\mathcal{V}(\lambda x.\varphi \vee \lambda x.\psi, I, w, \nu) = \mathcal{V}(\lambda x.(\varphi \vee \psi), I, w, \nu)$.

13. $\mathcal{V}(\lambda x.\varphi \longrightarrow \lambda x.\psi, I, w, \nu) = \mathcal{V}(\lambda x.(\varphi \longrightarrow \psi), I, w, \nu)$.

Proof. 1.

$$
\begin{aligned}
&\mathcal{V}(\varphi \wedge \psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top \\
\text{iff}\ &\mathcal{V}((\wedge\ \varphi), I, w, \nu)(\mathcal{V}(\psi, I, w, \nu))\ d_1\ \ldots\ d_n = \top \\
\text{iff}\ &(\mathcal{V}(\wedge, I, w, \nu)(\mathcal{V}(\varphi, I, w, \nu)))(\mathcal{V}(\psi, I, w, \nu))\ d_1\ \ldots\ d_n = \top \\
\text{iff}\ &(V(\wedge, w)(\mathcal{V}(\varphi, I, w, \nu)))(\mathcal{V}(\psi, I, w, \nu))\ d_1\ \ldots\ d_n = \top \\
\text{iff}\ &\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top \text{ and } \mathcal{V}(\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top.
\end{aligned}
$$

2.

$$
\begin{aligned}
&\mathcal{V}(\varphi \vee \psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top \\
\text{iff}\ &\mathcal{V}((\vee\ \varphi), I, w, \nu)(\mathcal{V}(\psi, I, w, \nu))\ d_1\ \ldots\ d_n = \top \\
\text{iff}\ &(\mathcal{V}(\vee, I, w, \nu)(\mathcal{V}(\varphi, I, w, \nu)))(\mathcal{V}(\psi, I, w, \nu))\ d_1\ \ldots\ d_n = \top \\
\text{iff}\ &(V(\vee, w)(\mathcal{V}(\varphi, I, w, \nu)))(\mathcal{V}(\psi, I, w, \nu))\ d_1\ \ldots\ d_n = \top \\
\text{iff}\ &\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top \text{ or } \mathcal{V}(\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top.
\end{aligned}
$$

3, Similar to the proof of Part 2.

4.

$$
\begin{aligned}
&\mathcal{V}(\neg\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top \\
\text{iff}\ &\mathcal{V}(\neg, I, w, \nu)(\mathcal{V}(\varphi, I, w, \nu))\ d_1\ \ldots\ d_n = \top \\
\text{iff}\ &V(\neg, w)(\mathcal{V}(\varphi, I, w, \nu))\ d_1\ \ldots\ d_n = \top \\
\text{iff}\ &\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \bot.
\end{aligned}
$$

5. Similar to the proof of Part 4.

6.

$$
\begin{aligned}
&\mathcal{V}(\neg\neg\varphi, I, w, \nu) \\
&= \mathcal{V}(\neg, I, w, \nu)(\mathcal{V}(\neg\varphi, I, w, \nu)) \\
&= V(\neg, w)(\mathcal{V}(\neg\varphi, I, w, \nu)) \\
&= V(\neg, w)(V(\neg, w)(\mathcal{V}(\varphi, I, w, \nu))) \\
&= \mathcal{V}(\varphi, I, w, \nu) \quad \text{[definition of } V(\neg, w)\text{]}.
\end{aligned}
$$

7.

$$
\begin{aligned}
&\mathcal{V}(\neg(\varphi \wedge \psi), I, w, \nu)\ d_1\ \ldots\ d_n \\
&= V(\neg, w)(\mathcal{V}(\varphi \wedge \psi, I, w, \nu))\ d_1\ \ldots\ d_n \\
&= \begin{cases} \top & \text{if } \mathcal{V}(\varphi \wedge \psi, I, w, \nu)\ d_1\ \ldots\ d_n = \bot \\ \bot & \text{otherwise} \end{cases} \\
&= \begin{cases} \top & \text{if } (V(\wedge, w)(\mathcal{V}(\varphi, I, w, \nu)))(\mathcal{V}(\psi, I, w, \nu))\ d_1\ \ldots\ d_n = \bot \\ \bot & \text{otherwise} \end{cases} \\
&= \begin{cases} \top & \text{if } \mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \bot \text{ or } \mathcal{V}(\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \bot \\ \bot & \text{otherwise} \end{cases} \\
&= \begin{cases} \top & \text{if } V(\neg, w)(\mathcal{V}(\varphi, I, w, \nu))\ d_1\ \ldots\ d_n = \top \text{ or } V(\neg, w)(\mathcal{V}(\psi, I, w, \nu))\ d_1\ \ldots\ d_n = \top \\ \bot & \text{otherwise} \end{cases} \\
&= (V(\vee, w)(V(\neg, w)(\mathcal{V}(\varphi, I, w, \nu))))(V(\neg, w)(\mathcal{V}(\psi, I, w, \nu)))\ d_1\ \ldots\ d_n \\
&= (V(\vee, w)(\mathcal{V}(\neg\varphi, I, w, \nu)))(\mathcal{V}(\neg\psi, I, w, \nu))\ d_1\ \ldots\ d_n \\
&= \mathcal{V}(\neg\varphi \vee \neg\psi, I, w, \nu)\ d_1\ \ldots\ d_n.
\end{aligned}
$$

Hence $\mathcal{V}(\neg(\varphi \wedge \psi), I, w, \nu) = \mathcal{V}(\neg\varphi \vee \neg\psi, I, w, \nu)$.

8. Similar to the proof of Part 7.

9.

$$
\begin{aligned}
&\mathcal{V}((\neg\varphi\ t), I, w, \nu) \\
&= \mathcal{V}(\neg\varphi, I, w, \nu)(\mathcal{V}(t, I, w, \nu)) \\
&= (\mathcal{V}(\neg, I, w, \nu)(\mathcal{V}(\varphi, I, w, \nu)))(\mathcal{V}(t, I, w, \nu)) \\
&= (V(\neg, w)(\mathcal{V}(\varphi, I, w, \nu)))(\mathcal{V}(t, I, w, \nu)) \\
&= V(\neg, w)(\mathcal{V}(\varphi, I, w, \nu)(\mathcal{V}(t, I, w, \nu))) \quad \text{[definition of } V(\neg, w)\text{]} \\
&= V(\neg, w)(\mathcal{V}((\varphi\ t), I, w, \nu)) \\
&= \mathcal{V}(\neg(\varphi\ t), I, w, \nu).
\end{aligned}
$$

10.

$$
\begin{aligned}
&\mathcal{V}(\lambda x.\neg\varphi, I, w, \nu) \\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(\neg\varphi, I, w, \nu'), \\
&\quad \text{where } x \text{ has type } \alpha \text{ and } \nu' \text{ is } \nu \text{ except } \nu'(x) = d \\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } V(\neg, w)(\mathcal{V}(\varphi, I, w, \nu')) \\
&= V(\neg, w)(\text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(\varphi, I, w, \nu')) \\
&= V(\neg, w)(\mathcal{V}(\lambda x.\varphi, I.w, \nu)) \\
&= \mathcal{V}(\neg\lambda x.\varphi, I, w, \nu).
\end{aligned}
$$

11.

$$
\begin{aligned}
&\mathcal{V}(\lambda x.\varphi \wedge \lambda x.\psi, I, w, \nu) \\
&= \mathcal{V}((\wedge\ \lambda x.\varphi), I, w, \nu)(\mathcal{V}(\lambda x.\psi, I, w, \nu))
\end{aligned}
$$

$= (V(\wedge, w)(\mathcal{V}(\lambda x.\varphi, I, w, \nu)))(\mathcal{V}(\lambda x.\psi, I, w, \nu))$
$= (V(\wedge, w)(\text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(\varphi, I, w, \nu')))$
(the function whose value for each $d \in \mathcal{D}_\alpha$ is $\mathcal{V}(\psi, I, w, \nu'))$,
where x has type α and ν' is ν except $\nu'(x) = d$
$=$ the function whose value for each $d \in \mathcal{D}_\alpha$ is $(V(\wedge, w)(\mathcal{V}(\varphi, I, w, \nu')))(\mathcal{V}(\psi, I, w, \nu'))$
$=$ the function whose value for each $d \in \mathcal{D}_\alpha$ is $\mathcal{V}(\varphi \wedge \psi, I, w, \nu')$
$= \mathcal{V}(\lambda x.(\varphi \wedge \psi), I, w, \nu)$.

12. Similar to the proof of Part 11.
13. Similar to the proof of Part 11. □

Proposition 3.2. *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 $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, and φ a biterm having type $\alpha_1 \to \cdots \to \alpha_n \to \Omega$. Then the following hold.*

1. $\mathcal{V}((\Sigma\ \varphi), I, w, \nu) = \top$ *iff, for some* $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$, $\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$.
2. $\mathcal{V}((\Pi\ \varphi), I, w, \nu) = \top$ *iff, for each* $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$, $\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$.

Proof. 1.

$\mathcal{V}((\Sigma\ \varphi), I, w, \nu) = \top$
iff $V(\Sigma, w)(\mathcal{V}(\varphi, I, w, \nu)) = \top$
iff $\mathcal{V}(\varphi, I, w, \nu) \neq \perp\!\!\!\perp$
iff $\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$, for some $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$.

2.

$\mathcal{V}((\Pi\ \varphi), I, w, \nu) = \top$
iff $V(\Pi, w)(\mathcal{V}(\varphi, I, w, \nu)) = \top$
iff $\mathcal{V}(\varphi, I, w, \nu) = \mathbb{T}$
iff $\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$, for each $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$.

□

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, ν a variable assignment with respect to $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, and φ a biterm. Then the following hold.*

1. $\mathcal{V}(\neg(\Sigma\ \varphi), I, w, \nu) = \mathcal{V}((\Pi\ \neg\varphi), I, w, \nu)$.
2. $\mathcal{V}(\neg(\Pi\ \varphi), I, w, \nu) = \mathcal{V}((\Sigma\ \neg\varphi), I, w, \nu)$.

Proof. 1.

$$\begin{aligned}
&\mathcal{V}(\neg(\Sigma\ \varphi), I, w, \nu)\\
&= V(\neg, w)(\mathcal{V}((\Sigma\ \varphi), I, w, \nu))\\
&= V(\neg, w)(V(\Sigma, w)(\mathcal{V}(\varphi, I, w, v)))\\
&= V(\neg, w)(\text{if } \mathcal{V}(\varphi, I, w, v) \neq \perp\!\!\!\perp \text{ then } \top \text{ else } \bot)\\
&= \text{if } \mathcal{V}(\varphi, I, w, v) \neq \perp\!\!\!\perp \text{ then } \bot \text{ else } \top\\
&= \text{if } \mathcal{V}(\varphi, I, w, v) = \perp\!\!\!\perp \text{ then } \top \text{ else } \bot\\
&= \text{if } V(\neg, w)(\mathcal{V}(\varphi, I, w, v)) = \mathbb{T} \text{ then } \top \text{ else } \bot\\
&= \text{if } \mathcal{V}(\neg\varphi, I, w, v) = \mathbb{T} \text{ then } \top \text{ else } \bot\\
&= V(\Pi, w)(\mathcal{V}(\neg\varphi, I, w, v))\\
&= \mathcal{V}((\Pi\ \neg\varphi), I, w, \nu).
\end{aligned}$$

2.

$$\begin{aligned}
&\mathcal{V}(\neg(\Pi\ \varphi), I, w, \nu)\\
&= V(\neg, w)(\mathcal{V}((\Pi\ \varphi), I, w, \nu))\\
&= V(\neg, w)(V(\Pi, w)(\mathcal{V}(\varphi, I, w, v)))\\
&= V(\neg, w)(\text{if } \mathcal{V}(\varphi, I, w, v) = \mathbb{T} \text{ then } \top \text{ else } \bot)\\
&= \text{if } \mathcal{V}(\varphi, I, w, v) = \mathbb{T} \text{ then } \bot \text{ else } \top\\
&= \text{if } \mathcal{V}(\varphi, I, w, v) \neq \mathbb{T} \text{ then } \top \text{ else } \bot\\
&= \text{if } V(\neg, w)(\mathcal{V}(\varphi, I, w, v)) \neq \perp\!\!\!\perp \text{ then } \top \text{ else } \bot\\
&= \text{if } \mathcal{V}(\neg\varphi, I, w, v) \neq \perp\!\!\!\perp \text{ then } \top \text{ else } \bot\\
&= V(\Sigma, w)(\mathcal{V}(\neg\varphi, I, w, v))\\
&= \mathcal{V}((\Sigma\ \neg\varphi), I, w, \nu).
\end{aligned}$$

□

Proposition 3.4. *Let $I \equiv \langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}, V\rangle$ be an interpretation, $w \in W$, φ a biterm, and x a variable that is not free in φ. Then $\mathcal{V}((\varphi\ x), I, w, \nu) = \mathbb{T}$, for every variable assignment ν, iff $\mathcal{V}(\varphi, I, w, \nu) = \mathbb{T}$, for every variable assignment ν.*

Proof.

$$\begin{aligned}
&\mathcal{V}((\varphi\ x), I, w, \nu) = \mathbb{T}, \text{ for every variable assignment } \nu\\
\text{iff } &\mathcal{V}(\varphi, I, w, \nu)(\nu(x)) = \mathbb{T}, \text{ for every variable assignment } \nu\\
\text{iff } &\mathcal{V}(\varphi, I, w, \nu) = \mathbb{T}, \text{ for every variable assignment } \nu \quad [x \text{ is not free in } \varphi].
\end{aligned}$$

□

Definition 3.11. The variable assignments ν and ν' are said to be *x-variants* if they agree on all variables except possibly the variable x.

Proposition 3.5. *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 $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, and φ a biterm. Then the following hold.*

1. $\mathcal{V}(\lambda x.\varphi, I, w, \nu) = \mathbb{T}$ *iff, for every* x*-variant* ν' *of* ν, $\mathcal{V}(\varphi, I, w, \nu') = \mathbb{T}$.
2. $\mathcal{V}(\lambda x.\varphi, I, w, \nu) = \perp\!\!\!\perp$ *iff, for every* x*-variant* ν' *of* ν, $\mathcal{V}(\varphi, I, w, \nu') = \perp\!\!\!\perp$.
3. $\mathcal{V}(\forall x.\varphi, I, w, \nu) = \top$ *iff, for every* x*-variant* ν' *of* ν, $\mathcal{V}(\varphi, I, w, \nu') = \mathbb{T}$.
4. $\mathcal{V}(\exists x.\varphi, I, w, \nu) = \top$ *iff, for some* x*-variant* ν' *of* ν, $\mathcal{V}(\varphi, I, w, \nu') \neq \perp\!\!\!\perp$.
5. $\mathcal{V}(\neg\exists x.\varphi, I, w, \nu) = \mathcal{V}(\forall x.\neg\varphi, I, w, \nu)$.
6. $\mathcal{V}(\neg\forall x.\varphi, I, w, \nu) = \mathcal{V}(\exists x.\neg\varphi, I, w, \nu)$.
7. $\mathcal{V}(\forall x.\varphi, I, w, \nu) = \mathcal{V}(\neg\exists x.\neg\varphi, I, w, \nu)$.
8. $\mathcal{V}(\exists x.\varphi, I, w, \nu) = \mathcal{V}(\neg\forall x.\neg\varphi, I, w, \nu)$.

Proof. 1.

$\mathcal{V}(\lambda x.\varphi, I, w, \nu) = \mathbb{T}$
iff $\mathcal{V}(\varphi, I, w, \nu') = \mathbb{T}$, for every x-variant ν' of ν [Semantics of $\lambda x.\varphi$].

2.

$\mathcal{V}(\lambda x.\varphi, I, w, \nu) = \perp\!\!\!\perp$
iff $\mathcal{V}(\varphi, I, w, \nu') = \perp\!\!\!\perp$, for every x-variant ν' of ν [Semantics of $\lambda x.\varphi$].

3.

$\mathcal{V}(\forall x.\varphi, I, w, \nu) = \top$
iff $V(\Pi, w)(\mathcal{V}(\lambda x.\varphi, I, w, \nu)) = \top$
iff $\mathcal{V}(\lambda x.\varphi, I, w, \nu) = \mathbb{T}$
iff $\mathcal{V}(\varphi, I, w, \nu') = \mathbb{T}$, for every x-variant ν' of ν [Part 1].

4.

$\mathcal{V}(\exists x.\varphi, I, w, \nu) = \top$
iff $V(\Sigma, w)(\mathcal{V}(\lambda x.\varphi, I, w, \nu)) = \top$
iff $\mathcal{V}(\lambda x.\varphi, I, w, \nu) \neq \perp\!\!\!\perp$
iff $\mathcal{V}(\varphi, I, w, \nu') \neq \perp\!\!\!\perp$, for some x-variant ν' of ν [Part 2].

5.

$\mathcal{V}(\neg\exists x.\varphi, I, w, \nu)$
$= \mathcal{V}(\neg(\Sigma\ \lambda x.\varphi), I, w, \nu)$
$= \mathcal{V}((\Pi\ \neg\lambda x.\varphi), I, w, \nu)$ [Part 1 of Proposition 3.3]
$= V(\Pi, w)(\mathcal{V}(\neg\lambda x.\varphi, I, w, \nu))$
$= V(\Pi, w)(\mathcal{V}(\lambda x.\neg\varphi, I, w, \nu))$ [Part 10 of Proposition 3.1]
$= \mathcal{V}((\Pi\ \lambda x.\neg\varphi), I, w, \nu)$
$= \mathcal{V}(\forall x.\neg\varphi, I, w, \nu)$.

6.

$$\begin{aligned}
&\mathcal{V}(\neg\forall x.\varphi, I, w, \nu)\\
&= \mathcal{V}(\neg(\Pi\ \lambda x.\varphi), I, w, \nu)\\
&= \mathcal{V}((\Sigma\ \neg\lambda x.\varphi), I, w, \nu) \quad \text{[Part 2 of Proposition 3.3]}\\
&= V(\Sigma, w)(\mathcal{V}(\neg\lambda x.\varphi, I, w, \nu))\\
&= V(\Sigma, w)(\mathcal{V}(\lambda x.\neg\varphi, I, w, \nu)) \quad \text{[Part 10 of Proposition 3.1]}\\
&= \mathcal{V}((\Sigma\ \lambda x.\neg\varphi), I, w, \nu)\\
&= \mathcal{V}(\exists x.\neg\varphi, I, w, \nu).
\end{aligned}$$

7. This part follows from Part 6 of Proposition 3.1 and Part 5.
8. This part follows from Part 6 of Proposition 3.1 and Part 6.

□

Now come some useful properties of modalities.

Proposition 3.6. *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 $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, $i \in \{1, \ldots, m\}$, $d_j \in \mathcal{D}_{\alpha_j} (j = 1, \ldots, n)$, and φ and ψ biterms having type $\alpha_1 \to \cdots \to \alpha_n \to \Omega$. Then the following hold.*

1. *$\mathcal{V}(\Box_i\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$ iff, for each w' such that $w\, R_i\, w'$, $\mathcal{V}(\varphi, I, w', \nu)\ d_1\ \ldots\ d_n = \top$.*
2. *$\mathcal{V}(\Diamond_i\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top$ iff, for some w' such that $w\, R_i\, w'$, $\mathcal{V}(\varphi, I, w', \nu)\ d_1\ \ldots\ d_n = \top$.*
3. *$\mathcal{V}(\Box_i\varphi, I, w, \nu) = \mathbb{T}$ iff, for each w' such that $w\, R_i\, w'$, $\mathcal{V}(\varphi, I, w', \nu) = \mathbb{T}$.*
4. *$\mathcal{V}(\Diamond_i\varphi, I, w, \nu) = \mathbb{T}$ iff, for some w' such that $w\, R_i\, w'$, $\mathcal{V}(\varphi, I, w', \nu) = \mathbb{T}$.*
5. *$\mathcal{V}(\Box_i(\varphi \wedge \psi), I, w, \nu) = \mathcal{V}(\Box_i\varphi \wedge \Box_i\psi, I, w, \nu)$.*
6. *$\mathcal{V}(\Diamond_i(\varphi \vee \psi), I, w, \nu) = \mathcal{V}(\Diamond_i\varphi \vee \Diamond_i\psi, I, w, \nu)$.*

Proof. 1.

$$\begin{aligned}
&\mathcal{V}(\Box_i\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top\\
\text{iff}\ &\mathcal{M}(\{\mathcal{V}(\varphi, I, w', \nu)\}_{w'\in W'})\ d_1\ \ldots\ d_n = \top\\
\text{iff}\ &\mathcal{V}(\varphi, I, w', \nu)\ d_1\ \ldots\ d_n = \top, \text{ for each } w' \text{ such that } w\, R_i\, w'.
\end{aligned}$$

2.

$$\begin{aligned}
&\mathcal{V}(\Diamond_i\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top\\
\text{iff}\ &\mathcal{V}(\neg\Box_i\neg\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top\\
\text{iff}\ &\mathcal{V}(\Box_i\neg\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \bot\\
\text{iff}\ &\mathcal{M}(\{\mathcal{V}(\neg\varphi, I, w', \nu)\}_{w'\in W'})\ d_1\ \ldots\ d_n = \bot\\
\text{iff}\ &\mathcal{V}(\neg\varphi, I, w', \nu)\ d_1\ \ldots\ d_n = \bot, \text{ for some } w' \text{ such that } w\, R_i\, w'.\\
\text{iff}\ &\mathcal{V}(\varphi, I, w', \nu)\ d_1\ \ldots\ d_n = \top, \text{ for some } w' \text{ such that } w\, R_i\, w'.
\end{aligned}$$

3.

$$\begin{aligned}
&\mathcal{V}(\Box_i\varphi, I, w, \nu) \;=\; \mathbb{T} \\
\text{iff }\; &\mathcal{M}(\{\mathcal{V}(\varphi, I, w', \nu)\}_{w' \in W'}) \;=\; \mathbb{T} \\
\text{iff }\; &\mathcal{V}(\varphi, I, w', \nu) \;=\; \mathbb{T}, \text{ for each } w' \text{ such that } w\, R_i\, w'.
\end{aligned}$$

4.

$$\begin{aligned}
&\mathcal{V}(\Diamond_i\varphi, I, w, \nu) \;=\; \mathbb{T} \\
\text{iff }\; &\mathcal{V}(\neg\Box_i\neg\varphi, I, w, \nu) \;=\; \mathbb{T} \\
\text{iff }\; &\mathcal{V}(\Box_i\neg\varphi, I, w, \nu) \;=\; \mathbb{\perp\!\!\!\perp} \\
\text{iff }\; &\text{for some } w' \text{ such that } w\, R_i\, w',\ \mathcal{V}(\neg\varphi, I, w', \nu) \;=\; \mathbb{\perp\!\!\!\perp} \\
\text{iff }\; &\text{for some } w' \text{ such that } w\, R_i\, w',\ \mathcal{V}(\varphi, I, w', \nu) \;=\; \mathbb{T}.
\end{aligned}$$

5.

$$\begin{aligned}
&\mathcal{V}(\Box_i(\varphi \wedge \psi), I, w, \nu)\ d_1\ \ldots\ d_n = \top \\
\text{iff }\; &\mathcal{V}(\varphi \wedge \psi, I, w', \nu)\ d_1\ \ldots\ d_n = \top, \text{ for each } w' \text{ such that } w\, R_i\, w' \quad \text{[Part 1]} \\
\text{iff }\; &\mathcal{V}(\varphi, I, w', \nu)\ d_1\ \ldots\ d_n = \top \text{ and } \mathcal{V}(\psi, I, w', \nu)\ d_1\ \ldots\ d_n = \top, \\
&\quad \text{for each } w' \text{ such that } w\, R_i\, w' \\
\text{iff }\; &\mathcal{V}(\Box_i\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top \text{ and } \mathcal{V}(\Box_i\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top \quad \text{[Part 1]} \\
\text{iff }\; &\mathcal{V}(\Box_i\varphi \wedge \Box_i\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top.
\end{aligned}$$

Hence $\mathcal{V}(\Box_i(\varphi \wedge \psi), I, w, \nu) = \mathcal{V}(\Box_i\varphi \wedge \Box_i\psi, I, w, \nu)$

6.

$$\begin{aligned}
&\mathcal{V}(\Diamond_i(\varphi \vee \psi), I, w, \nu)\ d_1\ \ldots\ d_n = \top \\
\text{iff }\; &\mathcal{V}(\varphi \vee \psi, I, w', \nu)\ d_1\ \ldots\ d_n = \top, \text{ for some } w' \text{ such that } w\, R_i\, w' \quad \text{[Part 2]} \\
\text{iff }\; &\mathcal{V}(\varphi, I, w', \nu)\ d_1\ \ldots\ d_n = \top \text{ or } \mathcal{V}(\psi, I, w', \nu)\ d_1\ \ldots\ d_n = \top, \\
&\quad \text{for some } w' \text{ such that } w\, R_i\, w' \\
\text{iff }\; &\mathcal{V}(\Diamond_i\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top \text{ or } \mathcal{V}(\Diamond_i\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top \quad \text{[Part 2]} \\
\text{iff }\; &\mathcal{V}(\Diamond_i\varphi \vee \Diamond_i\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top.
\end{aligned}$$

Hence $\mathcal{V}(\Diamond_i(\varphi \vee \psi), I, w, \nu) = \mathcal{V}(\Diamond_i\varphi \vee \Diamond_i\psi, I, w, \nu)$

□

Proposition 3.7. *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 $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, t be a term of type α having rank 0 and d a non-default element of $\mathcal{D}_\alpha$. Then $\mathcal{V}(\Box_{j_1}\cdots\Box_{j_n}t, I, w, \nu) = d$ iff, for each w' such that $w\, R_{j_1} \circ \cdots \circ R_{j_n}\, w'$, $\mathcal{V}(t, I, w', \nu) = d$.*

Proof. The proof is by induction on the length of the sequence of modalities. If $n = 0$, the result is obvious.

Suppose now the result holds for sequences of modalities of length n. Then

$\mathcal{V}(\Box_{j_1} \cdots \Box_{j_{n+1}} t, I, w, \nu) = d$
iff for each w'' such that $w \, R_{j_1} \, w''$, $\mathcal{V}(\Box_{j_2} \cdots \Box_{j_{n+1}} t, I, w'', \nu) = d$
iff for each w'' such that $w \, R_{j_1} \, w''$, for each w' such that $w'' \, R_{j_2} \circ \cdots \circ R_{j_{n+1}} \, w'$, $\mathcal{V}(t, I, w', \nu) = d$ [induction hypothesis]
iff for each w' such that $w \, R_{j_1} \circ \cdots \circ R_{j_{n+1}} \, w'$, $\mathcal{V}(t, I, w', \nu) = d$.

□

The next result shows that $\Box_i$ and Π can be 'switched', as can $\Diamond_i$ and Σ.

Proposition 3.8. *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 $\{\mathcal{D}_\alpha\}_{\alpha \in \mathfrak{S}}$, $w \in W$, φ a biterm, and $i \in \{1, \ldots, m\}$. Then the following hold.*

1. $\mathcal{V}(\Box_i(\Pi \, \varphi), I, w, \nu) = \mathcal{V}((\Pi \, \Box_i \varphi), I, w, \nu)$.
2. $\mathcal{V}(\Diamond_i \Sigma \, \varphi), I, w, \nu) = \mathcal{V}((\Sigma \, \Diamond_i \varphi), I, w, \nu)$.

Proof. Suppose that φ has type $\alpha_1 \to \cdots \to \alpha_n \to \Omega$.
1.

$\mathcal{V}(\Box_i(\Pi \, \varphi), I, w, \nu) = \top$
iff $\mathcal{V}((\Pi \, \varphi), I, w', \nu) = \top$, for each w' such that $w \, R_i \, w'$ [Proposition 3.6]
iff $\mathcal{V}(\varphi, I, w', \nu) \; d_1 \; \ldots \; d_n = \top$,
for each w' such that $w \, R_i \, w'$ and each $d_j \in \mathcal{D}_{\alpha_j} (j = 1, \ldots, n)$ [Proposition 3.2]
iff $\mathcal{V}(\Box_i \varphi, I, w, \nu) \; d_1 \; \ldots \; d_n = \top$, for each $d_j \in \mathcal{D}_{\alpha_j} (j = 1, \ldots, n)$ [Proposition 3.6]
iff $\mathcal{V}((\Pi \, \Box_i \varphi), I, w, \nu) = \top$ [Proposition 3.2].

Hence $\mathcal{V}(\Box_i(\Pi \, \varphi), I, w, \nu) = \mathcal{V}((\Pi \, \Box_i \varphi), I, w, \nu)$.
2. The proof is similar to Part 1.

□

Semantic versions of the Barcan and converse Barcan formulas [FM98, p.108] are now established, in the biterm setting. (So they are called the Barcan biterm and converse Barcan biterm.)

Proposition 3.9. *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 $\{\mathcal{D}_\alpha\}_{\alpha \in \mathfrak{S}}$, $w \in W$, φ a biterm, and $i \in \{1, \ldots, m\}$. Then the following hold.*

1. $\mathcal{V}(\Box_i \forall x.\varphi, I, w, \nu) = \mathcal{V}(\forall x. \Box_i \varphi, I, w, \nu)$.
2. $\mathcal{V}(\Diamond_i \exists x.\varphi, I, w, \nu) = \mathcal{V}(\exists x. \Diamond_i \varphi, I, w, \nu)$.

Proof. Suppose that φ has type $\alpha_1 \to \cdots \to \alpha_n \to \Omega$ and x has type α.

1.

$$
\begin{aligned}
&\mathcal{V}(\Box_i \forall x.\varphi, I, w, \nu) \\
&= \mathcal{M}(\{\mathcal{V}(\forall x.\varphi, I, w', \nu)\}_{w' \in W'}) \\
&= \mathcal{M}(\{\mathcal{V}((\Pi\ \lambda x.\varphi), I, w', \nu)\}_{w' \in W'}) \\
&= \mathcal{M}(\{V(\Pi, w')(\mathcal{V}(\lambda x.\varphi, I, w', \nu))\}_{w' \in W'}) \\
&= \mathcal{M}(\{\text{if } \mathcal{V}(\lambda x.\varphi, I, w', \nu) = \mathbb{T} \text{ then } \top \text{ else } \bot\}_{w' \in W'}), \\
&\quad \text{where } \mathbb{T} : \mathcal{D}_\alpha \to \mathcal{D}_{\alpha_1} \to \cdots \to \mathcal{D}_{\alpha_n} \to \mathcal{D}_\Omega \\
&= \mathcal{M}(\{\text{if } \mathcal{V}(\varphi, I, w', \nu') = \mathbb{T}, \text{ for all } d \in \mathcal{D}_\alpha, \text{ then } \top \text{ else } \bot\}_{w' \in W'}), \\
&\quad \text{where } \mathbb{T} : \mathcal{D}_{\alpha_1} \to \cdots \to \mathcal{D}_{\alpha_n} \to \mathcal{D}_\Omega \text{ and } \nu' \text{ is } \nu \text{ except } \nu'(x) = d \\
&= \text{if } \mathcal{M}(\{\mathcal{V}(\varphi, I, w', \nu')\}_{w' \in W'}) = \mathbb{T}, \text{ for all } d \in \mathcal{D}_\alpha, \text{ then } \top \text{ else } \bot \\
&= \text{if } \mathcal{V}(\Box_i \varphi, I, w, \nu') = \mathbb{T}, \text{ for all } d \in \mathcal{D}_\alpha, \text{ then } \top \text{ else } \bot \\
&= \text{if } \mathcal{V}(\lambda x.\Box_i \varphi, I, w, \nu) = \mathbb{T} \text{ then } \top \text{ else } \bot, \\
&\quad \text{where } \mathbb{T} : \mathcal{D}_\alpha \to \mathcal{D}_{\alpha_1} \to \cdots \to \mathcal{D}_{\alpha_n} \to \mathcal{D}_\Omega \\
&= V(\Pi, w)(\mathcal{V}(\lambda x.\Box_i \varphi, I, w, \nu)) \\
&= \mathcal{V}((\Pi\ \lambda x.\Box_i \varphi), I, w, \nu) \\
&= \mathcal{V}(\forall x.\Box_i \varphi, I, w, \nu).
\end{aligned}
$$

2.

$$
\begin{aligned}
&\mathcal{V}(\Diamond_i \exists x.\varphi, I, w, \nu) \\
&= \mathcal{V}(\neg \Box_i \neg \exists x.\varphi, I, w, \nu) \\
&= V(\neg, w)(\mathcal{V}(\Box_i \neg \exists x.\varphi, I, w, \nu)) \\
&= V(\neg, w)(\mathcal{M}(\{\mathcal{V}(\neg \exists x.\varphi, I, w', \nu)\}_{w' \in W'})) \\
&= V(\neg, w)(\mathcal{M}(\{\mathcal{V}(\forall x.\neg \varphi, I, w', \nu)\}_{w' \in W'})) \quad \text{[Part 5 of Proposition 3.5]} \\
&= V(\neg, w)(\mathcal{V}(\Box_i \forall x.\neg \varphi, I, w, \nu)) \\
&= V(\neg, w)(\mathcal{V}(\forall x.\Box_i \neg \varphi, I, w, \nu)) \quad \text{[Part 1]} \\
&= \mathcal{V}(\neg \forall x.\Box_i \neg \varphi, I, w, \nu) \\
&= \mathcal{V}(\exists x.\neg \Box_i \neg \varphi, I, w, \nu) \quad \text{[Part 6 of Proposition 3.5]} \\
&= \mathcal{V}(\exists x.\Diamond_i \varphi, I, w, \nu).
\end{aligned}
$$

□

Proposition 3.10. *Let s and t be terms of the same type, I an interpretation, w a world in I, and ν a variable assignment. Then $\mathcal{V}(s = t, I, w, \nu) = \top$ iff $\mathcal{V}(s, I, w, \nu) = \mathcal{V}(t, I, w, \nu)$.*

Proof.

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

□

The bi-implication connective $\longleftrightarrow$: $\alpha \to \alpha \to \alpha$, where α is a biterm type, can be introduced by the abbreviation $\varphi \longleftrightarrow \psi \equiv (\varphi \longrightarrow \psi) \wedge (\psi \longrightarrow \varphi)$. The next result implies that, applied to formulas, $\longleftrightarrow$ is just equality.

Proposition 3.11. *Let φ and ψ be biterms of type $\alpha_1 \to \cdots \to \alpha_n \to \Omega$, I an interpretation, w a world in I, and ν a variable assignment. Then $\mathcal{V}(\varphi \longleftrightarrow \psi, I, w, \nu) = \mathbb{T}$ iff $\mathcal{V}(\varphi = \psi, I, w, \nu) = \top$.*

Proof.

$\mathcal{V}((\varphi \longrightarrow \psi) \wedge (\psi \longrightarrow \varphi), I, w, \nu) = \mathbb{T}$
iff $(V(\wedge, w)(\mathcal{V}(\varphi \longrightarrow \psi, I, w, \nu)))(\mathcal{V}(\psi \longrightarrow \varphi, I, w, \nu)) = \mathbb{T}$
iff $\mathcal{V}(\varphi \longrightarrow \psi, I, w, \nu) = \mathbb{T}$ and $\mathcal{V}(\psi \longrightarrow \varphi, I, w, \nu) = \mathbb{T}$
iff $(\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \bot$ or $\mathcal{V}(\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top)$ and
$(\mathcal{V}(\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \bot$ or $\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top)$,
for each $d_i \in \mathcal{D}_{\alpha_i} (i = 1, \ldots, n)$
iff $\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \mathcal{V}(\psi, I, w, \nu)\ d_1\ \ldots\ d_n$,
for each $d_i \in \mathcal{D}_{\alpha_i} (i = 1, \ldots, n)$
iff $\mathcal{V}(\varphi, I, w, \nu) = \mathcal{V}(\psi, I, w, \nu)$
iff $\mathcal{V}(\varphi = \psi, I, w, \nu) = \top$ [Proposition 3.10].

□

Not surprisingly, the meaning of a rigid term is independent of the world, as shown by the next result.

Proposition 3.12. *Let t be a term, I an interpretation, and ν a variable assignment. If t is rigid, then $\mathcal{V}(t, I, w, \nu) = \mathcal{V}(t, I, w', \nu)$, for all w, w'.*

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

If t is a variable x, then $\mathcal{V}(x, I, w, \nu) = \nu(x) = \mathcal{V}(x, I, w', \nu)$, for all w, w'.

It t is a constant C, then C must be rigid, so that $\mathcal{V}(C, I, w, \nu) = V(C, w) = V(C, w') = \mathcal{V}(C, I, w, \nu)$, for all w, w'.

If t has the form $\lambda x.s$, then, for all w, w',

$\mathcal{V}(\lambda x.s, I, w, \nu)$
$=$ 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 $\nu'(x) = d$
$=$ the function whose value for each $d \in \mathcal{D}_\alpha$ is $\mathcal{V}(s, I, w', \nu')$ [induction hypothesis]
$= \mathcal{V}(\lambda x.s, I, w', \nu)$.

If t has the form $(s\ r)$, then, for all w, w',

$\mathcal{V}((s\ r), I, w, \nu)$
$= \mathcal{V}(s, I, w, \nu)(\mathcal{V}(r, I, w, \nu))$
$= \mathcal{V}(s, I, w', \nu)(\mathcal{V}(r, I, w', \nu))$ [induction hypothesis]
$= \mathcal{V}((s\ r), I, w', \nu)$.

If t has the form $(t_1, \ldots, t_n)$, the proof is similar to preceding case.
If t has the form $\Box_i s$, then, for all w, w',

$$\begin{aligned}
&\mathcal{V}(\Box_i s, I, w, \nu) \\
&= \mathcal{M}(\{\mathcal{V}(s, I, w_1, \nu)\}_{w_1 \in W_1}), \text{ where } W_1 = \{w_1 : w \, R_i \, w_1\} \\
&= \mathcal{M}(\{\mathcal{V}(s, I, w_2, \nu)\}_{w_2 \in W_2}), \text{ where } W_2 = \{w_2 : w' \, R_i \, w_2\} \quad \text{[induction hypothesis]} \\
&= \mathcal{V}(\Box_i s, I, w', \nu).
\end{aligned}$$

□

Proposition 3.13. *Let I be an interpretation, w a world in I, ν a variable assignment, and φ a formula. Then the following hold.*

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

Proof. 1.

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

2.

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

□

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

Proposition 3.14. *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 $\Box_i s$. Then

$$\begin{aligned}
&\mathcal{V}(\Box_i s, I, w, \nu_1)\\
&= \mathcal{M}(\{\mathcal{V}(s, I, w', \nu_1)\}_{w' \in W'})\\
&= \mathcal{M}(\{\mathcal{V}(s, I, w', \nu_2)\}_{w' \in W'}) \quad \text{[induction hypothesis]}\\
&= \mathcal{V}(\Box_i s, I, w, \nu_2).
\end{aligned}$$

□

The next three results in this subsection about modalities will later provide useful assumptions for use in computations. The first concerns the case when the term t in $\Box_i t$ is rigid.

Proposition 3.15. *Let t be a rigid term, I an interpretation, w a world in I, ν a variable assignment, and $i \in \{1, \ldots, m\}$. Then $\mathcal{V}(\Box_i t, I, w, \nu) = \mathcal{V}(t, I, w, \nu)$.*

Proof. Since t is rigid, by Proposition 3.12, all the $\mathcal{V}(t, I, w', \nu)$, for $w' \in W'$, are equal. The result then follows directly from the definition of $\mathcal{V}(\Box_i t, I, w, \nu)$. □

The next proposition gives an important property of terms having type of the form $\alpha \to \beta$ that are applied to a rigid argument.

Proposition 3.16. *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 $w \in W$. Let s be a term having type of the form $\alpha \to \beta$, t a term of type α that is rigid, and $i \in \{1, \ldots, m\}$. Then $\mathcal{V}((\Box_i s\ t), I, w, \nu) = \mathcal{V}(\Box_i(s\ t), I, w, \nu)$.*

Proof. The type of s has rank > 0. Thus

$$\begin{aligned}
&\mathcal{V}((\Box_i s\ t), I, w, \nu)\\
&= \mathcal{V}(\Box_i s, I, w, \nu)(\mathcal{V}(t, I, w, \nu))\\
&= \mathcal{M}(\{\mathcal{V}(s, I, w', \nu)\}_{w'\in W'})(\mathcal{V}(t, I, w, \nu))\\
&= \mathcal{M}(\{\mathcal{V}(s, I, w', \nu)(\mathcal{V}(t, I, w, \nu))\}_{w'\in W'})\\
&= \mathcal{M}(\{\mathcal{V}(s, I, w', \nu)(\mathcal{V}(t, I, w', \nu))\}_{w'\in W'}) \quad [t \text{ is rigid}]\\
&= \mathcal{M}(\{\mathcal{V}((s\ t), I, w', \nu)\}_{w'\in W'})\\
&= \mathcal{V}(\Box_i(s\ t), I, w, \nu).
\end{aligned}$$

□

Next I show that one can switch λ's and modalities.

Proposition 3.17. *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 $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, and $i \in \{1, \ldots, m\}$. Then $\mathcal{V}(\Box_i\lambda x.t, I, w, \nu) = \mathcal{V}(\lambda x.\Box_i t, I, w, \nu)$.*

Proof.

$$\begin{aligned}
&\mathcal{V}(\Box_i\lambda x.t, I, w, \nu)\\
&= \mathcal{M}(\{\mathcal{V}(\lambda x.t, I, w', \nu)\}_{w'\in W'})\\
&= \mathcal{M}(\{\text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(t, I, w', \nu'), \text{ where } x \text{ has type}\\
&\qquad \alpha \text{ and } \nu' \text{ is } \nu \text{ except } \nu'(x) = d\}_{w'\in W'})\\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{M}(\{\mathcal{V}(t, I, w', \nu')\}_{w'\in W'}),\\
&\quad \text{where } x \text{ has type } \alpha \text{ and } \nu' \text{ is } \nu \text{ except } \nu'(x) = d\\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(\Box_i t, I, w, \nu'),\\
&\quad \text{where } x \text{ has type } \alpha \text{ and } \nu' \text{ is } \nu \text{ except } \nu'(x) = d\\
&= \mathcal{V}(\lambda x.\Box_i t, I, w, \nu).
\end{aligned}$$

□

There are analogous results to the preceding three for dual modalities.

Proposition 3.18. *Let φ be a rigid biterm, I an interpretation, w a world in I, ν a variable assignment, and $i \in \{1, \ldots, m\}$. Then $\mathcal{V}(\Diamond_i\varphi, I, w, \nu) = \mathcal{V}(\varphi, I, w, \nu)$.*

Proof.

$$\begin{aligned}
&\mathcal{V}(\Diamond_i\varphi, I, w, \nu)\\
&= \mathcal{V}(\neg\Box_i\neg\varphi, I, w, \nu)\\
&= V(\neg, w)(\mathcal{V}(\Box_i\neg\varphi, I, w, \nu))\\
&= V(\neg, w)(\mathcal{V}(\neg\varphi, I, w, \nu)) \quad [\text{Proposition 3.15}]\\
&= \mathcal{V}(\neg\neg\varphi, I, w, \nu)\\
&= \mathcal{V}(\varphi, I, w, \nu) \quad [\text{Proposition 3.1}].
\end{aligned}$$

□

Proposition 3.19. *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 $w \in W$. Let φ be a biterm having type of the form $\alpha \to \beta$, t a term of type α that is rigid, and $i \in \{1, \ldots, m\}$. Then $\mathcal{V}((\Diamond_i\varphi\ t), I, w, \nu) = \mathcal{V}(\Diamond_i(\varphi\ t), I, w, \nu)$.*

Proof.

$$
\begin{aligned}
&\mathcal{V}((\Diamond_i\varphi\ t), I, w, \nu)\\
&= \mathcal{V}((\neg\Box_i\neg\varphi\ t), I, w, \nu)\\
&= \mathcal{V}(\neg(\Box_i\neg\varphi\ t), I, w, \nu)) \quad \text{[Part 6 of Proposition 3.1]}\\
&= V(\neg, w)(\mathcal{V}((\Box_i\neg\varphi\ t), I, w, \nu))\\
&= V(\neg, w)(\mathcal{V}(\Box_i(\neg\varphi\ t), I, w, \nu)) \quad \text{[Proposition 3.16]}\\
&= V(\neg, w)(\mathcal{M}(\{\mathcal{V}((\neg\varphi\ t), I, w', \nu)\}_{w'\in W'}))\\
&= V(\neg, w)(\mathcal{M}(\{\mathcal{V}(\neg(\varphi\ t), I, w', \nu)\}_{w'\in W'}))\\
&= V(\neg, w)(\mathcal{V}(\Box_i\neg(\varphi\ t), I, w, \nu))\\
&= \mathcal{V}(\neg\Box_i\neg(\varphi\ t), I, w, \nu))\\
&= \mathcal{V}(\Diamond_i(\varphi\ t), I, w, \nu).
\end{aligned}
$$

□

Proposition 3.20. *Let φ be a biterm, $I \equiv \langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}, V\rangle$ an interpretation, ν a variable assignment with respect to $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, and $i \in \{1, \ldots, m\}$. Then $\mathcal{V}(\Diamond_i\lambda x.\varphi, I, w, \nu) = \mathcal{V}(\lambda x.\Diamond_i\varphi, I, w, \nu)$.*

Proof.

$$
\begin{aligned}
&\mathcal{V}(\Diamond_i\lambda x.\varphi, I, w, \nu)\\
&= \mathcal{V}(\neg\Box_i\neg\lambda x.\varphi, I, w, \nu)\\
&= V(\neg, w)(\mathcal{V}(\Box_i\neg\lambda x.\varphi, I, w, \nu))\\
&= V(\neg, w)(\mathcal{M}(\{\mathcal{V}(\neg\lambda x.\varphi, I, w', \nu)\}_{w'\in W'}))\\
&= V(\neg, w)(\mathcal{M}(\{\mathcal{V}(\lambda x.\neg\varphi, I, w', \nu)\}_{w'\in W'})) \quad \text{[Part 9 of Proposition 3.1]}\\
&= V(\neg, w)(\mathcal{V}(\Box_i\lambda x.\neg\varphi, I, w, \nu))\\
&= V(\neg, w)(\mathcal{V}(\lambda x.\Box_i\neg\varphi, I, w, \nu)) \quad \text{[Proposition 3.17]}\\
&= \mathcal{V}(\neg\lambda x.\Box_i\neg\varphi, I, w, \nu)\\
&= \mathcal{V}(\lambda x.\neg\Box_i\neg\varphi, I, w, \nu) \quad \text{[Part 9 of Proposition 3.1]}\\
&= \mathcal{V}(\lambda x.\Diamond_i\varphi, I, w, \nu).
\end{aligned}
$$

□

The next result establishes semantically the Kripke axiom for the case of biterms.

Proposition 3.21. *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 $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, and φ and ψ biterms of the same type. Then $\mathcal{V}(\Box_i(\varphi \longrightarrow \psi) \longrightarrow (\Box_i\varphi \longrightarrow \Box_i\psi), I, w, \nu) = \mathbb{T}$.*

Proof. Suppose that the type of φ and ψ is $\alpha_1 \to \cdots \to \alpha_n \to \Omega$. Let $d_j \in \mathcal{D}_{\alpha_j} (j = 1, \ldots, n)$. Then

$$
\begin{aligned}
&\mathcal{V}(\Box_i(\varphi \longrightarrow \psi) \longrightarrow (\Box_i\varphi \longrightarrow \Box_i\psi), I, w, \nu)\ d_1\ \ldots\ d_n \\
&= (V(\longrightarrow, w)(\mathcal{V}(\Box_i(\varphi \longrightarrow \psi), I, w, \nu)))(\mathcal{V}(\Box_i\varphi \longrightarrow \Box_i\psi, I, w, \nu))\ d_1\ \ldots\ d_n \\
&= \begin{cases} \top & \text{if } \mathcal{V}(\Box_i(\varphi \longrightarrow \psi), I, w, \nu)\ d_1\ \ldots\ d_n = \bot \text{ or} \\ & \quad \mathcal{V}(\Box_i\varphi \longrightarrow \Box_i\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top \\ \bot & \text{otherwise} \end{cases} \\
&= \begin{cases} \top & \text{if } \mathcal{V}(\Box_i(\varphi \longrightarrow \psi), I, w, \nu)\ d_1\ \ldots\ d_n = \top \text{ and } \mathcal{V}(\Box_i\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \top \\ & \quad \text{implies } \mathcal{V}(\Box_i\psi, I, w, \nu)\ d_1\ \ldots\ d_n = \top \\ \bot & \text{otherwise} \end{cases} \\
&= \begin{cases} \top & \text{if } \mathcal{V}(\varphi \longrightarrow \psi, I, w', \nu)\ d_1\ \ldots\ d_n = \top \text{ and } \mathcal{V}(\varphi, I, w', \nu)\ d_1\ \ldots\ d_n = \top, \\ & \quad \text{for each } w' \text{ such that } w\, R_i\, w', \\ & \quad \text{implies } \mathcal{V}(\psi, I, w', \nu)\ d_1\ \ldots\ d_n = \top, \text{ for each } w' \text{ such that } w\, R_i\, w' \\ \bot & \text{otherwise} \end{cases} \\
&= \top.
\end{aligned}
$$

□

Proposition 3.21 justifies the Kripke axiom

$$\Box_i(\boldsymbol{\varphi} \longrightarrow \boldsymbol{\psi}) \longrightarrow (\Box_i\boldsymbol{\varphi} \longrightarrow \Box_i\boldsymbol{\psi}),$$

where $\boldsymbol{\varphi}$ and $\boldsymbol{\psi}$ are syntactical variables ranging over biterms of the same type, being used as a global assumption. (See below for the definition of a global assumption.)

Closely related to Proposition 3.21 is the next result.

Proposition 3.22. *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 $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, and s and t terms of the same type. Then $\mathcal{V}(\Box_i(s = t) \longrightarrow (\Box_i s = \Box_i t), I, w, \nu) = \top$.*

Proof. By Part 3 of Proposition 3.1, it suffices to show that if $\mathcal{V}(\Box_i(s = t), I, w, \nu) = \top$, then $\mathcal{V}(\Box_i s = \Box_i t, I, w, \nu) = \top$. Suppose that $\mathcal{V}(\Box_i(s = t), I, w, \nu) = \top$. Then, by Proposition 3.7, $\mathcal{V}(s = t, I, w', \nu) = \top$, for each w' such that $w'\, R_i\, w$. It follows that $\mathcal{V}(s, I, w', \nu) = \mathcal{V}(t, I, w', \nu)$, for each w' such that $w'\, R_i\, w$. Consequently $\mathcal{V}(\Box_i s, I, w, \nu) = \mathcal{V}(\Box_i t, I, w, \nu)$, and so $\mathcal{V}(\Box_i s = \Box_i t, I, w, \nu) = \top$. □

Proposition 3.22 justifies the global assumption

$$\Box_i(\boldsymbol{s} = \boldsymbol{t}) \longrightarrow (\Box_i\boldsymbol{s} = \Box_i\boldsymbol{t}),$$

where $\boldsymbol{s}$ and $\boldsymbol{t}$ are syntactical variables ranging over terms of the same type.

The next result gives a semantic version of η-reduction.

Proposition 3.23. *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 $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, t a term of type $\sigma \to \tau$, and x a variable of type σ that is not free in t. Then $\mathcal{V}(\lambda x.(t\ x) = t, I, w, \nu) = \top$.*

Proof.

$$
\begin{aligned}
&\mathcal{V}(\lambda x.(t\ x), I, w, \nu)\\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}((t\ x), I, w, \nu'),\\
&\quad \text{where } \nu' \text{ is } \nu \text{ except } \nu'(x) = d\\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(t, I, w, \nu')(\mathcal{V}(x, I, w, \nu'))\\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(t, I, w, \nu')(\nu'(x))\\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(t, I, w, \nu')(d)\\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(t, I, w, \nu)(d)\\
&\quad [\text{Proposition 3.14, since } x \text{ is not free in } t \text{ so that } \nu \text{ and } \nu' \text{ agree on the free variables of } t]\\
&= \mathcal{V}(t, I, w, \nu).
\end{aligned}
$$

Hence $\mathcal{V}(\lambda x.(t\ x) = t, I, w, \nu) = \top$. □

Proposition 3.23 justifies the global assumption

$$\forall f.(\lambda x.(f\ x) = f).$$

This proposition also provides justification for the rule of η-reduction.

Definition 3.12. The rule of *η-reduction* is as follows: $\lambda x.(t\ x) \succ_\eta t$, if x is not free in t.

Function definitions that are used by agents will be introduced below. These definitions will have the form either $\Box_{j_1} \cdots \Box_{j_r} \forall x.((f\ x) = t)$ or $\Box_{j_1} \cdots \Box_{j_r} (f = \lambda x.t)$. The next result shows that semantically these two definitions are the same and hence can be used interchangeably.

Proposition 3.24. *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 $\{\mathcal{D}_\alpha\}_{\alpha \in \mathfrak{S}}$, $w \in W$, f a function, and t a term. Then $\mathcal{V}((f = \lambda x.t) = \forall x.((f\ x) = t), I, w, \nu) = \top$.*

Proof.

$$
\begin{aligned}
&\mathcal{V}(\forall x.((f\ x) = t), I, w, \nu) = \top\\
&\text{iff } \mathcal{V}((f\ x) = t, I, w, \nu') = \top, \text{for every } x\text{-variant } \nu' \text{ of } \nu \quad [\text{Part 3 of Proposition 3.5}]\\
&\text{iff } \mathcal{V}((f\ x), I, w, \nu') = \mathcal{V}(t, I, w, \nu'), \text{for every } x\text{-variant } \nu' \text{ of } \nu \quad [\text{Proposition 3.10}]\\
&\text{iff } \mathcal{V}(\lambda x.(f\ x), I, w, \nu) = \mathcal{V}(\lambda x.t, I, w, \nu)\\
&\text{iff } \mathcal{V}(f, I, w, \nu) = \mathcal{V}(\lambda x.t, I, w, \nu) \quad [\text{Proposition 3.23}]\\
&\text{iff } \mathcal{V}(f = \lambda x.t, I, w, \nu) = \top \quad [\text{Proposition 3.10}].
\end{aligned}
$$

Hence $\mathcal{V}((f = \lambda x.t) = \forall x.((f\ x) = t), I, w, \nu) = \top$. □

A result closely related to Proposition 3.24 follows.

Proposition 3.25. *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 $\{\mathcal{D}_\alpha\}_{\alpha \in \mathfrak{S}}$, $w \in W$, s and t terms having type of the form $\sigma \to \tau$, and x a variable of type σ not occurring freely in s or t. Then $\mathcal{V}((s = t) = \forall x.((s\ x) = (t\ x)), I, w, \nu) = \top$.*

Proof.

$$
\begin{aligned}
&\mathcal{V}(\forall x.((s\ x) = (t\ x)), I, w, \nu) = \top \\
&\text{iff } \mathcal{V}((s\ x) = (t\ x), I, w, \nu') = \top, \text{ for every } x\text{-variant } \nu' \text{ of } \nu \\
&\text{iff } \mathcal{V}((s\ x), I, w, \nu') = \mathcal{V}((t\ x), I, w, \nu'), \text{ for every } x\text{-variant } \nu' \text{ of } \nu \\
&\text{iff } \mathcal{V}(s, I, w, \nu)(\nu'(x)) = \mathcal{V}(t, I, w, \nu)(\nu'(x)), \text{ for every } x\text{-variant } \nu' \text{ of } \nu \\
&\text{iff } \mathcal{V}(s, I, w, \nu) = \mathcal{V}(t, I, w, \nu) \\
&\text{iff } \mathcal{V}(s = t, I, w, \nu) = \top.
\end{aligned}
$$

Hence $\mathcal{V}((s = t) = \forall x.((s\ x) = (t\ x)), I, w, \nu) = \top$. □

Proposition 3.25 justifies the axiom of extensionality

$$\forall f.\forall g.((f = g) = \forall x.((f\ x) = (g\ x)))$$

being used as a global assumption.

The next result justifies a global assumption concerning tuples.

Proposition 3.26. *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 $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, and $(s_1, \ldots, s_n)$ and $(t_1, \ldots, t_n)$ terms of the same type. Then $\mathcal{V}(((s_1, \ldots, s_n) = (t_1, \ldots, t_n)) = (s_1 = t_1) \wedge \cdots \wedge (s_n = t_n), I, w, \nu) = \top$.*

Proof.

$$
\begin{aligned}
&\mathcal{V}((s_1, \ldots, s_n) = (t_1, \ldots, t_n), I, w, \nu) = \top \\
&\text{iff } \mathcal{V}((s_1, \ldots, s_n), I, w, \nu) = \mathcal{V}((t_1, \ldots, t_n), I, w, \nu) \\
&\text{iff } (\mathcal{V}(s_1, I, w, \nu), \ldots, \mathcal{V}(s_n, I, w, \nu)) = (\mathcal{V}(t_1, I, w, \nu), \ldots, \mathcal{V}(t_n, I, w, \nu)) \\
&\text{iff } \mathcal{V}(s_i, I, w, \nu) = \mathcal{V}(t_i, I, w, \nu), \text{ for } i = 1, \ldots, n \\
&\text{iff } \mathcal{V}(s_i = t_i, I, w, \nu) = \top, \text{ for } i = 1, \ldots, n \\
&\text{iff } \mathcal{V}((s_1 = t_1) \wedge \cdots \wedge (s_n = t_n), I, w, \nu) = \top.
\end{aligned}
$$

Hence $\mathcal{V}(((s_1, \ldots, s_n) = (t_1, \ldots, t_n)) = (s_1 = t_1) \wedge \cdots \wedge (s_n = t_n), I, w, \nu) = \top$. □

Proposition 3.26 justifies

$$\forall x_1 \ldots \forall x_n.\forall y_1 \ldots \forall y_n.(((x_1, \ldots, x_n) = (y_1, \ldots, y_n)) = (x_1 = y_1) \wedge \cdots \wedge (x_n = y_n))$$

being used as a global assumption.

Definition 3.5 gives the free interpretation to data constructors. From this interpretation, two results and their corresponding global assumption follow.

Proposition 3.27. *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 $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, $w \in W$, and C a data constructor such that $C\ s_1 \ldots s_n$ and $C\ t_1 \ldots t_n$ are terms. Then $\mathcal{V}((C\ s_1 \ldots s_n = C\ t_1 \ldots t_n) = (s_1 = t_1) \wedge \cdots \wedge (s_n = t_n), I, w, \nu) = \top$.*

Proof.

$$\begin{aligned}
&\mathcal{V}(C\ s_1 \ldots s_n = C\ t_1 \ldots t_n, I, w, \nu) = \top \\
\text{iff } &\mathcal{V}(C\ s_1 \ldots s_n, I, w, \nu) = \mathcal{V}(C\ t_1 \ldots t_n, I, w, \nu) \\
\text{iff } &V(C, w)\ \mathcal{V}(s_1, I, w, \nu) \ldots \mathcal{V}(s_n, I, w, \nu) = V(C, w)\ \mathcal{V}(t_1, I, w, \nu) \ldots \mathcal{V}(t_n, I, w, \nu) \\
\text{iff } &C\ \mathcal{V}(s_1, I, w, \nu) \ldots \mathcal{V}(s_n, I, w, \nu) = C\ \mathcal{V}(t_1, I, w, \nu) \ldots \mathcal{V}(t_n, I, w, \nu) \\
\text{iff } &\mathcal{V}(s_i, I, w, \nu) = \mathcal{V}(t_i, I, w, \nu), \text{ for } i = 1, \ldots, n \\
\text{iff } &\mathcal{V}(s_i = t_i, I, w, \nu) = \top, \text{ for } i = 1, \ldots, n \\
\text{iff } &\mathcal{V}((s_1 = t_1) \wedge \cdots \wedge (s_n = t_n), I, w, \nu) = \top.
\end{aligned}$$

Hence $\mathcal{V}((C\ s_1 \ldots s_n = C\ t_1 \ldots t_n) = (s_1 = t_1) \wedge \cdots \wedge (s_n = t_n), I, w, \nu) = \top$. □

Proposition 3.27 justifies

$$\forall x_1 \ldots \forall x_n . \forall y_1 \ldots \forall y_n . ((\boldsymbol{C}\ x_1 \ldots x_n = \boldsymbol{C}\ y_1 \ldots y_n) = (x_1 = y_1) \wedge \cdots \wedge (x_n = y_n)),$$

where $\boldsymbol{C}$ is a syntactical variable ranging over data constructors, being used as a global assumption.

Proposition 3.28. *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 $\{\mathcal{D}_\alpha\}_{\alpha \in \mathfrak{S}}$, $w \in W$, and C and D data constructors such that $C \neq D$ and $C\ s_1 \ldots s_n = D\ t_1 \ldots t_m$ is a term. Then $\mathcal{V}(C\ s_1 \ldots s_n = D\ t_1 \ldots t_m, I, w, \nu) = \bot$.*

Proof.

$$\begin{aligned}
&\mathcal{V}(C\ s_1 \ldots s_n, I, w, \nu) \\
&= V(C, w)\ \mathcal{V}(s_1, I, w, \nu) \ldots \mathcal{V}(s_n, I, w, \nu) \\
&= C\ \mathcal{V}(s_1, I, w, \nu) \ldots \mathcal{V}(s_n, I, w, \nu) \\
&\neq D\ \mathcal{V}(t_1, I, w, \nu) \ldots \mathcal{V}(t_m, I, w, \nu) \\
&= V(D, w)\ \mathcal{V}(t_1, I, w, \nu) \ldots \mathcal{V}(t_m, I, w, \nu) \\
&= \mathcal{V}(D\ t_1 \ldots t_m, I, w, \nu).
\end{aligned}$$

Hence $\mathcal{V}(C\ s_1 \ldots s_n = D\ t_1 \ldots t_m, I, w, \nu) = \bot$. □

Proposition 3.28 justifies

$$\forall x_1 \ldots \forall x_n . \forall y_1 \ldots \forall y_m . ((\boldsymbol{C}\ x_1 \ldots x_n = \boldsymbol{D}\ y_1 \ldots y_m) = \bot,$$

where $\boldsymbol{C}$ and $\boldsymbol{D}$ are syntactical variables ranging over data constructors such that $\boldsymbol{C} \neq \boldsymbol{D}$, being used as a global assumption.

3.3 Admissible Substitutions

The next few results are concerned with knowing the semantics of a term after a substitution is applied or a subterm replaced. For these, a restriction on the application of substitutions is needed.

Definition 3.13. Let t be a term and $\theta \equiv \{x_1/t_1, \ldots, x_n/t_n\}$ a substitution. Then θ is *admissible* with respect to t if, for $i = 1, \ldots, n$, whenever x_i has a free occurrence in t that is a modal occurrence, t_i is rigid.

Proposition 3.29. *Let t be a 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. Suppose that θ is admissible with respect to t. 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

$$
\begin{aligned}
& \mathcal{V}(t\theta, I, w, \nu) \\
&= \mathcal{V}(t\theta|_t, I, w, \nu) \quad \text{[Proposition 2.6]} \\
&= \mathcal{V}(t, I, w, \nu^*), \text{ where } \nu^*(x_{i_j}) = \mathcal{V}(t_{i_j}, I, w, \nu), \text{ for } j = 1, \ldots, k, \\
&\quad \text{and } \nu^*(y) = \nu(y), \text{ for } y \notin \{x_{i_1}, \ldots, x_{i_k}\} \quad \text{[assumption]} \\
&= \mathcal{V}(t, I, w, \nu'), \text{ where } \nu'(x_i) = \mathcal{V}(t_i, I, w, \nu), \text{ for } i = 1, \ldots, n, \\
&\quad \text{and } \nu'(y) = \nu(y), \text{ for } y \notin \{x_1, \ldots, x_n\} \\
&\quad \text{[Proposition 3.14, since } \nu^* \text{ and } \nu' \text{ agree on the free variables in } t\text{]}.
\end{aligned}
$$

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

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

(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

$$\begin{aligned}
&\mathcal{V}((\lambda x.s)\theta, I, w, \nu)\\
&= \mathcal{V}(\lambda y.(s(\{x/y\} \cup \theta)), I, w, \nu) \quad \text{[Proposition 2.8]}\\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(s(\{x/y\} \cup \theta), I, w, \overline{\nu}),\\
&\quad \text{where } \overline{\nu} \text{ is } \nu \text{ except } \overline{\nu}(y) = d\\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(s, I, w, \overline{\nu}'), \text{ where } \overline{\nu}'(x) = \mathcal{V}(y, I, w, \overline{\nu}),\\
&\quad \overline{\nu}'(x_i) = \mathcal{V}(t_i, I, w, \overline{\nu}), \text{ for } i = 1, \ldots, n, \text{ and } \overline{\nu}'(z) = \overline{\nu}(z), \text{ for } z \notin (\{x_1, \ldots, x_n\} \cup \{x\})\\
&\quad \text{[induction hypothesis]}\\
&= \text{the function whose value for each } d \in \mathcal{D}_\alpha \text{ is } \mathcal{V}(s, I, w, \overline{\nu'}),\\
&\quad \text{where } \overline{\nu'} \text{ is } \nu' \text{ except } \overline{\nu'}(x) = d, \text{ and } \nu'(x_i) = \mathcal{V}(t_i, I, w, \nu), \text{ for } i = 1, \ldots, n,\\
&\quad \text{and } \nu'(z) = \nu(z), \text{ for } z \notin \{x_1, \ldots, x_n\}\\
&\quad [\overline{\nu}' \text{ and } \overline{\nu'} \text{ agree, except possibly on } y]\\
&= \mathcal{V}(\lambda x.s, I, w, \nu').
\end{aligned}$$

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

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

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

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

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 $\Box_i s$. Then

$$\begin{aligned}
&\mathcal{V}((\Box_i s)\theta, I, w, \nu)\\
&= \mathcal{V}(\Box_i(s\theta), I, w, \nu)\\
&= \mathcal{M}(\{\mathcal{V}(s\theta, I, w', \nu)\}_{w' \in W'})\\
&= \mathcal{M}(\{\mathcal{V}(s, I, w', \nu') : \nu'(x_i) = \mathcal{V}(t_i, I, w', \nu), \text{ for } i = 1, \ldots, n,\\
&\qquad\qquad \text{and } \nu'(y) = \nu(y), \text{ for } y \notin \{x_1, \ldots, x_n\}\}_{w' \in W'}) \qquad \text{[induction hypothesis]}\\
&= \mathcal{M}(\{\mathcal{V}(s, I, w', \nu')\}_{w' \in W'}),\\
&\quad \text{where } \nu'(x_i) = \mathcal{V}(t_i, I, w, \nu), \text{ for } i = 1, \ldots, n, \text{ and } \nu'(y) = \nu(y), \text{ for } y \notin \{x_1, \ldots, x_n\}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [\theta \text{ is admissible w.r.t. } t]
\end{aligned}$$

$= \mathcal{V}(\Box_i s, I, w, \nu')$,
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 θ be admissible with respect to t in Proposition 3.29 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. Since C is not rigid, θ is not admissible with respect to t.

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 $y \neq x$.

Since variables are rigid, the next result is a special case of Proposition 3.29.

Proposition 3.30. *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 $z \neq x$.*

Proposition 3.31. *Let t be a term, s and r terms of the same type, I an interpretation, w a world in I, and ν a variable assignment. Suppose that $\{x/s\}$ and $\{x/r\}$ are admissible with respect to t. If $\mathcal{V}(s = r, I, w, \nu) = \top$, then $\mathcal{V}(t\{x/s\}, I, w, \nu) = \mathcal{V}(t\{x/r\}, I, w, \nu)$.*

Proof. Proposition 3.10 shows that $\mathcal{V}(s, I, w, \nu) = \mathcal{V}(r, I, w, \nu)$. Then

$\mathcal{V}(t\{x/s\}, I, w, \nu)$
$= \mathcal{V}(t, I, w, \nu')$, where ν' is ν except that $\nu'(x) = \mathcal{V}(s, I, w, \nu)$ [Proposition 3.29]
$= \mathcal{V}(t, I, w, \nu')$, where ν' is ν except that $\nu'(x) = \mathcal{V}(r, I, w, \nu)$
$= \mathcal{V}(t\{x/r\}, I, w, \nu)$ [Proposition 3.29].

□

Proposition 3.32. *Let φ be a biterm, I an interpretation, w a world in I, and θ a substitution. Suppose that θ is admissible with respect to φ and that $\mathcal{V}(\Box_{k_1} \ldots \Box_{k_m} \varphi, I, w, \nu) = \mathbb{T}$, for each variable assignment ν. Then $\mathcal{V}(\Box_{k_1} \ldots \Box_{k_m} \varphi\theta, I, w, \nu) = \mathbb{T}$, for each variable assignment ν.*

Proof. The proof is by induction on the length m of $\Box_{k_1} \ldots \Box_{k_m}$.

Suppose first that $m = 0$. Assume that $\mathcal{V}(\varphi, I, w, \nu) = \mathbb{T}$, for each variable assignment ν. Let θ be $\{x_1/t_1, \ldots, x_n/t_n\}$ and ν a variable assignment. By Proposition 3.29, $\mathcal{V}(\varphi\theta, I, w, \nu) = \mathcal{V}(\varphi, I, w, \nu')$, 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 $\mathcal{V}(\varphi\theta, I, w, \nu) = \mathbb{T}$.

Suppose next that the result holds for length m. Assume that $\mathcal{V}(\Box_{k_1} \ldots \Box_{k_{m+1}} \varphi, I, w, \nu) = \mathbb{T}$, for each variable assignment ν. Then it follows that, for each variable assignment ν, $\mathcal{V}(\Box_{k_2} \ldots \Box_{k_{m+1}} \varphi, I, w', \nu) = \mathbb{T}$, for each w' such that $w\ R_{k_1}\ w'$. By the induction hypothesis, for each variable assignment ν, $\mathcal{V}(\Box_{k_2} \ldots \Box_{k_{m+1}} \varphi\theta, I, w', \nu) = \mathbb{T}$, for each w' such that $w\ R_{k_1}\ w'$. Thus, for each variable assignment ν, $\mathcal{V}(\Box_{k_1} \ldots \Box_{k_{m+1}} \varphi\theta, I, w, \nu) = \mathbb{T}$. □

3.4 Term Replacement and Denotations

The next result will be important for the computational mechanism introduced in Section 6.

Proposition 3.33. *Let t be a term, s a subterm of t at occurrence o, r a term having the same type as s, I an interpretation, and w a world in I. Suppose that $k_1 \dots k_m$ is the modal path to o in t and $\mathcal{V}(\Box_{k_1} \cdots \Box_{k_m}(s = r), I, w, \nu) = \top$, for each variable assignment ν. Then $\mathcal{V}(t = t[s/r]_o, I, w, \nu) = \top$, 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, $t[s/r]_o$ is r, and the modal path to o is empty. Suppose that ν is a variable assignment. Then $\mathcal{V}(t = t[s/r]_o, I, w, \nu) = \mathcal{V}(s = r, I, w, \nu) = \top$.

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

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

$$
\begin{aligned}
&\mathcal{V}(\lambda x.q, I, w, \nu) \\
&= \text{the function whose value for each } d \in \mathcal{D}_\gamma \text{ is } \mathcal{V}(q, I, w, \nu'), \\
&\quad \text{where the type of } t \text{ is } \gamma \to \delta \text{ and } \nu' \text{ is } \nu \text{ except } \nu'(x) = d \\
&= \text{the function whose value for each } d \in \mathcal{D}_\gamma \text{ is } \mathcal{V}(q[s/r]_{o'}, I, w, \nu') \\
&\quad \text{[induction hypothesis (since modal path to } o' \text{ is same as to } o) \text{ and Proposition 3.10]} \\
&= \mathcal{V}(\lambda x.(q[s/r]_{o'}), I, w, \nu) \\
&= \mathcal{V}((\lambda x.q)[s/r]_o, I, w, \nu).
\end{aligned}
$$

By Proposition 3.10, $\mathcal{V}(\lambda x.q = (\lambda x.q)[s/r]_o, I, w, \nu) = \top$.

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

$$
\begin{aligned}
&\mathcal{V}((u\ v), I, w, \nu) \\
&= \mathcal{V}(u, I, w, \nu)(\mathcal{V}(v, I, w, \nu)) \\
&= \mathcal{V}(u[s/r]_{o'}, I, w, \nu)(\mathcal{V}(v, I, w, \nu)) \quad \text{[induction hypothesis and Proposition 3.10]} \\
&= \mathcal{V}((u[s/r]_{o'}\ v), I, w, \nu) \\
&= \mathcal{V}((u\ v)[s/r]_o, I, w, \nu).
\end{aligned}
$$

By Proposition 3.10, $\mathcal{V}((u\ v) = (u\ v)[s/r]_o, I, w, \nu) = \top$. The case when $o = 2o'$ is similar.

The case when t has the form $(t_1, \dots, t_n)$ is similar to the preceding one.

Consider finally the case when t has the form $\Box_i q$. Thus $i = k_1$, $o = 1o'$, for some o', and the modal path to o' in q is $k_2 \dots k_m$. Furthermore, for each variable assignment ν and each w' such that $w\, R_i\, w'$, $\mathcal{V}(\Box_{k_2} \cdots \Box_{k_m}(s = r), I, w', \nu) = \top$. Suppose that ν is a variable assignment. Then

$$
\begin{aligned}
&\mathcal{V}(\Box_i q, I, w, \nu) \\
&= \mathcal{M}(\{\mathcal{V}(q, I, w', \nu)\}_{w' \in W'}) \\
&= \mathcal{M}(\{\mathcal{V}(q[s/r]_{o'}, I, w', \nu)\}_{w' \in W'}) \quad \text{[Induction hypothesis and Proposition 3.10]} \\
&= \mathcal{V}(\Box_i(q[s/r]_{o'}, I, w, \nu) \\
&= \mathcal{V}((\Box_i q)[s/r]_o, I, w, \nu).
\end{aligned}
$$

By Proposition 3.10, $\mathcal{V}(\Box_i q = (\Box_i q)[s/r]_o, I, w, \nu) = \top$. □

3.5 Denotations of α-equivalent Terms

The next result establishes that α-equivalent terms have the same meaning.

Proposition 3.34. *Let s and t be terms, $I \equiv \langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}, V\rangle$ an interpretation, ν a variable assignment with respect to $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, and $w \in W$. Then the following hold.*

1. *If y is not free in t, then $\mathcal{V}(\lambda x.t, I, w, \nu) = \mathcal{V}(\lambda y.(t\{x/y\}), I, w, \nu)$.*
2. *If $s \longrightarrow_\alpha t$, then $\mathcal{V}(s, I, w, \nu) = \mathcal{V}(t, I, w, \nu)$.*
3. *If $s \overset{*}{\longleftrightarrow}_\alpha t$, then $\mathcal{V}(s, I, w, \nu) = \mathcal{V}(t, I, w, \nu)$.*

Proof. 1. Suppose that x and y are variables of type α. Then

$\mathcal{V}(\lambda y.(t\{x/y\}), I, w, \nu)$
$=$ the function whose value for each $d \in \mathcal{D}_\alpha$ is $\mathcal{V}(t\{x/y\}, I, w, \nu')$,
where ν' is ν except $\nu'(y) = d$
$=$ the function whose value for each $d \in \mathcal{D}_\alpha$ is $\mathcal{V}(t, I, w, \nu'')$,
where $\nu''(x) = \nu'(y)$, $\nu''(z) = \nu'(z)$, for $z \neq x$, and ν' is ν except $\nu'(y) = d$
[Proposition 3.30]
$=$ the function whose value for each $d \in \mathcal{D}_\alpha$ is $\mathcal{V}(t, I, w, \nu''')$,
where ν''' is ν except $\nu'''(x) = d$
[Proposition 3.14, since y is not free in t so ν'' and ν''' agree on the free variables of t]
$= \mathcal{V}(\lambda x.t, I, w, \nu)$.

2. Suppose that t is $s[u/v]_o$, where $u \succ_\alpha v$. From Part 1, $\mathcal{V}(u, I, w, \nu) = \mathcal{V}(v, I, w, \nu)$, for each world w and variable assignment ν. By Proposition 3.10, $\mathcal{V}(u = v, I, w, \nu) = \top$, for each world w and variable assignment ν. Now let $k_1 \dots k_m$ be the modal path to o in s. Thus $\mathcal{V}(\Box_{k_1} \cdots \Box_{k_m}(u = v), I, w, \nu) = \top$, for each world w and variable assignment ν. By Proposition 3.33, $\mathcal{V}(s = s[u/v]_o, I, w, \nu) = \top$, for each world w and variable assignment ν. Hence, by Proposition 3.10 again, $\mathcal{V}(s, I, w, \nu) = \mathcal{V}(s[u/v]_o, I, w, \nu)$, for each world w and variable assignment ν.

3. This is an easy induction on the length of the path of α-conversions from s to t, using Part 2. □

As shown by the next example, the condition that y not be free in t in Part 1 of Proposition 3.34 cannot be dropped.

Example 3.2. Let x and y be variables of type *Nat*, I an interpretation consisting of a single world w whose domain for the type *Nat* is the set of natural numbers $\mathbb{N}$, and ν a variable assignment that maps y to 0. Then

$\mathcal{V}(\lambda x.y, I, w, \nu)$
$=$ the function from $\mathbb{N}$ to $\mathbb{N}$ that maps each natural number to 0.

But

$\mathcal{V}(\lambda y.(y\{x/y\}), I, w, \nu)$
$= \mathcal{V}(\lambda y.y, I, w, \nu)$
$=$ the identity function on $\mathbb{N}$.

3.6 β-Reduction

I now define the relation $\succ_\beta$ corresponding to β-reduction.

Definition 3.14. The rule of *β-reduction* is as follows: $(\lambda x.s\ t) \ \succ_\beta \ s\{x/t\}$, if $\{x/t\}$ is admissible with respect to s.

Definition 3.15. The relation $\longrightarrow_\beta$ is defined by $u \longrightarrow_\beta u[s/t]_o$ if s is a subterm of u at occurrence o and $s \ \succ_\beta \ t$.

Let $\xrightarrow{*}_\beta$ be the reflexive, transitive closure of $\longrightarrow_\beta$.

Let $\xleftrightarrow{*}_\beta$ be the reflexive, symmetric, and transitive closure of $\longrightarrow_\beta$.

Definition 3.16. If $s \xleftrightarrow{*}_\beta t$, then s and t are said to be *β-equivalent*.

The next result establishes that β-equivalent terms have the same meaning.

Proposition 3.35. *Let s and t be terms, $I \equiv \langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}, V\rangle$ an interpretation, ν a variable assignment with respect to $\{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}$, and $w \in W$. Then the following hold.*

1. *If $\{x/t\}$ is admissible with respect to s, then $\mathcal{V}((\lambda x.s\ t), I, w, \nu) = \mathcal{V}(s\{x/t\}, I, w, \nu)$.*
2. *If $s \longrightarrow_\beta t$, then $\mathcal{V}(s, I, w, \nu) = \mathcal{V}(t, I, w, \nu)$.*
3. *If $s \xleftrightarrow{*}_\beta t$, then $\mathcal{V}(s, I, w, \nu) = \mathcal{V}(t, I, w, \nu)$.*

Proof. 1.

$$
\begin{aligned}
&\mathcal{V}((\lambda x.s\ t), I, w, \nu)\\
&= \mathcal{V}(\lambda x.s, I, w, \nu)\ (\mathcal{V}(t, I, w, \nu))\\
&= \mathcal{V}(s, I, w, \nu'), \text{where } \nu' \text{ is } \nu \text{ except that } \nu'(x) = \mathcal{V}(t, I, w, \nu)\\
&= \mathcal{V}(s\{x/t\}, I, w, \nu) \quad \text{[Proposition 3.29]}.
\end{aligned}
$$

2. Suppose that t is $s[u/v]_o$, where $u \succ_\beta v$. From Part 1, $\mathcal{V}(u, I, w, \nu) = \mathcal{V}(v, I, w, \nu)$, for each world w and variable assignment ν. By Proposition 3.10, $\mathcal{V}(u = v, I, w, \nu) = \top$, for each world w and variable assignment ν. Now let $k_1 \ldots k_m$ be the modal path to o in s. Thus $\mathcal{V}(\Box_{k_1} \cdots \Box_{k_m}(u = v), I, w, \nu) = \top$, for each world w and variable assignment ν. By Proposition 3.33, $\mathcal{V}(s = s[u/v]_o, I, w, \nu) = \top$, for each world w and variable assignment ν. Hence, by Proposition 3.10 again, $\mathcal{V}(s, I, w, \nu) = \mathcal{V}(s[u/v]_o, I, w, \nu)$, for each world w and variable assignment ν.

3. This is an easy induction on the length of the path of β-reductions from s to t, using Part 2. □

The next example shows that the condition that $\{x/t\}$ be admissible with respect to s in Part 1 of Proposition 3.35 cannot be dropped.

Example 3.3. Let α be a type, $p : \alpha \to \Omega$, $C : \alpha$, $s \equiv \Box_i(p\ x)$, and $t \equiv C$. 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. Since C is not rigid, $\{x/C\}$ is not admissible with respect to $\Box_i(p\ x)$. Then

$$\begin{aligned}
&\mathcal{V}((\lambda x.\Box_i(p\ x)\ C), I, w, \nu)\\
&= \mathcal{V}(\lambda x.\Box_i(p\ x), I, w, \nu)(\mathcal{V}(C, I, w, \nu))\\
&= \mathcal{V}(\Box_i(p\ x), I, w, \nu'), \text{where } \nu' \text{ is } \nu \text{ except that } \nu'(x) = \mathcal{V}(C, I, w, \nu)\\
&= \mathcal{V}((p\ x), I, w', \nu'), \text{where } \nu' \text{ is } \nu \text{ except that } \nu'(x) = \mathcal{V}(C, I, w, \nu)\\
&= \top.
\end{aligned}$$

But

$$\begin{aligned}
&\mathcal{V}(\Box_i(p\ x)\{x/C\}, I, w, \nu)\\
&= \mathcal{V}(\Box_i(p\ C), I, w, \nu)\\
&= \mathcal{V}((p\ C), I, w', \nu)\\
&= \bot.
\end{aligned}$$

3.7 Validity and Consequence

Now come the important concepts of validity and consequence.

Definition 3.17. A biterm φ is *valid at a world w in an interpretation I* if $\mathcal{V}(\varphi, I, w, \nu) = \mathbb{T}$, for every variable assignment ν.

Definition 3.18. A biterm φ is *satisfiable at a world w in an interpretation I* if $\mathcal{V}(\varphi, I, w, \nu) \neq \mathbb{\perp}$, for some variable assignment ν.

The concepts of validity and satisfiability are dual to one another, as the next result shows.

Proposition 3.36. *Let I be an interpretation and φ a biterm. Then the following hold.*

1. *φ is valid at w in I iff $\neg\varphi$ is not satisfiable at w in I.*
2. *φ is satisfiable at w in I iff $\neg\varphi$ is not valid at w in I.*

Proof. 1.

$$\begin{aligned}
&\varphi \text{ is valid at } w \text{ in } I\\
\text{iff } &\mathcal{V}(\varphi, I, w, \nu) = \mathbb{T}, \text{ for every variable assignment } \nu\\
\text{iff } &\mathcal{V}(\neg\varphi, I, w, \nu) = \mathbb{\perp}, \text{ for every variable assignment } \nu\\
\text{iff } &\neg\varphi \text{ is not satisfiable at } w \text{ in } I.
\end{aligned}$$

2.

$$\begin{aligned}
&\varphi \text{ is satisfiable at } w \text{ in } I\\
\text{iff } &\mathcal{V}(\varphi, I, w, \nu) \neq \mathbb{\perp}, \text{ for some variable assignment } \nu\\
\text{iff } &\mathcal{V}(\neg\varphi, I, w, \nu) \neq \mathbb{T}, \text{ for some variable assignment } \nu\\
\text{iff } &\neg\varphi \text{ is not valid at } w \text{ in } I.
\end{aligned}$$

□

Proposition 3.37. *Let $I \equiv \langle W, \{R_i\}_{i=1}^m, \{\mathcal{D}_\alpha\}_{\alpha\in\mathfrak{S}}, V\rangle$ be an interpretation, $w \in W$, φ a biterm, and x a variable that is not free in φ.*

1. *φ is valid at w in I iff $(\varphi\, x)$ is valid at w in I.*
2. *φ is valid in I iff $(\varphi\, x)$ is valid in I.*

Proof. 1. This follows from Proposition 3.4.
2. This follows from Part 1 and the definition of validity in an interpretation. □

Definition 3.19. A biterm φ is *valid in an interpretation* if it is valid at each world in the interpretation.

Proposition 3.38. *Let φ be a biterm, I an interpretation, and w a world in I. Then the following hold.*

1. *φ is valid at w in I iff $\lambda x.\varphi$ is valid at w in I.*
2. *φ is valid in I iff $\lambda x.\varphi$ is valid in I.*
3. *φ is valid at w in I iff $(\Pi\, \varphi)$ is valid at w in I.*
4. *φ is valid in I iff $(\Pi\, \varphi)$ is valid in I.*
5. *φ is valid at w in I iff $\forall(\varphi)$ is valid at w in I.*
6. *φ is valid in I iff $\forall(\varphi)$ is valid in I.*

Proof. 1. This follows from Part 1 of Proposition 3.5.
2. This follows from Part 1 and the definition of validity in an interpretation.
3. This follows from Part 2 of Proposition 3.2.
4. This follows from Part 1 and the definition of validity in an interpretation.
5. This follows from Part 3 of Proposition 3.5.
6. This follows from Part 3 and the definition of validity in an interpretation. □

Definition 3.20. 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 3.21. A biterm is *valid in a frame* if it is valid in every interpretation based on the frame.

Definition 3.22. If $\mathtt{L}$ is a class of frames, then a biterm is $\mathtt{L}$-*valid* if it is valid in each frame in $\mathtt{L}$.

A theory is just a set of biterms, but it will be useful to distinguish two different kinds of biterms in the theory. This leads to the next definition.

Definition 3.23. A *theory* is a pair $(\mathcal{G}, \mathcal{L})$, where $\mathcal{G}$ is a set of biterms, called *global assumptions*, and $\mathcal{L}$ is a set of biterms, called *local assumptions*.

Next comes the fundamental concept of a biterm being a consequence of a theory.

Definition 3.24. Let L be a class of frames, $\mathcal{T} \equiv (\mathcal{G}, \mathcal{L})$ a theory, and φ a biterm. Then φ is an L-*consequence* of the theory $\mathcal{T}$ if the following holds: for each interpretation I based on a frame in 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 K denote the class of all frames. In the following, I abbreviate 'K-valid' to 'valid' and 'K-consequence' to 'consequence'.

For any class L of frames, $\mathtt{L} \subseteq \mathtt{K}$. Thus φ is a consequence of $\mathcal{T}$ implies that φ is an L-consequence of $\mathcal{T}$. Similarly, φ is valid implies that φ is L-valid.

Clearly each global assumption and each local assumption is a consequence of the theory. Hence, for any class L of frames, each global assumption and each local assumption is an L-consequence of the theory.

In applications, the concept of L-consequence is used in the following way. Each application has a distinguished pointed interpretation (I, w) known as the *intended pointed interpretation*, where I is based on a frame in some class L of frames. This means that, in the application, w is the actual world and I provides the worlds accessible to w by the various accessibility relations. Typically, in applications, given some term t, one needs to know the meaning of t in the intended pointed interpretation. If a formal definition of the intended pointed interpretation is available (as there is in the model checking setting), then this problem can be solved (under some finiteness assumptions). However, it is much more usual not to have a formal definition of the intended pointed interpretation, but instead to have a theory $\mathcal{T} \equiv (\mathcal{G}, \mathcal{L})$ for the application. The assumption is that (I, w) and $\mathcal{T}$ are related in the following way:

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

Suppose now that a biterm φ is shown to be an L-consequence of $\mathcal{T}$. Then φ is valid at w in I. Intuitively, φ is 'true in the intended pointed interpretation'.

To emphasise, the key concept is that of meaning in the intended pointed interpretation, since it describes the actual situation the application has to deal with. But usually it is not possible for an application to have much detail about the intended pointed interpretation. Instead, it is usually much more feasible to be able to maintain a theory for the application, which serves as a *surrogate* for the intended pointed interpretation. The theory can then be used in many cases to answer the original question about the meaning of a term in the intended pointed interpretation. But note that this approach does not always work. For example, a biterm being true in the intended pointed interpretation is not the same as the biterm being a consequence of the theory: the latter also implies truth in any pointed interpretation satisfying the conditions in the preceding paragraph, not just the intended one, and usually there are many of these. Notwithstanding this difficulty, the convenience of working with the theory makes it worthwhile. Besides, in practice, there is usually little choice in the matter!

Sometimes it is convenient to have available the concept of a pointed model of a theory.

Definition 3.25. Let $\mathcal{T} \equiv (\mathcal{G}, \mathcal{L})$ be a theory. A *pointed model* for $\mathcal{T}$ is a pair (I, w) consisting of an interpretation I and a world w in I such that each $\psi \in \mathcal{G}$ is valid in I and each $\psi \in \mathcal{L}$ is valid at w in I.

In this terminology, a biterm φ is a consequence of $\mathcal{T}$ if, for each pointed model (I, w) of $\mathcal{T}$, φ is valid at w in I. Also, of course, an intended pointed interpretation should be a pointed model of the relevant theory.

The next result shows that it makes no difference semantically whether one works with the (possibly open) assumptions of a theory or the universal closures of all the assumptions (or, for that matter, anything in between).

Proposition 3.39. *Let $\mathcal{T}$ be a theory and $\overline{\mathcal{T}}$ the theory obtained from $\mathcal{T}$ by taking as its global assumptions the universal closures of the global assumptions in $\mathcal{T}$ and its local assumptions the universal closures of the local assumptions in $\mathcal{T}$. Let* L *be a class of frames and φ a biterm. Then φ is an* L*-consequence of $\mathcal{T}$ iff φ is an* L*-consequence of $\overline{\mathcal{T}}$.*

Proof. The result follows immediately from Proposition 3.38 and the definition of L-consequence. □

Note. I will often make use of Proposition 3.39 in the following by leaving off outer universal quantifiers in assumptions, thus avoiding syntactic clutter. In particular, this will usually be done with assumptions, such as those in the standard equality theory of Section 6.1 and the definitions of list-processing functions, that are used in computations.

The next four results will be needed to show the correctness of the computational mechanism introduced in Section 6.

Proposition 3.40. *Let $\mathcal{T}$ be a theory and φ a global assumption of $\mathcal{T}$. Then, for each class* L *of frames and for each sequence $l_1 \dots l_p$ of indices, $\Box_{l_1} \cdots \Box_{l_p} \varphi$ in an* L*-consequence of $\mathcal{T}$.*

Proof. Suppose that $\mathcal{T}$ is $(\mathcal{G}, \mathcal{L})$. Let I be an interpretation based on a frame in L for which all members of $\mathcal{G}$ are valid in I and consider a world w in I such that each member of $\mathcal{L}$ is valid at w in I. Since φ is a global assumption, it is valid in I and therefore valid at each world v in I. Thus, whatever the sequence $l_1 \dots l_p$ of indices, $\Box_{l_1} \cdots \Box_{l_p} \varphi$ is valid at w in I. It follows that $\Box_{l_1} \cdots \Box_{l_p} \varphi$ in an L-consequence of $\mathcal{T}$. □

Proposition 3.41. *Let $\mathcal{T}$ be a theory,* L *a class of frames, t a biterm, and $i \in \{1, \dots, m\}$. Then $\forall x.\Box_i t$ is an* L*-consequence of $\mathcal{T}$ iff $\Box_i \forall x.t$ is an* L*-consequence of $\mathcal{T}$.*

Proof. Let $\mathcal{T}$ be $(\mathcal{G}, \mathcal{L})$, I an interpretation based on a frame in L, and w a world in I. Suppose that each member of $\mathcal{G}$ is valid in I and that each member of $\mathcal{L}$ is valid at w in I. Then, by Part 1 of Proposition 3.9, $\mathcal{V}(\forall x.\Box_i t, I, w, \nu) = \mathcal{V}(\Box_i \forall x.t, I, w, \nu)$, for each variable assignment ν. Hence $\forall x.\Box_i t$ is an L-consequence of $\mathcal{T}$ iff $\Box_i \forall x.t$ is an L-consequence of $\mathcal{T}$. □

Proposition 3.42. *Let $\mathcal{T}$ be a theory,* L *a class of frames, and φ a biterm. Then φ is an* L*-consequence of $\mathcal{T}$ iff $\forall(\varphi)$ is an* L*-consequence of $\mathcal{T}$.*

Proof. Let $\mathcal{T}$ be $(\mathcal{G}, \mathcal{L})$, I an interpretation based on a frame in L, and w a world in I. Suppose that each member of $\mathcal{G}$ is valid in I and that each member of $\mathcal{L}$ is valid at w in I. Then, by Part 5 of Proposition 3.38, φ is valid at w in I iff $\forall(\varphi)$ is valid at w in I. Hence φ is an L-consequence of $\mathcal{T}$ iff $\forall(\varphi)$ is an L-consequence of $\mathcal{T}$. □

Proposition 3.43. *Let $\mathcal{T}$ be a theory,* L *a class of frames, $\Box_{l_1} \cdots \Box_{l_p} \varphi$ a biterm that is an* L*-consequence of $\mathcal{T}$, and θ a substitution that is admissible with respect to φ. Then $\Box_{l_1} \cdots \Box_{l_p} \varphi\theta$ is an* L*-consequence of $\mathcal{T}$.*

Proof. Let $\mathcal{T}$ be $(\mathcal{G}, \mathcal{L})$, I an interpretation based on a frame in L, and w a world in I. Suppose that each member of $\mathcal{G}$ is valid in I and that each member of $\mathcal{L}$ is valid at w in I. Since $\Box_{l_1} \cdots \Box_{l_p} \varphi$ is an L-consequence of $\mathcal{T}$, it follows that $\mathcal{V}(\Box_{l_1} \cdots \Box_{l_p} \varphi, I, w, \nu) = \mathbb{T}$, for each variable assignment ν. By Proposition 3.32, $\mathcal{V}(\Box_{l_1} \cdots \Box_{l_p} \varphi\theta, I, w, \nu) = \mathbb{T}$, for each variable assignment ν. Thus $\Box_{l_1} \cdots \Box_{l_p} \varphi\theta$ is an L-consequence of $\mathcal{T}$. □

3.8 Modalities for Applications

As is well known, modalities can have a variety of meanings, depending on the application. In this subsection, some of these are indicated. The setting here is more general than usual in that it allows modalities to be applied to terms and dual modalities to be applied to biterms Much more detail for the usual setting of formulas can be found in [FHMV95] and [GKWZ03], for example.

In multi-agent applications, one meaning for $\Box_i t$ is that 'agent i knows t', where t is a term. In this case, the modality $\Box_i$ is written as $\boldsymbol{K}_i$. The logic $\mathbf{S5}_m$ is commonly used to capture the intended meaning of knowledge. The axioms for this logic are as follows (where $i = 1, \ldots, m$).

$$\begin{aligned}
&\boldsymbol{K}_i\boldsymbol{\varphi} \longrightarrow \boldsymbol{\varphi} \quad (T_i)\\
&\boldsymbol{\varphi} \longrightarrow \boldsymbol{K}_i\neg\boldsymbol{K}_i\neg\boldsymbol{\varphi} \quad (B_i)\\
&\boldsymbol{K}_i\boldsymbol{\varphi} \longrightarrow \boldsymbol{K}_i\boldsymbol{K}_i\boldsymbol{\varphi} \quad (4_i)
\end{aligned}$$

where $\boldsymbol{\varphi}$ is a syntactical variable ranging over biterms. Axiom T_i is reflexivity, B_i is symmetry, and 4_i is transitivity (of the corresponding accessibility relation). Note that while a term such as $\boldsymbol{K}_i 42$ is admitted, it is not subject to the preceding axioms that only apply to biterms.

A weaker notion is that of belief. In this case, $\Box_i t$ means that 'agent i believes t' and the modality $\Box_i$ is written as $\boldsymbol{B}_i$. The logic $\mathbf{KD45}_m$ is commonly used to capture the intended meaning of belief.

A term of the form $\boldsymbol{B}_i t$, where t is not a formula, while perhaps unfamiliar, is intuitive and useful. Intuitively, $\boldsymbol{B}_i t$ means 'agent i believes t'; formally, its semantics is given by the $\mathcal{M}$ function in Definition 3.10. A typical example of this, other than when t is a formula, is when t is a function symbol, say, f. Then $\boldsymbol{B}_i f$ means 'agent i believes function f'. This is normally captured by a function definition for f that is the one believed by agent i (in contrast to other definitions for f that may be believed by other agents). Plenty of examples like this are given in Section 7.

There is considerable attention in the literature on logics such as $\mathbf{S5}_m$, $\mathbf{KD45}_m$, and related ones concerning the subtle differences in meaning that the various axioms might bestow on modalities for knowledge and belief. When trying to build agent applications these subtleties are usually completely overshadowed by the issues of dealing with the various function definitions for different agents at different times and the choice of interaction axioms that relate the various modalities. In any case, the consequences that might follow from using the axioms T_i, B_i, 4_i, and so on, are not explored in this paper.

The modalities also have a variety of temporal readings. Here I remark on one version of the case of discrete time. For this case, the intended interpretation includes the natural numbers $\mathbb{N}$ (or, perhaps, the integers $\mathbb{Z}$). Each $n \in \mathbb{N}$ is interpreted as a time point with $n+1$ the next time point after n.

I adopt the usual modalities $\bigcirc$ ('next'), $\Box$ ('always in the future'), $\Diamond$ ('sometime in the future'), and $\boldsymbol{U}$ ('until'). The modalities $\bigcirc$ and $\Box$ are assumed to be among the $\Box_1, \ldots, \Box_m$.

In particular, $\bigcirc$ is interpreted by the relation $\{(n, n+1) \mid n \in \mathbb{N}\}$. The modalities $\bigcirc$ and $\Box$ can be applied to arbitrary terms. The modality $\Diamond$ is dual to $\Box$ and thus can only be applied to biterms.

The modality $\boldsymbol{U}$ is a binary modality and thus, strictly speaking, cannot even be written down in the logic since it only admits unary modalities. However, as I show now, one can introduce $\boldsymbol{U}$ as an abbreviation for a higher-order term involving $\bigcirc$ that can be applied to biterms. By this means, the higher-order nature of the logic be used to effectively admit a binary modality into the logic. For this, $\bigcirc^n$ and $\bigcirc^{\leq n}$, where $n \in \mathbb{N}$, need to be defined. Let φ be a biterm. Then $\bigcirc^0\varphi = \bigcirc^{\leq 0}\varphi = \varphi$ and, for $n \geq 0$,

$$\bigcirc^{n+1}\varphi = \bigcirc\bigcirc^n\varphi$$
$$\bigcirc^{\leq n+1}\varphi = \bigwedge_{i=0}^{n+1} \bigcirc^i\varphi.$$

These definitions can be expressed in the logic by the following axiom schemas.

$$\mathit{sometime} : \alpha \times \mathit{Nat} \to \alpha$$
$$(\mathit{sometime}\ (\boldsymbol{\varphi}, n)) = \mathit{if}\ n = 0\ \mathit{then}\ \boldsymbol{\varphi}\ \mathit{else}\ \bigcirc(\mathit{sometime}\ (\boldsymbol{\varphi}, n-1))$$

$$\mathit{upto\_sometime} : \alpha \times \mathit{Nat} \to \alpha$$
$$(\mathit{upto\_sometime}\ (\boldsymbol{\varphi}, n)) =$$
$$\mathit{if}\ n = 0\ \mathit{then}\ \boldsymbol{\varphi}\ \mathit{else}\ ((\mathit{sometime}\ (\boldsymbol{\varphi}, n)) \wedge (\mathit{upto\_sometime}\ (\boldsymbol{\varphi}, n-1))),$$

where α is a biterm type and $\boldsymbol{\varphi}$ is a syntactical variable standing for an arbitrary biterm of type α. (Syntactical variables are needed (at least in the definition of *sometime*) because the modality restricts too much the substitutions that can be made in the first argument of *sometime*.) Then, for $n \in \mathbb{N}$ and φ a biterm, the procedure of Section 6.1 can be used to compute $(\mathit{sometime}\ (\varphi, n))$ and $(\mathit{upto\_sometime}\ (\varphi, n))$. The answers returned are

$$(\mathit{sometime}\ (\varphi, n)) = \bigcirc^n\varphi$$

and

$$(\mathit{upto\_sometime}\ (\varphi, n)) = \bigcirc^{\leq n}\varphi.$$

Now $\varphi\,\mathbf{U}\,\psi$ can be defined with the meaning 'φ holds until ψ holds'. Define $\varphi\,\boldsymbol{U}\,\psi$ to be an abbreviation for

$$\psi \vee \exists n.((\mathit{upto\_sometime}\ (\varphi, n)) \wedge (\mathit{sometime}\ (\psi, n+1))).$$

This clearly captures the intended meaning of $\boldsymbol{U}$.

Next I give a useful collection of equations about the modalities $\bigcirc$, $\Box$, $\Diamond$, and $\boldsymbol{U}$ that are valid at every world in every interpretation and, therefore, can be used as global assumptions in any theory.

$$\boldsymbol{\varphi}\,\boldsymbol{U}\,\boldsymbol{\psi} = \boldsymbol{\psi} \vee (\boldsymbol{\varphi} \wedge \bigcirc(\boldsymbol{\varphi}\,\boldsymbol{U}\,\boldsymbol{\psi}))$$
$$\Diamond\boldsymbol{\varphi} = \boldsymbol{\varphi} \vee \bigcirc\Diamond\boldsymbol{\varphi}$$
$$\Box\boldsymbol{\varphi} = \boldsymbol{\varphi} \wedge \bigcirc\Box\boldsymbol{\varphi}$$
$$\bigcirc(\boldsymbol{\varphi} \wedge \boldsymbol{\psi}) = \bigcirc\boldsymbol{\varphi} \wedge \bigcirc\boldsymbol{\psi}$$
$$\bigcirc(\boldsymbol{\varphi} \vee \boldsymbol{\psi}) = \bigcirc\boldsymbol{\varphi} \vee \bigcirc\boldsymbol{\psi}$$
$$\bigcirc(\boldsymbol{\varphi} \longrightarrow \boldsymbol{\psi}) = \bigcirc\boldsymbol{\varphi} \longrightarrow \bigcirc\boldsymbol{\psi}$$
$$\bigcirc\neg\boldsymbol{\varphi} = \neg\bigcirc\boldsymbol{\varphi},$$

where $\boldsymbol{\varphi}$ and $\boldsymbol{\psi}$ are syntactical variables ranging over biterms of the same type.

Companions to the above (future) temporal modalities are the past temporal modalities $\bullet$ ('last'), $\blacksquare$ ('always in the past'), $\blacklozenge$ ('sometime in the past'), and $\boldsymbol{S}$ ('since'). The modality $\bullet$ is interpreted by the relation $\{(n+1, n) \mid n \in \mathbb{N}\} \cup \{(0,0)\}$. Also $\boldsymbol{S}$ can be defined in terms of $\bullet$ in an analogous way as for the corresponding definitions for the future temporal modalities. There is also a set of equations for the past temporal modalities that are useful for reasoning.

$$\boldsymbol{\varphi}\,\boldsymbol{S}\,\boldsymbol{\psi} = \boldsymbol{\psi} \vee (\boldsymbol{\varphi} \wedge \bullet(\boldsymbol{\varphi}\,\boldsymbol{S}\,\boldsymbol{\psi}))$$
$$\blacklozenge\boldsymbol{\varphi} = \boldsymbol{\varphi} \vee \bullet\blacklozenge\boldsymbol{\varphi}$$
$$\blacksquare\boldsymbol{\varphi} = \boldsymbol{\varphi} \wedge \bullet\blacksquare\boldsymbol{\varphi}$$
$$\bullet(\boldsymbol{\varphi} \wedge \boldsymbol{\psi}) = \bullet\boldsymbol{\varphi} \wedge \bullet\boldsymbol{\psi}$$
$$\bullet(\boldsymbol{\varphi} \vee \boldsymbol{\psi}) = \bullet\boldsymbol{\varphi} \vee \bullet\boldsymbol{\psi}$$
$$\bullet(\boldsymbol{\varphi} \longrightarrow \boldsymbol{\psi}) = \bullet\boldsymbol{\varphi} \longrightarrow \bullet\boldsymbol{\psi}$$
$$\bullet\neg\boldsymbol{\varphi} = \neg\bullet\boldsymbol{\varphi},$$

where $\boldsymbol{\varphi}$ and $\boldsymbol{\psi}$ are syntactical variables ranging over biterms of the same type.

The next global assumption is given by Proposition 3.15 which shows that the following schema can be used as a global assumption.

$$\Box_i\boldsymbol{t} = \boldsymbol{t}, \tag{3.1}$$

where $\boldsymbol{t}$ is a syntactical variable ranging over *rigid* terms and $i \in \{1, \ldots, m\}$. Instances of this schema that could be used as global assumptions include the following.

$$\boldsymbol{B}_i 42 = 42$$
$$\bullet 42 = 42$$
$$\boldsymbol{B}_i [] = []$$
$$\boldsymbol{B}_i \top = \top$$
$$\boldsymbol{B}_i \bot = \bot$$
$$\bullet\top = \top$$
$$\bullet\bot = \bot.$$

Proposition 3.18 shows that the following schema dual to (3.1) can be used as a global assumption.

$$\Diamond_i\boldsymbol{\varphi} = \boldsymbol{\varphi}, \tag{3.2}$$

where $\boldsymbol{\varphi}$ is a syntactical variable ranging over *rigid* biterms and $i \in \{1, \ldots, m\}$.

Proposition 3.16 shows that the following schema can be used as a global assumption.

$$(\Box_i \mathbf{s}\ \mathbf{t}) = \Box_i(\mathbf{s}\ \mathbf{t}), \tag{3.3}$$

where $\mathbf{s}$ is a syntactical variable ranging over terms having type of the form $\alpha \to \beta$, $\mathbf{t}$ is a syntactical variable ranging over *rigid* terms of type α, and $i \in \{1, \ldots, m\}$.

Specialised to some of the epistemic and temporal modalities discussed so far, this means that

$$(\boldsymbol{B}_i\mathbf{s}\ \mathbf{t}) = \boldsymbol{B}_i(\mathbf{s}\ \mathbf{t})$$
$$(\bigcirc\mathbf{s}\ \mathbf{t}) = \bigcirc(\mathbf{s}\ \mathbf{t})$$
$$(\Box\mathbf{s}\ \mathbf{t}) = \Box(\mathbf{s}\ \mathbf{t})$$
$$(\bullet\mathbf{s}\ \mathbf{t}) = \bullet(\mathbf{s}\ \mathbf{t})$$
$$(\blacksquare\mathbf{s}\ \mathbf{t}) = \blacksquare(\mathbf{s}\ \mathbf{t})$$

are global assumptions (under the rigidity assumption on $\mathbf{t}$).

Global assumption (3.3) is also useful when the rank of the type of s is > 1. Suppose that s is a term whose type has rank 2 and t_1 and t_2 are rigid terms such that $((s\ t_1)\ t_2)$ is a term. Then $((\Box_i s\ t_1)\ t_2)$ can be rewritten first to $(\Box_i(s\ t_1)\ t_2)$ and then to $\Box_i((s\ t_1)\ t_2)$.

Proposition 3.19 shows that the following schema dual to (3.3) can be used as a global assumption.

$$(\Diamond_i\boldsymbol{\varphi}\ \mathbf{t}) = \Diamond_i(\boldsymbol{\varphi}\ \mathbf{t}), \tag{3.4}$$

where $\boldsymbol{\varphi}$ is a syntactical variable ranging over biterms having type of the form $\alpha \to \beta$, $\mathbf{t}$ is a syntactical variable ranging over *rigid* terms of type α, and $i \in \{1, \ldots, m\}$.

Proposition 3.17 shows that the following schema can be used as a global assumption.

$$\Box_i\lambda x.\mathbf{t} = \lambda x.\Box_i\mathbf{t}, \tag{3.5}$$

where $\mathbf{t}$ is a syntactical variable ranging over terms and $i \in \{1, \ldots, m\}$.

Proposition 3.20 shows that the following schema dual to (3.5) can be used as a global assumption.

$$\Diamond_i\lambda x.\boldsymbol{\varphi} = \lambda x.\Diamond_i\boldsymbol{\varphi}, \tag{3.6}$$

where $\boldsymbol{\varphi}$ is a syntactical variable ranging over biterms and $i \in \{1, \ldots, m\}$.

In addition, there are interactions between the temporal and epistemic modalities for which implicational schemas such as the following may be appropriate (depending on the application). These model agents with perfect recall.

$$\boldsymbol{B}_i\bigcirc\boldsymbol{\varphi} \longrightarrow \bigcirc\boldsymbol{B}_i\boldsymbol{\varphi}$$
$$\boldsymbol{K}_i\bigcirc\boldsymbol{\varphi} \longrightarrow \bigcirc\boldsymbol{K}_i\boldsymbol{\varphi},$$

where $\boldsymbol{\varphi}$ is a syntactical variable ranging over biterms.

Similarly, there are implicational schemas available for use that connect the past temporal and epistemic modalities. These model agents with another form of perfect recall.

$$\bullet\boldsymbol{B}_i\boldsymbol{\varphi} \longrightarrow \boldsymbol{B}_i\bullet\boldsymbol{\varphi}$$
$$\bullet\boldsymbol{K}_i\boldsymbol{\varphi} \longrightarrow \boldsymbol{K}_i\bullet\boldsymbol{\varphi}$$

where $\boldsymbol{\varphi}$ is a syntactical variable ranging over biterms. In Section 7.3, an illustration of a computation using the past temporal modalities will be given. Generally, the past temporal modalities are useful in applications for which an agent uses the past (and present) to select a suitable action.

4 Polymorphism

In most applications, (parametric) polymorphism arises naturally. This section shows how polymorphism can be introduced into the logic. Considerably more detail about polymorphism (in the non-modal case) is given in [Llo03].

4.1 Types

To begin with, the earlier concept of a type has to be generalised. For this purpose, parameters, which are type variables, are introduced.

Definition 4.1. An *alphabet* consists of four sets:

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

Another difference in the above definition of an alphabet compared to the earlier one is that the variables in $\mathfrak{V}$ do not have fixed types; instead each occurrence of a variable in a term is given a type according to its position in the term.

Definition 4.2. A *type* is defined inductively as follows.

1. Each parameter in $\mathfrak{P}$ is a type.
2. If T is a type constructor in $\mathfrak{T}$ 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.)
3. If α and β are types, then $\alpha \to \beta$ is a type.
4. 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.)

Example 4.1. Typical types are $\mathit{Int} \times (\mathit{List}\ a)$ and $\mathit{List}\ (\mathit{List}\ a)$.

Definition 4.3. A type is *closed* if it contains no parameters.

A closed type in the sense of Definition 4.3 is the same as a type in the sense of Definition 2.2.

Since types may now contain parameters, the concept of a type substitution is introduced whose purpose is to instantiate parameters that appear in types.

Definition 4.4. A *type substitution* is a finite set of the form $\{a_1/\alpha_1, \ldots, a_n/\alpha_n\}$, where each a_i is a parameter, each α_i is a type distinct from a_i, and $a_1, \ldots, a_n$ are distinct. Each element a_i/α_i is called a *binding*.

In particular, $\{\}$ is a type substitution called the *identity substitution*.

Notation. If $\mu = \{a_1/\alpha_1, \ldots, a_n/\alpha_n\}$, then $\mathit{domain}(\mu) = \{a_1, \ldots, a_n\}$ and $\mathit{range}(\mu)$ is the set of parameters appearing in $\{\alpha_1, \ldots, \alpha_n\}$.

Definition 4.5. A type substitution $\{a_1/\alpha_1, \ldots, a_n/\alpha_n\}$ is *closed* if α_i is closed, for $i = 1, \ldots, n$.

Definition 4.6. Let $\mu = \{a_1/\alpha_1, \ldots, a_n/\alpha_n\}$ be a type substitution and α a type. Then $\alpha\mu$, the *instance* of α by μ, is the type obtained from α by simultaneously replacing each occurrence of the parameter a_i in α by the type α_i $(i = 1, \ldots, n)$.

Definition 4.7. Let $\mu = \{a_1/\alpha_1, \ldots, a_m/\alpha_m\}$ and $\nu = \{b_1/\beta_1, \ldots, b_n/\beta_n\}$ be type substitutions. Then the *composition* $\mu\nu$ of μ and ν is the type substitution obtained from the set

$$\{a_1/\alpha_1\nu, \ldots, a_m/\alpha_m\nu, b_1/\beta_1, \ldots, b_n/\beta_n\}$$

by deleting any binding $a_i/\alpha_i\nu$ for which $a_i = \alpha_i\nu$ and deleting any binding b_j/β_j for which $b_j \in \{a_1, \ldots, a_m\}$.

Composition is defined so that $\alpha(\mu\nu) = (\alpha\mu)\nu$, for any type α and type substitutions μ and ν. Also $\mu\{\} = \{\}\mu = \mu$, for any μ, so $\{\}$ really is an identity. Composition of type substitutions is associative.

One type can be more general than another.

Definition 4.8. Let α and β be types. Then α is *more general than* β if there exists a type substitution ξ such that $\beta = \alpha\xi$.

Note that 'more general than' includes 'equal to', since ξ can be the identity substitution.

Example 4.2. Let $\alpha = (List\ a) \times \Omega$ and $\beta = (List\ Int) \times \Omega$. Then α is more general than β, since $\beta = \alpha\xi$, where $\xi = \{a/Int\}$.

Definition 4.9. Let $E = \{\alpha_1 = \beta_1, \ldots, \alpha_n = \beta_n\}$ be a set of equations about types. Then a type substitution μ is a *unifier* for E if $\alpha_i\mu$ is identical to $\beta_i\mu$, for $i = 1, \ldots, n$.

Example 4.3. Let $E = \{a \to List\ b = c \to List\ M, a \times b = a \times M\}$, where a, b and c are parameters and M is a unary type constructor. Then the type substitution $\{a/M, b/M, c/M\}$ is a unifier for E.

One type substitution can be more general than another.

Definition 4.10. Let μ and ν be type substitutions. Then μ is *more general* than ν if there exists a type substitution γ such that $\mu\gamma = \nu$.

Definition 4.11. Let E be a set of equations about types and μ be a unifier for E. Then μ is a *most general unifier* for E if, for every unifier ν for E, μ is more general than ν.

Notation. The phrase 'most general unifier' is often abbreviated to 'mgu'.

Example 4.4. Let $E = \{a \to List\ b = c \to List\ M, a \times b = a \times M\}$ be a set of equations about types. Then the type substitution $\{a/c, b/M\}$ is an mgu for E. Note that, for the unifier $\{a/M, b/M, c/M\}$ of Example 4.3, $\{a/M, b/M, c/M\} = \{a/c, b/M\}\{c/M\}$. Thus $\{a/c, b/M\}$ is indeed more general than this unifier.

4.2 Terms

Next the concept of a term from Section 2 is generalised to the polymorphic case. For this, I first turn to a discussion of the constants in $\mathfrak{C}$.

In the polymorphic setting, constants can have signatures that are not closed, as is illustrated by the next example.

Example 4.5. The data constructors for constructing lists are [] having signature *List* a and $\sharp$ having signature $a \to \mathit{List}\ a \to \mathit{List}\ a$, where $\sharp$ is usually written infix. [] represents the empty list and the term $s \sharp t$ represents the list with head s and tail t. Because [] and $\sharp$ are polymorphic, [] and $s \sharp t$ can be used to represent lists of any type. Thus $4 \sharp 5 \sharp 6 \sharp []$ represents the list $[4, 5, 6]$, and $A \sharp B \sharp C \sharp []$ represents the list $[A, B, C]$, where A, B, and C are constants of some type T, say.

The set $\mathfrak{C}$ is similar to before, except that $=_\alpha$, $\neg_\alpha$, $\wedge_\alpha$, $\vee_\alpha$, $\longrightarrow_\alpha$, $\longleftarrow_\alpha$, Σ_α and Π_α are replaced by the following polymorphic versions (where a and the a_i are parameters):

1. $=$, having signature $a \to a \to \Omega$.
2. $\neg$, having signature $(a_1 \to \cdots \to a_n \to \Omega) \to (a_1 \to \cdots \to a_n \to \Omega)$.
3. $\wedge$, $\vee$, $\longrightarrow$, and $\longleftarrow$, having signature $(a_1 \to \cdots \to a_n \to \Omega) \to (a_1 \to \cdots \to a_n \to \Omega) \to (a_1 \to \cdots \to a_n \to \Omega)$.
4. Σ and Π, having signature $(a_1 \to \cdots \to a_n \to \Omega) \to \Omega$.

Data constructors always have a signature of the form $\sigma_1 \to \cdots \to \sigma_n \to (T\ a_1 \ldots a_k)$, where T is a type constructor of arity k, $a_1, \ldots, a_k$ are distinct parameters, and all the parameters appearing in $\sigma_1, \ldots, \sigma_n$ occur among $a_1, \ldots, a_k$ ($n \geq 0,\ k \geq 0$). The *arity* of the data constructor is n. A *nullary* data constructor is a data constructor of arity 0.

Now the concept of a (polymorphic) term, which generalises that of a (monomorphic) term in Definition 2.7, can be defined. However, the polymorphic case is rather more complicated than the monomorphic case since, when putting terms together to make larger terms, it is generally necessary to solve a system of equations and these equations depend upon the relative types of free variables in the component terms. The effect of this is that to define a term one has to define simultaneously its type, and its set of free variables and their relative types.

Definition 4.12. A *term*, together with its type, and its set of free variables and their relative types, is defined inductively as follows.

1. Each variable x in $\mathfrak{V}$ is a term of type a, where a is a parameter.

   The variable x is free with relative type a in x.

2. Each constant C in $\mathfrak{C}$, where C has signature α, is a term of type α.

3. If t is a term of type β and x a variable in $\mathfrak{V}$, then $\lambda x.t$ is a term of type $\alpha \to \beta$, if x is free with relative type α in t, or type $a \to \beta$, where a is a new parameter, otherwise.

   A variable other than x is free with relative type σ in $\lambda x.t$ if the variable is free with relative type σ in t.

4. If s is a term of type α and t a term of type β such that the equation

   $$\alpha = \beta \to b,$$

   where b is a new parameter, augmented with equations of the form

   $$\varrho = \delta,$$

   for each variable that is free with relative type ϱ in s and is also free with relative type δ in t, have a most general unifier ξ, then $(s\ t)$ is a term of type $b\xi$.

   A variable is free with relative type $\sigma\xi$ in $(s\ t)$ if the variable is free with relative type σ in s or t.

5. If $t_1, \ldots, t_n$ are terms of type $\alpha_1, \ldots, \alpha_n$, respectively, such that the set of equations of the form

   $$\varrho_{i_1} = \cdots = \varrho_{i_k},$$

   for each variable that is free with relative type ϱ_{i_j} in the term t_{i_j} ($j = 1, \ldots, k$ and $k > 1$), have a most general unifier ξ, then $(t_1, \ldots, t_n)$ is a term of type $\alpha_1\xi \times \cdots \times \alpha_n\xi$.

   A variable is free with relative type $\sigma\xi$ in $(t_1, \ldots, t_n)$ if the variable is free with relative type σ in t_j, for some $j \in \{1, \ldots, n\}$.

6. If t is a term of type α and $i \in \{1, \ldots, m\}$, then $\Box_i t$ is a term of type α.

The type substitution ξ in Parts 4 and 5 of the definition is called the *associated* mgu.

Notation. Terms of the form $(\Sigma\ \lambda x.t)$ are written as $\exists x.t$ and terms of the form $(\Pi\ \lambda x.t)$ are written as $\forall x.t$.

Here are two examples to illustrate Definition 4.12.

Example 4.6. Let M be a nullary type, and $A : M$ and $concat : List\ a \times List\ a \to List\ a$ be constants. Recall that $[] : List\ a$ and $\sharp : a \to List\ a \to List\ a$ are the data constructors for lists. I will show that $(concat\ ([], [A]))$ is a term. For this, $([], [A])$ must be shown to be a term, which leads to the consideration of $[]$ and $[A]$. Now $[]$ is a term of type $List\ a$, by Part 2 of the definition of a term. By Parts 2 and 4, $(\sharp\ A)$ is a term of type $List\ M \to List\ M$, where along the way the equation $a = M$ is solved with the mgu $\{a/M\}$. Then $((\sharp\ A)\ [])$, which is the list $[A]$, is a term of type $List\ M$ by Part 4, where the equation $List\ M = List\ a$ is solved. By Part 5, it follows that $([], [A])$ is a term of type $List\ a \times List\ M$. Finally, by Part 4 again, $(concat\ ([], [A]))$ is a term of type $List\ M$, where the equation to be solved is $List\ a \times List\ a = List\ a \times List\ M$ whose mgu is $\{a/M\}$.

Example 4.7. Consider the constants $append : List\ a \to List\ a \to List\ a \to \Omega$ and $process : List\ a \to List\ a$. I will show that $\Box_i(((append\ x)\ [])\ (process\ x))$ is a term. First, the variable x is a term of type b, where the parameter is chosen to avoid a clash in the next step. Then $(append\ x)$ is a term of type $List\ a \to List\ a \to \Omega$, for which the equation solved is $List\ a = b$. Next $((append\ x)\ [])$ is a term of type $List\ a \to \Omega$ and x has relative type $List\ a$ in $((append\ x)\ [])$. Now consider $(process\ x)$, for which the constituent parts are $process$ of type $List\ c \to List\ c$ and the variable x of type d. Thus $(process\ x)$ is a term of type $List\ c$

and x has relative type *List* c in (*process* x). Finally, one has to apply ((*append* x) []) to the term (*process* x). For this, by Part 4, there are two equations. These are *List* a = *List* c, coming from the top-level types, and *List* a = *List* c, coming from the free variable x in each of the components. These equations have the mgu $\{c/a\}$. Thus (((*append* x) []) (*process* x)) is a term of type Ω and hence, by Part 6, $\Box_i$(((*append* x) []) (*process* x)) is also a term of type Ω.

The concepts of an occurrence set in Definition 2.17 and a subterm in Definition 2.18 apply equally well to (polymorphic) terms given by Definition 4.12. The concept of the relative type of a subterm in a (polymorphic) term will be useful.

Definition 4.13. The *relative type* of the subterm $t|_o$ of the term t at $o \in \mathcal{O}(t)$ is defined by induction on the length of o as follows.

If the length of o is 0, then the relative type of $t|_o$ is the same as the type of t.

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 $\lambda x.s$, then the relative type of $t|_o$ is the same as the relative type of $s|_{o'}$ in s.

If $o = 1o'$, for some o', and t has the form $(u\ v)$, then the relative type of $t|_o$ is $\sigma\xi$, where σ is the relative type of $u|_{o'}$ in u and ξ is the associated mgu for $(u\ v)$.

If $o = 2o'$, for some o', and t has the form $(u\ v)$, then the relative type of $t|_o$ is $\sigma\xi$, where σ is the relative type of $v|_{o'}$ in v and ξ is the associated mgu for $(u\ v)$.

If $o = io'$, for some $i \in \{1, \ldots, n\}$ and o', and t has the form $(t_1, \ldots, t_n)$, then the relative type of $t|_o$ is $\sigma\xi$, where σ is the relative type of $t_i|_{o'}$ in t_i and ξ is the associated mgu for $(t_1, \ldots, t_n)$.

If $o = 1o'$, for some o', and t has the form $\Box_i s$, then the relative type of $t|_o$ is the same as the relative type of $s|_{o'}$ in s.

Example 4.8. Let M be a nullary type, and *append* : *List* $a \to$ *List* $a \to$ *List* $a \to \Omega$, *process* : *List* $a \to$ *List* a and $A, B, C : M$ be constants. Consider the first occurrence of x in the term $\Box_i$(((*append* x) []) (*process* x)). As a term in its own right, x has type a, for some parameter a. As a subterm of $\Box_i$(((*append* x) []) (*process* x)), x has relative type *List* a.

The occurrence of x in the term $\Box_i$(((*append* x) []) (*process* $[A, B, C]$)) has relative type *List* M. Also the subterm *process* of this term has relative type *List* $M \to$ *List* M.

Next the exact connection between the two definitions of the concept of a term is established. Informally, a polymorphic term can be regarded as standing for a collection of monomorphic terms. I now give a precise meaning to this last statement.

For this purpose, attention must be paid to the alphabets that each definition of a term uses. In particular, given the alphabet that is used in Definition 4.12, a suitable alphabet must be defined for use in Definition 2.7. In both alphabets, the same set of type constructors is used, but the other components are different. Note that, for some fixed set of type constructors, the closed types in the sense of Definition 4.2 are exactly the types in the sense of Definition 2.2.

Definition 4.14. Let A be an alphabet in the sense of Definition 4.1 consisting of the set $\mathfrak{T}$ of type constructors, the set $\mathfrak{P}$ of parameters, the set $\mathfrak{C}$ of constants, and the set $\mathfrak{V}$ of variables. Then $\overline{A}$, the *underlying monomorphic alphabet* for A, is an alphabet in the sense

of Definition 2.1 given by the set $\mathfrak{T}$ of type constructors, the set $\overline{\mathfrak{C}}$ of constants, and the set $\overline{\mathfrak{V}}$ of variables, where

$$\overline{\mathfrak{C}} \equiv \{C_\beta \mid C \in \mathfrak{C}, C \text{ has signature } \alpha, \text{ and } \beta \text{ is a closed instance of } \alpha\},$$

and

$$\overline{\mathfrak{V}} \equiv \{x_\alpha \mid x \in \mathfrak{V} \text{ and } \alpha \text{ is a closed type}\}.$$

The preceding definition captures the intuition that a declaration for a polymorphic constant stands for the collection of declarations for the (monomorphic) constants that can be obtained by instantiating all parameters in the polymorphic declaration with closed types. The constant C_β in $\overline{\mathfrak{C}}$ has signature β (by definition). If $C \in \mathfrak{C}$ has closed signature α, then $C_\alpha \in \overline{\mathfrak{C}}$ also has signature α, and this is the only (monomorphic) constant that underlies C. By definition, a variable $x_\alpha \in \overline{\mathfrak{V}}$ has type α. The set $\overline{\mathfrak{V}}$ thus contains an infinite set of variables of each type.

Definition 4.15. Let t be a term. A *grounding* type substitution for t is closed type substitution whose domain includes the set of all parameters in the relative types of subterms of t.

Definition 4.16. Let A be an alphabet in the sense of Definition 4.1, t be a term using A in the sense of Definition 4.12, and γ a grounding type substitution for t. Then the expression t^γ using symbols from the alphabet $\overline{A}$ is defined inductively as follows:

1. If t is a variable x of type a, then x^γ is the variable $x_{a\gamma}$ in the alphabet $\overline{A}$.
2. If t is a constant constant C having signature α, then C^γ is the constant $C_{\alpha\gamma}$ in the alphabet $\overline{A}$.
3. If t is an abstraction $\lambda x.s$ and x is not free in s, then $(\lambda x.s)^\gamma$ is $\lambda x^\gamma.s^\gamma$; if x is free with relative type α in s, then $(\lambda x.s)^\gamma$ is $\lambda x^{\gamma'}.s^\gamma$, where x has type a and $\gamma' = \{a/\alpha\}\gamma$.
4. If t is an application $(u\ v)$, then $(u\ v)^\gamma$ is $(u^{\xi\gamma}\ v^{\xi\gamma})$, where ξ is the associated mgu.
5. If t is a tuple $(t_1, \ldots, t_n)$, then $(t_1, \ldots, t_n)^\gamma$ is $(t_1^{\xi\gamma}, \ldots, t_n^{\xi\gamma})$, where ξ is the associated mgu.
6. If t is a box term $\Box_i s$, then $(\Box_i s)^\gamma$ is $\Box_i s^\gamma$.

The next result shows that t^γ is a term in the sense of Definition 2.7.

Proposition 4.1. *Let A be an alphabet in the sense of Definition 4.1, t a term of type α in the sense of Definition 4.12 using A, and γ a grounding type substitution for t. Then t^γ is a term of type $\alpha\gamma$ in the sense of Definition 2.1 using the alphabet $\overline{A}$.*

Proof. The proof uses the principle of inductive construction given in Proposition A.4. To apply this result, it is necessary to restate Definition 4.16 more formally as the definition of a certain function. To set this up, let $\mathfrak{L}$ be the set of terms given by Definition 4.12 for the alphabet A and $\overline{\mathfrak{L}}$ the set of terms given by Definition 2.7 for the alphabet $\overline{A}$. Denote by $\mathfrak{G}$ the set of type substitutions. Let $\prec$ be the substring relation on $\mathfrak{L}$ and extend this to a relation $\prec_2$ on $\mathfrak{L} \times \mathfrak{G}$ by $(s, \theta) \prec_2 (t, \psi)$ if $s \prec t$. Clearly $\prec_2$ is a well-founded order on $\mathfrak{L} \times \mathfrak{G}$

Put

$$\mathfrak{LG} = \{(t, \gamma) \mid t \in \mathfrak{L} \text{ and } \gamma \text{ is a grounding type substitution for } t\}.$$

Since $\mathfrak{LG}$ is a subset of $\mathfrak{L} \times \mathfrak{G}$, $\prec_2$ is a well-founded order on $\mathfrak{LG}$. The minimal elements in $\mathfrak{LG}$ are tuples (t, θ), where t is a variable or a constant.

I now define a function

$$U : \mathfrak{LG} \to \overline{\mathfrak{L}}.$$

For this, two partitions are needed. Let $\mathfrak{S}$ be the set of closed types. For each $\alpha \in \mathfrak{S}$, let

$$\mathfrak{LG}_\alpha = \{(t, \gamma) \mid t \in \mathfrak{L}_\beta, \gamma \text{ is a grounding type substitution for } t \text{ and } \alpha = \beta\gamma\}.$$

Also, for each $\alpha \in \mathfrak{S}$, let $\overline{\mathfrak{L}}_\alpha$ be the set of terms in $\overline{\mathfrak{L}}$ of type α. Then $\{\mathfrak{LG}_\alpha\}_{\alpha\in\mathfrak{S}}$ is a partition of $\mathfrak{LG}$ and $\{\overline{\mathfrak{L}}_\alpha\}_{\alpha\in\mathfrak{S}}$ is a partition of $\overline{\mathfrak{L}}$. Now, using Proposition A.4, $U(t, \gamma)$ is, in effect, defined to be t^γ, for each $(t, \gamma) \in \mathfrak{LG}$.

First, consider the minimal elements in $\mathfrak{LG}$. If t is a variable x of type a, then define $U(x, \gamma)$ to be the variable $x_{a\gamma}$ in the alphabet $\overline{A}$. Also the consistency condition is satisfied since $(x, \gamma) \in \mathfrak{LG}_{a\gamma}$ implies that $U(x, \gamma) \in \overline{\mathfrak{L}}_{a\gamma}$. If t is a constant C having signature α, then define $U(C, \gamma)$ to be the constant $C_{\alpha\gamma}$ in the alphabet $\overline{A}$. Once again the consistency is satisfied since $(x, \gamma) \in \mathfrak{LG}_{a\gamma}$ implies that $U(C, \gamma) \in \overline{\mathfrak{L}}_{a\gamma}$.

Next consider the non-minimal elements in $\mathfrak{LG}$. For these, the first component of the pair is an abstraction, application, tuple or box term.

Let t be an abstraction $\lambda x.s$, where x is not free in s and s has type β. Thus $\lambda x.s$ has type $a \to \beta$, where x has type a. Now $(s, \gamma) \in \mathfrak{LG}_{\beta\gamma}$ and $(s, \gamma) \prec_2 (\lambda x.s, \gamma)$. Assume that (s, γ) is consistent so that $U(s, \gamma) \in \overline{\mathfrak{L}}_{\beta\gamma}$. Now define $U(\lambda x.s, \gamma)$ to be $\lambda x^\gamma.s^\gamma$. This uniquely defines U on $(\lambda x.s, \gamma)$ and, furthermore, $(\lambda x.s, \gamma)$ is consistent since $(\lambda x.s, \gamma) \in \mathfrak{LG}_{(a\to\beta)\gamma}$ and $U(\lambda x.s, \gamma) \in \overline{\mathfrak{L}}_{(a\to\beta)\gamma}$.

Let t be an abstraction $\lambda x.s$, where x is free with relative type α in s and s has type β. Thus $\lambda x.s$ has type $\alpha \to \beta$. Now $(s, \gamma) \in \mathfrak{LG}_{\beta\gamma}$ and $(s, \gamma) \prec_2 (\lambda x.s, \gamma)$. Assume that (s, γ) is consistent so that $U(s, \gamma) \in \overline{\mathfrak{L}}_{\beta\gamma}$. Now define $U(\lambda x.s, \gamma)$ to be $\lambda x^{\gamma'}.s^\gamma$, where x has type a and $\gamma' = \{a/\alpha\}\gamma$. This uniquely defines U on $(\lambda x.s, \gamma)$ and, furthermore, $(\lambda x.s, \gamma)$ is consistent since $(\lambda x.s, \gamma) \in \mathfrak{LG}_{(\alpha\to\beta)\gamma}$ and $U(\lambda x.s, \gamma) \in \overline{\mathfrak{L}}_{(\alpha\to\beta)\gamma}$.

Let t be an application $(u\ v)$, where u has type α, v has type β, and ξ is the associated mgu. Thus ξ is an mgu of the equation $\alpha = \beta \to b$ and any additional equations that come from free variables common to u and v, and the type of $(u\ v)$ is $b\xi$. Now $(u, \xi\gamma) \in \mathfrak{LG}_{\alpha\xi\gamma}$ and $(u, \xi\gamma) \prec_2 ((u\ v), \gamma)$. Similarly, $(v, \xi\gamma) \in \mathfrak{LG}_{\beta\xi\gamma}$ and $(v, \xi\gamma) \prec_2 ((u\ v), \gamma)$. Assume that $(u, \xi\gamma)$ is consistent, so that $U(u, \xi\gamma) \in \overline{\mathfrak{L}}_{\alpha\xi\gamma}$ and that $(v, \xi\gamma)$ is consistent, so that $U(v, \xi\gamma) \in \overline{\mathfrak{L}}_{\beta\xi\gamma}$. Now define $U((u\ v), \gamma)$ to be $(u^{\xi\gamma}\ v^{\xi\gamma})$. This uniquely defines U on $((u\ v), \gamma)$ and, furthermore, $((u\ v), \gamma)$ is consistent since $((u\ v), \gamma) \in \mathfrak{LG}_{b\xi\gamma}$ and $U((u\ v), \gamma) \in \overline{\mathfrak{L}}_{b\xi\gamma}$.

Let t be a tuple $(t_1, \ldots, t_n)$, where t_i has type α_i, for $i = 1, \ldots, n$, and ξ the associated mgu. Thus the type of $(t_1, \ldots, t_n)$ is $\alpha_1\xi \times \cdots \times \alpha_n\xi$. Now $(t_i, \xi\gamma) \in \mathfrak{LG}_{\alpha_i\xi\gamma}$ and $(t_i, \xi\gamma) \prec_2 ((t_1, \ldots, t_n), \gamma)$, for $i = 1, \ldots, n$. Assume that $(t_i, \xi\gamma)$ is consistent, so that $U(t_i, \xi\gamma) \in \overline{\mathfrak{L}}_{\alpha_i\xi\gamma}$, for $i = 1, \ldots, n$. Now define $U((t_1, \ldots, t_n), \gamma)$ to be $(t_1^{\xi\gamma}, \ldots, t_n^{\xi\gamma})$. This uniquely defines U on $((t_1, \ldots, t_n), \gamma)$ and, furthermore, $((t_1, \ldots, t_n), \gamma)$ is consistent since $((t_1, \ldots, t_n), \gamma) \in \mathfrak{LG}_{(\alpha_1\xi\times\cdots\times\alpha_n\xi)\gamma}$ and $U((t_1, \ldots, t_n), \gamma) \in \overline{\mathfrak{L}}_{(\alpha_1\xi\times\cdots\times\alpha_n\xi)\gamma}$.

Let t be a box term $\Box_i s$, for some $i \in \{1, \ldots, m\}$, where s has type α. Now $(s, \gamma) \in \mathfrak{LG}_{\alpha\gamma}$ and $(s, \gamma) \prec_2 (\Box_i s, \gamma)$. Assume that (s, γ) is consistent, so that $U(s, \gamma) \in \overline{\mathfrak{L}}_{\alpha\gamma}$. Now define

$U(\Box_i s, \gamma)$ to be $\Box_i s^\gamma$. This uniquely defines U on $(\Box_i s, \gamma)$ and, furthermore, $(\Box_i s, \gamma)$ is consistent since $(\Box_i s, \gamma) \in \mathfrak{LG}_{\alpha\gamma}$ and $U(\Box_i s, \gamma) \in \overline{\mathfrak{L}}_{\alpha\gamma}$.

Proposition A.4 now shows that U is uniquely defined. Furthermore, $U(\mathfrak{LG}_\alpha) \subseteq \overline{\mathfrak{L}}_\alpha$, for each $\alpha \in \mathfrak{S}$. In other words, if t a term of type α in the sense of Definition 4.12 using A and γ a grounding type substitution for t, then t^γ is a term of type $\alpha\gamma$ in the sense of Definition 2.1 using the alphabet $\overline{A}$. □

Definition 4.17. Let t be a term in the sense of Definition 4.12 and γ a grounding type substitution for t. Then t^γ is called the *underlying monomorphic term* of t with respect to γ.

It is instructive to study the properties of the function U in Proposition 4.1 on subterms.

Proposition 4.2. *Let t be a term in the sense of Definition 4.12 and γ a grounding type substitution for t. If o is an occurrence in t, then o is also an occurrence in t^γ. Furthermore, if s is a subterm of relative type σ at occurrence o in t, then the subterm of t^γ at occurrence o has type $\sigma\gamma$.*

Proof. The first part is proved by induction on the length of the occurrence o in t. If the length of o is 0, then o is ε which is also an occurrence in t^γ. Suppose next that the result holds for occurrences of length n and o is an occurrence in t of length $n+1$. Thus t must be an abstraction, application, tuple or modal term. Suppose that t is a tuple $(t_1, \ldots, t_n)$ and $o = io'$, for some $i \in \{1, \ldots, n\}$. Now $(t_1, \ldots, t_n)^\gamma = (t_1^{\xi\gamma}, \ldots, t_n^{\xi\gamma})$, where ξ is the associated mgu. By the induction hypothesis, o' is an occurrence in $t_i^{\xi\gamma}$. Hence o is an occurrence in $(t_1, \ldots, t_n)^\gamma$. The other cases are similar.

The second part is also proved by induction on the length of the occurrence o in t. If the length of o is 0, then s is t and the subterm at occurrence o in t^γ, which has type $\alpha\gamma$, where α is the type of t, by Proposition 4.1. Suppose next that the result holds for occurrences of length n and o is an occurrence in t of length $n+1$. Thus t must be an abstraction, application, tuple or modal term. Suppose that t is a tuple $(t_1, \ldots, t_n)$ and $o = io'$, for some $i \in \{1, \ldots, n\}$. Then s is a subterm at occurrence o' of t_i of relative type σ', where $\sigma = \sigma'\xi$. Now $(t_1, \ldots, t_n)^\gamma = (t_1^{\xi\gamma}, \ldots, t_n^{\xi\gamma})$, where ξ is the associated mgu. By the induction hypothesis, the subterm of $t_i^{\xi\gamma}$ at occurrence o' has type $\sigma'\xi\gamma = \sigma\gamma$. That is, the subterm at occurrence o in t^γ has type $\sigma\gamma$. The other cases are similar. □

In the following, when needed, I will declare signatures of polymorphic constants and give definitions of polymorphic functions. So, for example, the polymorphic function *append* having signature

$$append \ : \ List\ a \times List\ a \times List\ a \to \Omega$$

will be useful because appending lists is independent of the type of the items in the list. This declaration is thought of as a way of introducing the set of (monomorphic) functions of the form $append_{List\ \alpha \times List\ \alpha \times List\ \alpha \to \Omega}$, where α ranges over all closed types. Generally, the type subscript is suppressed and the actual 'instance' of *append* intended is determined by the context. Similarly, one can give a polymorphic definition of the function *append* as follows.

$$(append\ (u, v, w)) =$$
$$((u = []) \wedge (v = w)) \vee \exists r.\exists x.\exists y.((u = r \sharp x) \wedge (w = r \sharp y) \wedge (append\ (x, v, y)))$$

This is thought of as providing a definition for each of the underlying monomorphic functions by applying the appropriate grounding type substitution to the above definition.

The polymorphic syntax is extremely convenient and will be heavily exploited in the subsequent development because many common data constructors and functions are (parametrically) polymorphic. Furthermore, (non-nullary) type constructors are only interesting in the polymorphic case when they can take parameters as arguments. However, there is a cost associated with the use of polymorphism which is the need to infer the types of the variables and polymorphic constants from the context. So, for example, the monomorphic function underlying *append* that appears in

$$(\mathit{append}\ (1 \sharp 2 \sharp [], 3 \sharp [], x))$$

can be inferred to be $\mathit{append}_{\mathit{List\ Int} \times \mathit{List\ Int} \times \mathit{List\ Int} \to \Omega}$, since 1, 2, and 3 have type *Int*.

Generally, I will use just the name itself of a polymorphic symbol (that is, constant or variable) in a term, relying on the context to make clear which underlying monomorphic symbol is intended. For some purposes, it may be helpful to make explicit the grounding type substitution that is intended to be applied. For example, the term $(\mathit{append}\ (x, y, z))$ on its own does not determine the type of the monomorphic function underlying *append*; this would then have to be given separately with a suitable grounding type substitution. An extreme case like this is that of a variable x of type a on its own that would need a grounding type substitution of the form $\{a/\alpha\}$ to specify the intended underlying monomorphic variable. It may also be helpful to explicitly indicate the underlying monomorphic symbol with a type subscript.

Note that the mixing of monomorphic and polymorphic syntax has been facilitated by using the same notation for variables in both the monomorphic and polymorphic cases. But the meanings in each case are somewhat different: in the monomorphic case, x stands for a variable of some specified type; in the polymorphic case, x takes the type inferred from its context. However, in the monomorphic case, since the syntax used for variables does not indicate their type, it effectively has to be inferred as well. Thus, while there is therefore potential for confusion about which kind of variable is intended in the mixed case, it does not really matter – in both cases, the type of the variable is inferred from its context (or, in extreme cases, specified using an explicit grounding type substitution).

5 Predicates

In this section, I study grammars for defining spaces of predicates that appear in hypothesis languages in learning applications. There are two parts to this: first I define a class of predicates, called standard predicates, that have a convenient form for use in grammars; then I define a predicate grammar formalism, called a predicate rewrite system, that generates standard predicates. The presentation here extends the one in [Llo03] to the modal case. More detail about how the concepts and results of this section can be exploited by learning systems is contained in [Ng05a, Ng05b, Ng06, LN07a].

5.1 Standard Predicates

Predicates are built up by composing basic functions called transformations. Composition is handled by the (reverse) composition function

$$\circ : (a \to b) \to (b \to c) \to (a \to c)$$

defined by $((f \circ g)\ x) = (g\ (f\ x))$.

Definition 5.1. A *transformation* f is a function having a signature of the form

$$f : (\varrho_1 \to \Omega) \to \cdots \to (\varrho_k \to \Omega) \to \mu \to \sigma,$$

where any parameters in $\varrho_1, \ldots, \varrho_k$ and σ appear in μ, and $k \geq 0$. The type σ is called the *target* of the transformation. The number k is called the *rank* of the transformation.

I now give some examples of transformations. Many more are given in [Llo03].

Example 5.1. The transformation $top : a \to \Omega$ is defined by $top\ x = \top$, for each x. The transformation $bottom : a \to \Omega$ is defined by $bottom\ x = \bot$, for each x.

Example 5.2. The transformation $\wedge_n : (a \to \Omega) \to \cdots \to (a \to \Omega) \to a \to \Omega$ defined by

$$\wedge_n\ p_1 \ldots p_n\ x = (p_1\ x) \wedge \cdots \wedge (p_n\ x),$$

where $n \geq 2$, provides a 'conjunction' with n conjuncts.

The transformation $\vee_n : (a \to \Omega) \to \cdots \to (a \to \Omega) \to a \to \Omega$ defined by

$$\vee_n\ p_1 \ldots p_n\ x = (p_1\ x) \vee \cdots \vee (p_n\ x),$$

where $n \geq 2$, provides a 'disjunction' with n disjuncts.

The transformation $\neg\ : (a \to \Omega) \to a \to \Omega$ defined by

$$\neg\ p\ x = \neg(p\ x),$$

provides negation.

Example 5.3. The transformation $setExists_1 : (a \to \Omega) \to \{a\} \to \Omega$ defined by

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

checks whether a set t has an element that satisfies p.

Example 5.4. Given a type of the form $a_1 \times a_2 \times \cdots \times a_n$, I can define, for each i, a transformation $proj_i : a_1 \times a_2 \times \cdots \times a_n \to a_i$ defined by

$$proj_i\ (t_1, t_2, \ldots, t_n) = t_i$$

to project out the i-th component.

Example 5.5. The function $isRectangle : (Float \to \Omega) \to (Float \to \Omega) \to Shape \to \Omega$ defined by

$$isRectangle\ p\ q\ t = \exists x.\exists y.((t = Rectangle\ x\ y) \wedge (p\ x) \wedge (q\ y))$$

is a transformation. For predicates p and q, $(isRectangle\ p\ q)$ is a predicate on geometrical shapes which returns $\top$ iff the shape is a rectangle whose length satisfies p and whose breadth satisfies q.

Next the definition of the class of predicates formed by composing transformations is presented. In the following definition, it is assumed that some (possibly infinite) class of transformations is given and all transformations considered are taken from this class. A standard predicate is defined by induction on the number of (occurrences of) transformations it contains as follows.

Notation. Let $\Box$ denote a (possibly empty) sequence of modalities $\Box_{j_1} \cdots \Box_{j_r}$.

Definition 5.2. A *standard predicate* is a term of the form

$$\Box_1(f_1\ p_{1,1} \ldots p_{1,k_1}) \circ \cdots \circ \Box_n(f_n\ p_{n,1} \ldots p_{n,k_n}),$$

where f_i is a transformation of rank k_i $(i = 1, \ldots, n)$, the target of f_n is Ω, $\Box_i$ is a sequence of modalities $(i = 1, \ldots, n)$, p_{i,j_i} is a standard predicate $(i = 1, \ldots, n,\ \ j_i = 1, \ldots, k_i)$, $k_i \geq 0$ $(i = 1, \ldots, n)$ and $n \geq 1$.

Definition 5.2 extends that of a (non-modal) standard predicate in [Llo03], precisely in that the definition here allows modalities to appear.

Example 5.6. Let p and q be transformations of type $\sigma \to \Omega$. Then

$$\boldsymbol{B}_i(\mathit{setExists}_1\ (\wedge_2\ \bullet\boldsymbol{B}_j p\ \blacklozenge\boldsymbol{B}_j q))$$

is a standard predicate of type $\{\sigma\} \to \Omega$. If t is a (rigid) set of elements of type σ, then

$$(\boldsymbol{B}_i(\mathit{setExists}_1\ (\wedge_2\ \bullet\boldsymbol{B}_j p\ \blacklozenge\boldsymbol{B}_j q))\ \ t)$$

simplifies to

$$\boldsymbol{B}_i \exists x.((\bullet\boldsymbol{B}_j(p\ x) \wedge \blacklozenge\boldsymbol{B}_j(q\ x)) \wedge (x \in t)),$$

which is true iff agent i believes that there is an element x in t satisfying the property that at the last time agent j believed that x satisfied p and at some time in the past agent j believed that x satisfied q.

The set of all standard predicates is denoted by $\mathbf{S}$. Standard predicates have type of the form $\mu \to \Omega$, for some type μ. Note that the set of standard predicates is defined *relative to* a previously declared set of transformations.

Definition 5.3. For each $\alpha \in \mathfrak{S}^c$, define $\mathbf{S}_\alpha = \{p \in \mathbf{S} \mid p \text{ has type } \mu \to \Omega \text{ and } \mu \text{ is more general than } \alpha\}$.

The intuitive meaning of $\mathbf{S}_\alpha$ is that it is the set of all predicates of a particular form given by the transformations on individuals of type α.

Careful attention needs to be paid to the subterms of a standard predicate, especially the subterms that are themselves standard predicates.

Definition 5.4. Let $\Box_1(f_1\ p_{1,1} \ldots p_{1,k_1}) \circ \cdots \circ \Box_n(f_n\ p_{n,1} \ldots p_{n,k_n})$ be a standard predicate. A *suffix* of the standard predicate is a term of the form

$$\Box_i(f_i\ p_{i,1} \ldots p_{i,k_i}) \circ \cdots \circ \Box_n(f_n\ p_{n,1} \ldots p_{n,k_n}),$$

for some $i \in \{1, \ldots, n\}$. The suffix is *proper* if $i > 1$.

A *prefix* of the standard predicate is a term of the form

$$\Box_1(f_1\ p_{1,1} \ldots p_{1,k_1}) \circ \cdots \circ \Box_i(f_i\ p_{i,1} \ldots p_{i,k_i}),$$

for some $i \in \{1, \ldots, n\}$. The prefix is *proper* if $i < n$.

A suffix of a standard predicate is a standard predicate, but a prefix of a standard predicate may not be a predicate.

Proposition 5.1. *Let p be a standard predicate*

$$\square_1(f_1\ p_{1,1}\dots p_{1,k_1})\circ\cdots\circ\square_n(f_n\ p_{n,1}\dots p_{n,k_n})$$

and q a term. Then q is a subterm of p iff (at least) one of the following conditions holds.

1. *q is a suffix of p.*
2. *q is a subterm of $(\circ\ \square_i(f_i\ p_{i,1}\dots p_{i,k_i}))$, for some $i\in\{1,\dots,n-1\}$.*
3. *q is a subterm of $\square_n(f_n\ p_{n,1}\dots p_{n,k_n})$,*

Proof. The proof is by induction on n. If $n=1$, the result is obvious. Consider now a standard predicate

$$\square_1(f_1\ p_{1,1}\dots p_{1,k_1})\circ\cdots\circ\square_{n+1}(f_{n+1}\ p_{n+1,1}\dots p_{n+1,k_{n+1}}),$$

which can be written in the form

$$((\circ\ \square_1(f_1\ p_{1,1}\dots p_{1,k_1}))\ \square_2(f_2\ p_{2,1}\dots p_{2,k_2})\circ\cdots\circ\square_{n+1}(f_{n+1}\ p_{n+1,1}\dots p_{n+1,k_{n+1}})).$$

A subterm of this is either the term itself or a subterm of $(\circ\ \square_1(f_1\ p_{1,1}\dots p_{1,k_1}))$ or a subterm of $\square_2(f_2\ p_{2,1}\dots p_{2,k_2})\circ\cdots\circ\square_{n+1}(f_{n+1}\ p_{n+1,1}\dots p_{n+1,k_{n+1}})$. The result follows by applying the induction hypothesis to this last term. □

Proposition 5.2. *Let p be a standard predicate*

$$\square_1(f_1\ p_{1,1}\dots p_{1,k_1})\circ\cdots\circ\square_n(f_n\ p_{n,1}\dots p_{n,k_n})$$

and q a subterm of p. Then q is a standard predicate iff (at least) one of the following conditions holds.

1. *q is a suffix of p.*
2. *q has the form $\square_i(f_i\ p_{i,1}\dots p_{i,k_i})$ or $\square_{i,2}\cdots\square_{i,l_i}(f_i\ p_{i,1}\dots p_{i,k_i})$ or $\dots$ or $(f_i\ p_{i,1}\dots p_{i,k_i})$, for some $i\in\{1,\dots,n\}$, and f_i has target Ω, where $\square_i$ is $\square_{i,1}\cdots\square_{i,l_i}$.*
3. *q is a subterm of $p_{i,j}$, for some i and j, and q is a standard predicate.*

Proof. According to Proposition 5.1, the subterms of p are the suffixes of p, the subterms of $(\circ\ \square_i(f_i\ p_{i,1}\dots p_{i,k_i}))$, for $i=1,\dots,n-1$, and the subterms of $\square_n(f_n\ p_{n,1}\dots p_{n,k_n})$, Thus it is only necessary to establish which of these subterms are standard predicates. First, all the suffixes are standard predicates. Since $\circ$ is not a standard predicate, it remains to investigate the subterms of $\square_i(f_i\ p_{i,1}\dots p_{i,k_i})$, for $i=1,\dots,n$. Now, provided the target of f_i is Ω, $\square_i(f_i\ p_{i,1}\dots p_{i,k_i})$ and $\square_{i,2}\cdots\square_{i,l_i}(f_i\ p_{i,1}\dots p_{i,k_i})$ and $\dots$ and $(f_i\ p_{i,1}\dots p_{i,k_i})$ are standard predicates. Also the only proper subterms of $(f_i\ p_{i,1}\dots p_{i,k_i})$ that could possibly be standard predicates are subterms of $p_{i,1},\dots,p_{i,k_i}$. (The reason is that for f_i to be the first symbol in a standard predicate, it must be followed by *all* k of its predicate arguments.) The result follows. □

Example 5.7. Consider the standard predicate

$$(f_1\ p)\circ(f_2\ q\ r)\circ(f_3\ s),$$

where the transformations f_1, f_2, f_3, p, q, r, and s have suitable signatures. Thus p, q, r, and s are predicates. Suppose also that neither f_1 nor f_2 have target Ω. Taking the associativity into account, this standard predicate can be written more precisely as

$$((\circ(f_1\ p))\ \ ((\circ((f_2\ q)\ r))\ \ (f_3\ s))).$$

The subterms of $(f_1\ p)\circ(f_2\ q\ r)\circ(f_3\ s)$ are illustrated in Fig. 2. The proper subterms that are standard predicates are $(f_2\ q\ r)\circ(f_3\ s)$, $(f_3\ s)$, p, q, r, and s.

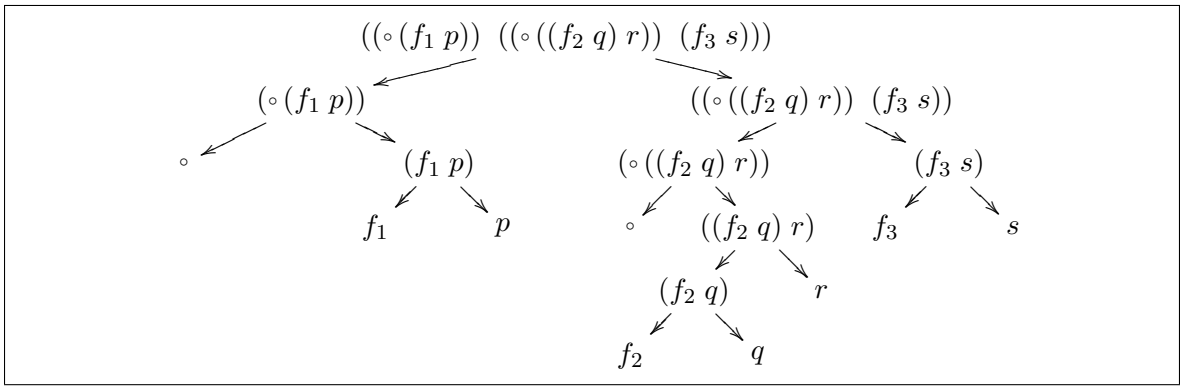

Figure 2: Subterms of a standard predicate

The class of standard predicates just defined contains some redundancy in that there are syntactically distinct predicates that are equivalent, in a certain sense.

Definition 5.5. The theory consisting of the definitions of the transformations (and associated functions) is called the *background theory*.

In an agent application, the background theory would normally be the belief base of an agent (or, perhaps, the collected belief bases of several agents).

Definition 5.6. Let $\mathcal{B}$ be the background theory, and s and t terms having the same type (up to variants). Then s and t are *equivalent with respect to* $\mathcal{B}$ if $\Box(s = t)$ is a consequence of $\mathcal{B}$, for every sequence of modalities $\Box$.

Equivalence with respect to a background theory is an equivalence relation on the set of terms. When discussing equivalence in the following, explicit mention of the background theory is usually suppressed.

Example 5.8. Without spelling out the background theory, one would expect that the standard predicates $(\wedge_2\ p\ q)$ and $(\wedge_2\ q\ p)$ are equivalent, where p and q are standard predicates. Similarly, $(\vee_2\ p\ q)$ and $(\vee_2\ q\ p)$ are equivalent.

It is important to remove as much of this redundancy in standard predicates as possible. Ideally, one would like to be able to determine just *one* representative from each class of equivalent predicates. However, determining the equivalent predicates is undecidable, so one usually settles for some easily checked syntactic conditions that reveal equivalence of predicates. Thus these syntactic conditions are sufficient, but not necessary, for equivalence. These considerations motivate the next definition.

Definition 5.7. A transformation f is *symmetric* if it has a signature of the form

$$f : (\varrho \to \Omega) \to \cdots \to (\varrho \to \Omega) \to \mu \to \sigma,$$

and, whenever $(f\ p_1 \dots p_k)$ is a term, where $p_1, \dots, p_k$ are standard predicates, it follows that $(f\ p_1 \dots p_k)$ and $(f\ p_{i_1} \dots p_{i_k})$ are equivalent, for all permutations i of $\{1, \dots, k\}$, where k is the rank of f.

Note that if $(f\ p_1 \dots p_k)$ is a term, then $(f\ p_{i_1} \dots p_{i_k})$ is a term of the same type. It is common for a symmetric transformation to have Ω as its target; however, this does not have to be the case. Clearly every transformation of rank k, where $k \leq 1$, is (trivially) symmetric. Furthermore, the transformations $\wedge_n$ and $\vee_n$ are symmetric. However, *isRectangle* is not symmetric since it distinguishes the length argument from the breadth argument.

Since any permutation of the predicate arguments of a symmetric transformation produces an equivalent function, it is advisable to choose one particular order of arguments and ignore the others. For this purpose, a total order on standard predicates is defined and then arguments for symmetric transformations are chosen in increasing order according to this total order. To define the total order on standard predicates, one must start with a total order on modalities and transformations. Therefore, it is supposed that the modalities $\Box_1, \dots, \Box_m$ are ordered according to some (arbitrary) strict total order $<$. This order can can lifted to the strict total order also denoted by $<$ on (finite) sequences of modalities given by the lexicographic order. Furthermore, transformations are ordered according to some (arbitrary) strict total order also denoted by $<$.

In preparation for the definition of $\prec$, the total order on standard predicates, a structural result about standard predicates is needed.

Proposition 5.3. *Let $p, q \in \mathbf{S}$, where p is*

$$\Box_{p,1}(f_1\ p_{1,1} \dots p_{1,k_1}) \circ \cdots \circ \Box_{p,n}(f_n\ p_{n,1} \dots p_{n,k_n})$$

and q is

$$\Box_{q,1}(g_1\ q_{1,1} \dots q_{1,s_1}) \circ \cdots \circ \Box_{q,r}(g_r\ q_{r,1} \dots q_{r,s_r}).$$

Suppose that p and q are (syntactically) distinct. Then exactly one of the following alternatives holds.

1. *One of p or q is a proper prefix of the other.*
2. *There exists i such that $\Box_{p,i}$ and $\Box_{q,i}$ are distinct, and p and q agree to the left of $\Box_{p,i}$ and $\Box_{q,i}$.*
3. *There exists i such that the transformation f_i in p and the transformation g_i in q are distinct, and p and q agree to the left of f_i and g_i.*
4. *There exist i and j such that $p_{i,j}$ in p and $q_{i,j}$ in q are distinct, and p and q agree to the left of $p_{i,j}$ and $q_{i,j}$.*

Proof. The proof is by induction on the maximum of the number of (occurrences of) transformations in p and the number in q.

If this maximum number is one, then the result is obvious with the second and third alternatives being the only possibilities.

Now consider the inductive step. Suppose first that $\Box_{p,1}$ and $\Box_{q,1}$ are distinct. Then the second alternative holds. Suppose next that $\Box_{p,1}$ and $\Box_{q,1}$ are identical, but f_1 is distinct from g_1. Then the third alternative holds. Otherwise, f_1 and g_1 are identical, which gives rise to two cases: either the arguments to f_1 in p and g_1 in q are pairwise identical or they are not. In the first case, consider the suffixes of p and q obtained by removing the common prefix $\Box_{p,1}(f_1\ p_{1,1} \dots p_{1,k_1})$. If one of the suffixes is empty, then the first alternative holds; otherwise, the inductive hypothesis can be applied to the suffixes to obtain the result. In the other case, the leftmost pair of $p_{1,j}$ and $q_{1,j}$ that are distinct gives the fourth alternative. □

Example 5.9. As an illustration of the first alternative in the preceding proposition, consider $\Box_1 p$ and $\Box_1 p \circ \neg$, for some standard predicate p. For the second alternative, consider $\Box_1 p$ and $\Box_2 p$, for some standard predicate p. For the fourth alternative, consider $\Box_1(\wedge_2\ p\ q)$ and $\Box_1(\wedge_2\ (p \circ \neg)\ q)$, for some standard predicates p and q.

The following definition of the relation $p \prec q$ uses induction on the maximum of the number of (occurrences of) transformations in p and the number in q. To emphasise: the definition of $\prec$ depends upon the order on the modalities and the order on the transformations.

Definition 5.8. The binary relation $\prec$ on $\mathbf{S}$ is defined inductively as follows. Let $p, q \in \mathbf{S}$, where p is

$$\Box_{p,1}(f_1\ p_{1,1} \dots p_{1,k_1}) \circ \dots \circ \Box_{p,n}(f_n\ p_{n,1} \dots p_{n,k_n})$$

and q is

$$\Box_{q,1}(g_1\ q_{1,1} \dots q_{1,s_1}) \circ \dots \circ \Box_{q,r}(g_r\ q_{r,1} \dots q_{r,s_r}).$$

Then $p \prec q$ if one of the following holds.

1. p is a proper prefix of q.
2. There exists an i such that $\Box_{p,i} < \Box_{q,i}$ and p and q agree to the left of $\Box_{p,i}$ and $\Box_{q,i}$.
3. There exists i such that $f_i < g_i$, and p and q agree to the left of f_i and g_i.
4. There exist i and j such that $p_{i,j} \prec q_{i,j}$, and p and q agree to the left of $p_{i,j}$ and $q_{i,j}$.

Proposition 5.4. *The relation $\prec$ is a strict total order on $\mathbf{S}$.*

Proof. Let $p, q, r \in \mathbf{S}$, where p is

$$\Box_{p,1}(f_1\ p_{1,1} \dots p_{1,k_1}) \circ \dots \circ \Box_{p,n}(f_n\ p_{n,1} \dots p_{n,k_n}),$$

q is

$$\Box_{q,1}(g_1\ q_{1,1} \dots q_{1,s_1}) \circ \dots \circ \Box_{q,u}(g_u\ q_{u,1} \dots q_{u,s_u})$$

and r is

$$\Box_{r,1}(h_1\ r_{1,1} \dots r_{1,t_1}) \circ \dots \circ \Box_{r,v}(h_v\ r_{v,1} \dots r_{v,t_v}).$$

First, I show by induction on the number of (occurrences of) transformations in p that $p \not\prec p$. For this, note that p cannot be a proper prefix of itself. Also there cannot exist i such that $\Box_{p,i} < \Box_{p,i}$, since the lexicographic order is a strict total order on sequences of modalities, and there cannot exist i such that $f_i < f_i$, since $<$ is a strict total order. Furthermore, there cannot exist i and j such that $p_{i,j} \prec p_{i,j}$, by the induction hypothesis.

Next I show by induction on the maximum of the number of transformations in p and the number in q that $p \prec q$ implies $q \not\prec p$. Thus suppose that $p \prec q$. If p is a proper prefix of q, then it is clear that $q \not\prec p$. If there exists an i such that $\Box_{p,i} < \Box_{q,i}$ and p and q agree to the left of $\Box_{p,i}$ and $\Box_{q,i}$, then it is clear that $q \not\prec p$, since the lexicographic order $<$ is a strict total order. If there exists i such that $f_i < g_i$, and p and q agree to the left of f_i and g_i, then it is clear that $q \not\prec p$, since $<$ is a strict total order on transformations. Finally, if there exist i and j such that $p_{i,j} \prec q_{i,j}$, and p and q agree to the left of $p_{i,j}$ and $q_{i,j}$, then $q \not\prec p$ since, by the induction hypothesis, it follows that $q_{i,j} \not\prec p_{i,j}$.

Now I show by induction on the maximum of the number of transformations in p, the number in q, and the number in r that $p \prec q$ and $q \prec r$ imply that $p \prec r$.

For the base step, each of p, q and r contains a single transformation. If the sequences of modalities for the three standard predicates are not all the same, then the result follows immediately from the transitivity of the lexicographic order on sequences of modalities. Otherwise, the result follows immediately from the transitivity of the order on transformations.

Now I consider the inductive step. There are four cases to consider, corresponding to the four cases in the definition of $p \prec q$.

Suppose first that p is a proper prefix of q. If q is a proper prefix of r, then p is a proper prefix of r and so $p \prec r$. If there exists i such that such that $\Box_{q,i} < \Box_{r,i}$ and q and r agree to the left of $\Box_{q,i}$ and $\Box_{r,i}$, then either p is a proper prefix of r or $\Box_{p,i} < \Box_{r,i}$ and p and r agree to the left of $\Box_{p,i}$ and $\Box_{r,i}$, so that $p \prec r$. If there exists i such that $g_i < h_i$, and q and r agree to the left of g_i and h_i, then either p is a proper prefix of r or $f_i < h_i$ and p and r agree to the left of f_i and h_i, and so $p \prec r$. If there exist i and j such that $q_{i,j} \prec r_{i,j}$, and q and r agree to the left of $q_{i,j}$ and $r_{i,j}$, then either p is a proper prefix of r or $p_{i,j} \prec r_{i,j}$ and p and r agree to the left of $p_{i,j}$ and $r_{i,j}$, and so $p \prec r$.

For the second case, suppose there exists an i such that $\Box_{p,i} < \Box_{q,i}$ and p and q agree to the left of $\Box_{p,i}$ and $\Box_{q,i}$. If q is a proper prefix of r, then $\Box_{p,i} < \Box_{r,i}$ and p and r agree up to this point, so that $p \prec r$. If there exists an i' such that $\Box_{q,i'} < \Box_{r,i'}$ and q and r agree to the left of $\Box_{q,i'}$ and $\Box_{r,i'}$, then if $i \neq i'$, clearly I have that $p \prec r$, and if $i = i'$, then $p \prec r$, by the transitivity of the lexicographic order $<$ on sequences of modalities. If there exists i' such that $g_{i'} < h_{i'}$, and q and r agree to the left of $g_{i'}$ and $h_{i'}$, then if $i > i'$, then $f_{i'} < h_{i'}$ and p and r agree up to this point, so that $p \prec r$, and if $i \leq i'$, then $\Box_{p,i} < \Box_{r,i}$ and p and r agree up to this point, so that $p \prec r$. If there exist i' and j such that $q_{i',j} \prec r_{i',j}$, and q and r agree to the left of $q_{i',j}$ and $r_{i',j}$, then if $i > i'$, then $p_{i',j} \prec r_{i',j}$, and p and r agree up to this point, so that $p \prec r$, and if $i \leq i'$, then $\Box_{p,i} < \Box_{r,i}$ and p and r agree up to this point, so that $p \prec r$.

For the third case, suppose there exists i such that $f_i < g_i$, and p and q agree to the left of f_i and g_i. If q is a proper prefix of r, then $f_i < h_i$, and p and r agree to the left of f_i and h_i, and so $p \prec r$. If there exists i' such that $\Box_{q,i'} < \Box_{r,i'}$ and q and r agree to the left of $\Box_{q,i'}$ and $\Box_{r,i'}$, then, if $i \geq i'$, then $\Box_{p,i'} < \Box_{r,i'}$ and p and r agree to the left of $\Box_{p,i'}$ and $\Box_{r,i'}$, so that $p \prec r$, and if $i < i'$, then $f_i < h_i$, and p and r agree to the left of f_i and h_i, so that $p \prec r$. If there exists i' such that $g_{i'} < h_{i'}$, and q and r agree to the left of $g_{i'}$ and $h_{i'}$, then, for $i'' = min(i, i')$, $f_{i''} < h_{i''}$ and p and r agree to the left of $f_{i''}$ and $h_{i''}$, and so $p \prec r$. Finally, suppose that there exist i' and j such that $q_{i',j} \prec r_{i',j}$, and q and r agree to the left

of $q_{i',j}$ and $r_{i',j}$. If $i \leq i'$, then $f_i < h_i$ and p and r agree to the left of f_i and h_i. If $i' < i$, then $p_{i',j} \prec r_{i',j}$ and p and r agree to the left of $p_{i',j}$ and $r_{i',j}$. In either case, $p \prec r$.

For the fourth case, suppose there exist i and j such that $p_{i,j} \prec q_{i,j}$, and p and q agree to the left of $p_{i,j}$ and $q_{i,j}$. If q is a proper prefix of r, then $p_{i,j} \prec r_{i,j}$, and p and r agree to the left of $p_{i,j}$ and $r_{i,j}$, and so $p \prec r$. If there exists i' such that $\Box_{q,i'} < \Box_{r,i'}$ and q and r agree to the left of $\Box_{q,i'}$ and $\Box_{r,i'}$, then if $i \geq i'$, then $\Box_{p,i'} < \Box_{r,i'}$ and p and r agree to the left of $\Box_{p,i'}$ and $\Box_{r,i'}$, so that $p \prec r$, and if $i < i'$, then $p_{i,j} \prec r_{i,j}$, and p and r agree to the left of $p_{i,j}$ and $r_{i,j}$, so that $p \prec r$. Suppose that there exists i' such that $g_{i'} < h_{i'}$, and q and r agree to the left of $g_{i'}$ and $h_{i'}$. If $i' \leq i$, then $f_{i'} < h_{i'}$ and p and r agree to the left of $f_{i'}$ and $h_{i'}$. If $i < i'$, then $p_{i,j_i} \prec r_{i,j_i}$ and p and r agree to the left of p_{i,j_i} and r_{i,j_i}. In either case, $p \prec r$. Finally, suppose there exist i' and j' such that $q_{i',j'} \prec r_{i',j'}$, and q and r agree to the left of $q_{i',j'}$ and $r_{i',j'}$. Put $i'' = min(i, i')$ and let j'' be j if $i < i'$, or j' if $i' < i$, or $min(j, j')$, otherwise. Then $p_{i'',j''} \prec r_{i'',j''}$ and p and r agree to the left of $p_{i'',j''}$ and $r_{i'',j''}$, and so $p \prec r$. (Here the induction hypothesis is used if $i = i'$ and $j = j'$.)

Thus $\prec$ is a strict partial order.

Finally, I show that $\prec$ is total, that is, for any standard predicates p and q, either $p = q$ or $p \prec q$ or $q \prec p$. The proof proceeds by induction on the maximum of the number of transformations in p and the number in q. The base step is straightforward. Now I consider the inductive step. Suppose that p and q are distinct. By Proposition 5.3, either one of p or q is a proper prefix of the other; or there exists i such that $\Box_{p,i}$ and $\Box_{q,i}$ are distinct, and p and q agree to the left of $\Box_{p,i}$ and $\Box_{q,i}$; or there exists i such that the transformation f_i in p and the transformation g_i in q are distinct, and p and q agree to the left of f_i and g_i; or there exist i and j such that $p_{i,j}$ in p and $q_{i,j}$ in q are distinct, and p and q agree to the left of $p_{i,j}$ and $q_{i,j}$. In the first case, either $p \prec q$ or $q \prec p$. In the second case, either $p \prec q$ or $q \prec p$, since the lexicographic ordering on sequences of modalities is a total order. In the third case, since $<$ is a total order on transformations, either $f_i < g_i$ or $g_i < f_i$, and hence either $p \prec q$ or $q \prec p$. In the fourth case, by the induction hypothesis, either $p_{i,j} \prec q_{i,j}$ or $q_{i,j} \prec p_{i,j}$, and so once again either $p \prec q$ or $q \prec p$. □

The relation $\preceq$ is defined by $p \preceq q$ if either $p = q$ or $p \prec q$. Clearly, $\preceq$ is a total order on **S**.

Now the class of regular predicates can be defined by induction on the number of transformations in a predicate.

Definition 5.9. A standard predicate $\Box_1(f_1\ p_{1,1} \dots p_{1,k_1}) \circ \cdots \circ \Box_n(f_n\ p_{n,1} \dots p_{n,k_n})$ is *regular* if p_{i,j_i} is a regular predicate, for $i = 1, \dots, n$ and $j_i = 1, \dots, k_i$, and f_i is symmetric implies that $p_{i,1} \preceq \cdots \preceq p_{i,k_i}$, for $i = 1, \dots, n$.

The set of all regular predicates is denoted by **R**.

The next result shows that each standard predicate is equivalent to a regular predicate and hence attention can be confined to the generally much smaller class of regular predicates.

Proposition 5.5. *For every $p \in \mathbf{S}$, there exists $q \in \mathbf{R}$ such that p and q are equivalent.*

Proof. The existence of the regular predicate q is shown by induction on the number of transformations in p. Let p be $\Box_1(f_1\ p_{1,1} \dots p_{1,k_1}) \circ \cdots \circ \Box_n(f_n\ p_{n,1} \dots p_{n,k_n})$. By the induction hypothesis, for each $i = 1, \dots, n$ and $j_i = 1, \dots, k_i$, there is a regular predicate p'_{i,j_i} such that p'_{i,j_i} and p_{i,j_i} are equivalent. Let r be $\Box_1(f_1\ p'_{1,1} \dots p'_{1,k_1}) \circ \cdots \circ \Box_n(f_n\ p'_{n,1} \dots p'_{n,k_n})$. Then p and r are equivalent. Now, since $\preceq$ is a total order on standard predicates, for each symmetric

transformation f_i ($i \in \{1, \ldots, n\}$) in r, one can sort the arguments $p'_{i,1}, \ldots, p'_{i,k_i}$ according to $\preceq$ to obtain the desired regular predicate q such that p and q are equivalent. □

5.2 Predicate Rewrite Systems

This subsection gives the definition of the concept of a predicate rewrite system.

First, this is explained informally. A predicate rewrite is an expression of the form $p \rightarrowtail q$, where p and q are standard predicates. The predicate p is called the *head* and q is the *body* of the rewrite. A predicate rewrite system is a finite set of predicate rewrites. One should think of a predicate rewrite system as a kind of grammar for generating a particular class of predicates. Roughly speaking, this works as follows. Starting from the weakest predicate *top*, all predicate rewrites that have *top* (of the appropriate type) in the head are selected to make up child predicates that consist of the bodies of these predicate rewrites. Then, for each child predicate and each redex in that predicate, all child predicates are generated by replacing each redex by the body of the predicate rewrite whose head is identical to the redex. This generation of predicates continues to produce the entire space of predicates given by the predicate rewrite system.

Here is the formal development.

Definition 5.10. A *predicate rewrite system* is a finite relation $\rightarrowtail$ on $\mathbf{S}$ satisfying the following two properties.

1. For each $p \rightarrowtail q$, the type of p is more general than the type of q.
2. For each $p \rightarrowtail q$, there does not exist $s \rightarrowtail t$ such that q is a proper subterm of s.

If $p \rightarrowtail q$, then $p \rightarrowtail q$ is called a *predicate rewrite*, p the *head*, and q the *body* of the predicate rewrite.

The second condition of Definition 5.10 states that no body of a rewrite is a proper subterm of the head of any rewrite. In practice, the heads of rewrites are usually standard predicates consisting of just one transformation, such as *top*. In this case, the second condition is automatically satisfied, since the heads have no proper subterms at all.

Definition 5.11. A subterm of a standard predicate

$$\square_1(f_1\ p_{1,1} \ldots p_{1,k_1}) \circ \cdots \circ \square_n(f_n\ p_{n,1} \ldots p_{n,k_n})$$

is *eligible* if it is a suffix of the standard predicate or it is an eligible subterm of $p_{i,j}$, for some $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, k_i\}$.

Example 5.10. The eligible subterms of

$$\square_1 \mathit{vertices} \circ \square_1\square_2(\mathit{setExists}_2\ (\wedge_2\ (\mathit{proj}_1 \circ (= A))\ (\mathit{proj}_2 \circ (= 3)))\ (\mathit{proj}_1 \circ (= B)))$$

are

$\Box_1 \mathit{vertices} \circ \Box_1\Box_2(\mathit{setExists}_2\ (\wedge_2\ (\mathit{proj}_1 \circ (= A))\ (\mathit{proj}_2 \circ (= 3)))\ (\mathit{proj}_1 \circ (= B)))$,
$\Box_1\Box_2 \mathit{setExists}_2\ (\wedge_2\ (\mathit{proj}_1 \circ (= A))\ (\mathit{proj}_2 \circ (= 3)))\ (\mathit{proj}_1 \circ (= B))$,
$\wedge_2\ (\mathit{proj}_1 \circ (= A))\ (\mathit{proj}_2 \circ (= 3))$,
$\mathit{proj}_1 \circ (= A)$,
$\mathit{proj}_2 \circ (= 3)$,
$(= A)$,
$(= 3)$,
$\mathit{proj}_1 \circ (= B)$, and
$(= B)$.

Proposition 5.6. *An eligible subterm of a standard predicate is a standard predicate.*

Proof. This is a straightforward induction argument. □

Proposition 5.7. *Let p, q, and r be standard predicates, and q an eligible subterm of p such that $p[q/r]$ is a term. Then $p[q/r]$ is a standard predicate.*

Proof. The proof is by induction on the number of transformations in p. If the number of transformations in p is one, then the result is obvious. Suppose the result holds for standard predicates that have $< m$ transformations and p has m transformations. Let p have the form

$$\Box_1(f_1\ p_{1,1} \ldots p_{1,k_1}) \circ \cdots \circ \Box_n(f_n\ p_{n,1} \ldots p_{n,k_n}).$$

If q is a suffix of p, then the result follows since $p[q/r]$ is a term. If q is an eligible subterm of some p_{i,j_i}, then the induction hypothesis gives that $p_{i,j_i}[q/r]$ is a standard predicate. Since $p[q/r]$ is a term, it follows that $p[q/r]$ is a standard predicate. □

Definition 5.12. Let $\rightarrowtail$ be a predicate rewrite system and p a standard predicate. An eligible subterm r of p is a *redex* with respect to $\rightarrowtail$ if there exists a predicate rewrite $r \rightarrowtail b$ such that $p[r/b]$ is a standard predicate. In this case, r is said to be a redex *via* $r \rightarrowtail b$.

It follows from Proposition 5.7 that r is a redex iff $p[r/b]$ is a term. The phrase 'redex with respect to $\rightarrowtail$' is usually abbreviated to simply 'redex' with the predicate rewrite system understood.

Definition 5.13. Let $\rightarrowtail$ be a predicate rewrite system, and p and q standard predicates. Then q is obtained by a *predicate derivation step* from p using $\rightarrowtail$ if there is a redex r via $r \rightarrowtail b$ in p and $q = p[r/b]$. The redex r is called the *selected* redex.

Definition 5.14. A *predicate derivation* with respect to a predicate rewrite system $\rightarrowtail$ is a finite sequence $\langle p_0, p_1, \ldots, p_n \rangle$ of standard predicates such that p_i is obtained by a derivation step from p_{i-1} using $\rightarrowtail$, for $i = 1, \ldots, n$. The *length* of the predicate derivation is n. The standard predicate p_0 is called the *initial predicate* and the standard predicate p_n is called the *final predicate*.

Usually the initial predicate is *top*, the weakest predicate.

Example 5.11. Consider the following predicate rewrite system.

$$
\begin{aligned}
&\mathit{top} \rightarrowtail \boldsymbol{B}_i(\mathit{setExists}_1\ (\wedge_2\ \mathit{top}\ \mathit{top}))\\
&\mathit{top} \rightarrowtail \bullet\boldsymbol{B}_j\mathit{top}\\
&\mathit{top} \rightarrowtail \blacklozenge\boldsymbol{B}_j\mathit{top}\\
&\mathit{top} \rightarrowtail p\\
&\mathit{top} \rightarrowtail q\\
&\mathit{top} \rightarrowtail r.
\end{aligned}
$$

The following is a derivation in the predicate space defined by the rewrite system.

$$
\begin{aligned}
\mathit{top} \rightsquigarrow\ & \boldsymbol{B}_i(\mathit{setExists}_1\ (\wedge_2\ \mathit{top}\ \mathit{top})) \rightsquigarrow\ \boldsymbol{B}_i(\mathit{setExists}_1\ (\wedge_2\ \bullet\boldsymbol{B}_j\mathit{top}\ \ \mathit{top}))\\
&\rightsquigarrow\ \boldsymbol{B}_i(\mathit{setExists}_1\ (\wedge_2\ \bullet\boldsymbol{B}_jp\ \ \mathit{top}))\\
&\rightsquigarrow \cdots \rightsquigarrow\ \boldsymbol{B}_i(\mathit{setExists}_1\ (\wedge_2\ \bullet\boldsymbol{B}_jp\ \blacklozenge\boldsymbol{B}_jq)).
\end{aligned}
$$

The set $P_{\rightarrowtail}$ of predicates that can be generated from a predicate rewrite system $\rightarrowtail$ is called a *predicate language*. Given some predicate language, it remains to specify the *hypothesis language*, that is, the form of learned theories that employ predicates in the predicate language.

A language of particular importance for belief acquisition is defined next.

Definition 5.15. Let α be a type. A *basic predicate* for the type α is one of the form $(= t)$, for some $t \in \mathfrak{B}_\alpha$.

The set $\mathbf{B}_\alpha = \{(= t) \mid t \in \mathfrak{B}_\alpha\}$ of basic predicates for the type α is called the *basic language* for the type α.

Note that $\mathbf{B}_\alpha$ is the set of predicates that can be generated from the predicate rewrite system consisting of all rewrites of the form

$$\mathit{top} \rightarrowtail (= t),$$

where $t \in \mathfrak{B}_\alpha$.

6 Reasoning

This section contains a discussion of a reasoning system that combines computation and proof, in the sense that a computation can call for a proof to be performed and, conversely, a proof can call for a computation to be performed. Thus the definitions below of computation and proof are mutually recursive. To get started, I consider first the cases of pure computation, where no proof is involved, and pure proof, where no computation is involved, then later put the two together.

6.1 Computation

This subsection studies the case of (pure) computation.

Consider the problem of determining the meaning of a term t in the intended pointed interpretation. If a formal definition of the intended pointed interpretation is available, then

this problem can be solved (under some finiteness assumptions). However, I assume here that the intended pointed interpretation is not available, as is usually the case, so that the problem cannot be solved directly. Nevertheless, there is still a lot that can be done if the theory $\mathcal{T}$ of the application is available and enough of it is in equational form. Intuitively, if t can be 'simplified' sufficiently using $\mathcal{T}$, its meaning may become apparent even in the absence of detailed knowledge of the intended pointed interpretation. For example, if t can be simplified to a term containing only data constructors, then the meaning of t will be known since the data constructors have a fixed meaning in every interpretation.

More formally, the *computation problem* is as follows.

> Given a theory $\mathcal{T}$, a term t, and a sequence $\Box_{j_1} \cdots \Box_{j_r}$ of modalities, find a 'simpler' term t' such that $\Box_{j_1} \cdots \Box_{j_r} \forall (t = t')$ is a consequence of $\mathcal{T}$.

Thus t and t' have the same meaning in all worlds accessible from the point world in the intended pointed interpretation according to the modalities $\Box_{j_1} \cdots \Box_{j_r}$.

Here now are the details about a mechanism that addresses the computational problem by employing equational reasoning to rewrite terms to 'simpler' terms that have the same meaning.

Definition 6.1. Let $\mathcal{T} \equiv (\mathcal{G}, \mathcal{L})$ be a theory. A *computation of rank 0 using* $\Box_{j_1} \cdots \Box_{j_r}$ *with respect to* $\mathcal{T}$ is a sequence $\{t_i\}_{i=1}^n$ of terms such that the following conditions are satisfied.

1. For $i = 1, \ldots, n-1$, there is
   (a) a subterm s_i of t_i at occurrence o_i, where the modal path to o_i in t_i is $k_1 \ldots k_{m_i}$,
   (b) i. a formula $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_i}} \forall (u_i = v_i)$ in $\mathcal{L}$, or
   ii. a formula $\forall (u_i = v_i)$ in $\mathcal{G}$, and
   (c) a substitution θ_i that is admissible with respect to $u_i = v_i$

   such that $u_i\theta_i$ is α-equivalent to s_i and t_{i+1} is $t_i[s_i/v_i\theta_i]_{o_i}$.

The term t_1 is called the *goal* of the computation and t_n is called the *answer*.
Each subterm s_i is called a *redex*.
Each formula $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_i}} \forall (u_i = v_i)$ or $\forall (u_i = v_i)$ is called an *input equation*.
The formula $\Box_{j_1} \cdots \Box_{j_r} \forall (t_1 = t_n)$ is called the *result* of the computation.

Note that, by Proposition 2.9 and the remarks following Definition 2.23, $t_i[s_i/v_i\theta_i]_{o_i}$ is a term of the same type as t_i, for $i = 1, \ldots, n$.

A *selection rule* chooses the redex at each step of a computation. A common selection rule is the *leftmost* one which chooses the leftmost outermost subterm that satisfies the requirements of Definition 6.1. Note that it is straightforward to extend the definition of computation so that multiple redexes can be selected at each step. Then a common selection rule is the *parallel-outermost* one that selects all outermost subterms that each satisfy the requirements of Definition 6.1. Later, especially for partial evaluation, there will be a need for selection rules that do not select the leftmost redex.

Computations generally require use of definitions of $=$, the connectives and quantifiers, and some other basic functions. These definitions, which constitute what I call the standard equality theory, are discussed in detail in [Llo03], so here I just outline the main ideas.

Before starting on this, note that one difference between the setting in [Llo03] and the one here is that here modalities are allowed. Thus, in particular, here one must distinguish between local and global assumptions in theories. Given the intended meanings of equality, the connectives and the quantifiers, it is natural that their definitions would normally be taken to be *global* assumptions in the theories of applications.

Now I give a series of definitions of $=$, the connectives, the functions Σ and Π, and so on, that constitute the standard equality theory. All substitutions appearing in these definitions are assumed to be admissible. Furthermore, some of the equations that follow are schemas which contain syntactical variables that are denoted by boldface style. Syntactical variables are needed in situations where free variable capture is actually intended (and, therefore, the usual substitution mechanism is not appropriate). In general, a schema is intended to stand for the collection of terms that can be obtained from the schema by replacing its syntactical variables by terms, in such a way as to produce well-typed terms, of course. In a few places below, further *ad hoc* restrictions on the syntactical variables will be employed (such as requiring a syntactical variable to range over data constructors or object-level variables or rigid terms only). When using a schema in a computation or proof, a choice of terms to replace its syntactical variables is first made. This results in a formula that can then be handled by Definitions 6.1, 6.4, 6.6, and 6.7. Thus each of the following schemas is a compact way of specifying a (possibly infinite) collection of formulas.

The first definition is that of the equality predicate $=$. (Note that a % indicates that the remainder of the line is a comment.)

$= :\ a \to a \to \Omega$

$(\boldsymbol{C}\ x_1 \ldots x_n = \boldsymbol{C}\ y_1 \ldots y_n) = (x_1 = y_1) \wedge \cdots \wedge (x_n = y_n)$

% where $\boldsymbol{C}$ is a data constructor of arity n.

$(\boldsymbol{C}\ x_1 \ldots x_n = \boldsymbol{D}\ y_1 \ldots y_m) = \bot$

% where $\boldsymbol{C}$ is a data constructor of arity n,

% $\boldsymbol{D}$ is a data constructor of arity m, and $\boldsymbol{C} \neq \boldsymbol{D}$.

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

% where $n = 2, 3, \ldots$.

$(\lambda x.\boldsymbol{u} = \lambda y.\boldsymbol{v}) = (\mathit{less}\ \lambda x.\boldsymbol{u}\ \lambda y.\boldsymbol{v}) \wedge (\mathit{less}\ \lambda y.\boldsymbol{v}\ \lambda x.\boldsymbol{u})$

The first two schemas in the above definition simply capture the intended meaning of data constructors, while the third captures an important property of tuples. (Note that for the first schema, if $n = 0$, then the body is $\top$.)

However, the fourth schema is more subtle. In the proof-theoretic component below of the reasoning system, as is common in formulations of higher-order logics, one can adopt the axiom of extensionality:

$$\forall f.\forall g.((f = g) = \forall x.((f\ x) = (g\ x))).$$

However, this axiom is not used in the computational component of the reasoning system for the reason that it is not computationally useful: showing that $\forall x.((f\ x) = (g\ x))$ is not generally possible as there can be infinitely many values of x to consider. Instead, a special case of the axiom of extensionality is used in the standard equality theory. Its purpose is to provide a method of checking whether certain abstractions representing finite sets, finite

multisets and similar data types are equal. In such cases, one can check for equality in a finite amount of time. The fourth schema relies on the two following definitions.

$less\ :\ (a \to b) \to (a \to b) \to \Omega$

$less\ \lambda x.\boldsymbol{d}\ z = \top$

% where $\boldsymbol{d}$ is a default term.

$less\ (\lambda x.if\ \boldsymbol{u}\ then\ v\ else\ \boldsymbol{w})\ z = (\forall x.(\boldsymbol{u} \longrightarrow v = (z\ x))) \wedge (less\ (remove\ \{x \mid \boldsymbol{u}\}\ \lambda x.\boldsymbol{w})\ z)$

$remove\ :\ (a \to \Omega) \to (a \to b) \to (a \to b)$

$remove\ s\ \lambda x.\boldsymbol{d} = \lambda x.\boldsymbol{d}$

% where $\boldsymbol{d}$ is a default term.

$remove\ s\ \lambda x.if\ \boldsymbol{u}\ then\ v\ else\ \boldsymbol{w} = \lambda x.if\ \boldsymbol{u} \wedge (x \notin s)\ then\ v\ else\ ((remove\ s\ \lambda x.\boldsymbol{w})\ x)$

The intended meaning of *less* is best given by an illustration. Consider the multisets m and n. Then *less* m n is true iff each item in the support of m is also in the support of n and has the same multiplicity there. For sets, *less* is simply the subset relation. If s is a set and m a multiset, then *remove* s m returns the multiset obtained from m by removing all the items from its support that are in s.

The next definition is the obvious one for disequality.

$\neq :\ a \to a \to \Omega$

$x \neq y = \neg\ (x = y)$

The following definitions are for the connectives $\wedge$, $\vee$, and $\neg$.

$\wedge\ :\ \Omega \to \Omega \to \Omega$

$\top \wedge x = x$
$x \wedge \top = x$
$\bot \wedge x = \bot$
$x \wedge \bot = \bot$
$(x \vee y) \wedge z = (x \wedge z) \vee (y \wedge z)$
$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
$(if\ u\ then\ v\ else\ w) \wedge t = if\ u \wedge t\ then\ v\ else\ w \wedge t$
$t \wedge (if\ u\ then\ v\ else\ w) = if\ t \wedge u\ then\ v\ else\ t \wedge w$
$u \wedge (\exists x_1.\cdots\exists x_n.\boldsymbol{v}) = \exists x_1.\cdots\exists x_n.(u \wedge \boldsymbol{v})$
$(\exists x_1.\cdots\exists x_n.\boldsymbol{u}) \wedge v = \exists x_1.\cdots\exists x_n.(\boldsymbol{u} \wedge v)$
$\boldsymbol{u} \wedge (\boldsymbol{x} = \boldsymbol{t}) \wedge \boldsymbol{v} = \boldsymbol{u}\{\boldsymbol{x}/\boldsymbol{t}\} \wedge (\boldsymbol{x} = \boldsymbol{t}) \wedge \boldsymbol{v}\{\boldsymbol{x}/\boldsymbol{t}\}$

% where $\boldsymbol{x}$ is a variable free in $\boldsymbol{u}$ or $\boldsymbol{v}$ or both, but not free in $\boldsymbol{t}$, and $\boldsymbol{t}$ is not a variable.

$\boldsymbol{u} \wedge (\boldsymbol{t} = \boldsymbol{x}) \wedge \boldsymbol{v} = \boldsymbol{u}\{\boldsymbol{x}/\boldsymbol{t}\} \wedge (\boldsymbol{x} = \boldsymbol{t}) \wedge \boldsymbol{v}\{\boldsymbol{x}/\boldsymbol{t}\}$

% where $\boldsymbol{x}$ is a variable free in $\boldsymbol{u}$ or $\boldsymbol{v}$ or both, but not free in $\boldsymbol{t}$, and $\boldsymbol{t}$ is not a variable.

$\vee\ :\ \Omega \to \Omega \to \Omega$

$\top \vee x = \top$
$x \vee \top = \top$
$\bot \vee x = x$
$x \vee \bot = x$
$(\mathit{if}\ u\ \mathit{then}\ \top\ \mathit{else}\ w) \vee t = \mathit{if}\ u\ \mathit{then}\ \top\ \mathit{else}\ w \vee t$
$(\mathit{if}\ u\ \mathit{then}\ \bot\ \mathit{else}\ w) \vee t = (\neg\, u \wedge w) \vee t$
$t \vee (\mathit{if}\ u\ \mathit{then}\ \top\ \mathit{else}\ w) = \mathit{if}\ u\ \mathit{then}\ \top\ \mathit{else}\ t \vee w$
$t \vee (\mathit{if}\ u\ \mathit{then}\ \bot\ \mathit{else}\ w) = t \vee (\neg\, u \wedge w)$

$\neg\ :\ \Omega \to \Omega$

$\neg \bot = \top$
$\neg \top = \bot$
$\neg\, (\neg\, x) = x$
$\neg\, (x \wedge y) = (\neg\, x) \vee (\neg\, y)$
$\neg\, (x \vee y) = (\neg\, x) \wedge (\neg\, y)$
$\neg\, (\mathit{if}\ u\ \mathit{then}\ v\ \mathit{else}\ w) = \mathit{if}\ u\ \mathit{then} \neg\, v\ \mathit{else} \neg\, w$

These definitions are straightforward, except perhaps for the last three schemas in the definition of $\wedge$. The second and third last schemas allow the scope of existential quantifiers to be extended provided it does not result in free variable capture. (Recall the convention restricting the possible term replacements for syntactical variables.)

The last schema allows the elimination of some occurrences of a free variable ($\boldsymbol{x}$, in this case), thus simplifying an expression. A similar schema allowing this kind of simplification also occurs in the definition of Σ below. However, a few words about the expression $\boldsymbol{u} \wedge (\boldsymbol{x} = \boldsymbol{t}) \wedge \boldsymbol{v}$ are necessary. The intended meaning of this expression is that it is a term such that $(\boldsymbol{x} = \boldsymbol{t})$ is 'embedded conjunctively' inside it. More formally, a term t is embedded conjunctively in t and, if t is embedded conjunctively in r (or s), then t is embedded conjunctively in $r \wedge s$. So, for example, $x = s$ is embedded conjunctively in the term $((p \wedge q) \wedge r) \wedge ((x = s) \wedge (t \wedge u))$.

Next come the definitions of Σ and Π.

$\Sigma\ :\ (a \to \Omega) \to \Omega$

$\exists x_1. \cdots \exists x_n. \top = \top$
$\exists x_1. \cdots \exists x_n. \bot = \bot$
$\exists x_1. \cdots \exists x_n. (\boldsymbol{x} \wedge (x_i = \boldsymbol{u}) \wedge \boldsymbol{y}) = \exists x_1. \cdots \exists x_{i-1}. \exists x_{i+1}. \cdots \exists x_n. (\boldsymbol{x}\{x_i/\boldsymbol{u}\} \wedge \boldsymbol{y}\{x_i/\boldsymbol{u}\})$
% where x_i is not free in $\boldsymbol{u}$.
$\exists x_1. \cdots \exists x_n. (\boldsymbol{x} \wedge (\boldsymbol{u} = x_i) \wedge \boldsymbol{y}) = \exists x_1. \cdots \exists x_{i-1}. \exists x_{i+1}. \cdots \exists x_n. (\boldsymbol{x}\{x_i/\boldsymbol{u}\} \wedge \boldsymbol{y}\{x_i/\boldsymbol{u}\})$
% where x_i is not free in $\boldsymbol{u}$.
$\exists x_1. \cdots \exists x_n. (\boldsymbol{u} \vee \boldsymbol{v}) = (\exists x_1. \cdots \exists x_n. \boldsymbol{u}) \vee (\exists x_1. \cdots \exists x_n. \boldsymbol{v})$
$\exists x_1. \cdots \exists x_n. (\mathit{if}\ \boldsymbol{u}\ \mathit{then}\ \top\ \mathit{else}\ \boldsymbol{w}) = \mathit{if}\ \exists x_1. \cdots \exists x_n. \boldsymbol{u}\ \mathit{then}\ \top\ \mathit{else}\ \exists x_1. \cdots \exists x_n. \boldsymbol{w}$
$\exists x_1. \cdots \exists x_n. (\mathit{if}\ \boldsymbol{u}\ \mathit{then}\ \bot\ \mathit{else}\ \boldsymbol{w}) = \exists x_1. \cdots \exists x_n. (\neg\, \boldsymbol{u} \wedge \boldsymbol{w})$
$\exists x_1. \cdots \exists x_n. (\mathit{if}\ \boldsymbol{u}\ \mathit{then}\ \boldsymbol{v}\ \mathit{else}\ \boldsymbol{w}) = \mathit{if}\ \exists x_1. \cdots \exists x_n. (\boldsymbol{u} \wedge \boldsymbol{v})\ \mathit{then}\ \top\ \mathit{else}\ \exists x_1. \cdots \exists x_n. (\neg \boldsymbol{u} \wedge \boldsymbol{w})$

% where $\boldsymbol{v}$ is neither $\top$ nor $\bot$.

$\Pi \ : \ (a \to \Omega) \to \Omega$

$\forall x_1.\cdots\forall x_n.(\bot \longrightarrow \boldsymbol{u}) = \top$

$\forall x_1.\cdots\forall x_n.(\boldsymbol{x} \wedge (x_i = \boldsymbol{u}) \wedge \boldsymbol{y} \longrightarrow \boldsymbol{v}) =$
$\forall x_1.\cdots\forall x_{i-1}.\forall x_{i+1}.\cdots\forall x_n.(\boldsymbol{x}\{x_i/\boldsymbol{u}\} \wedge \boldsymbol{y}\{x_i/\boldsymbol{u}\} \longrightarrow \boldsymbol{v}\{x_i/\boldsymbol{u}\})$

% where x_i is not free in $\boldsymbol{u}$.

$\forall x_1.\cdots\forall x_n.(\boldsymbol{x} \wedge (\boldsymbol{u} = x_i) \wedge \boldsymbol{y} \longrightarrow \boldsymbol{v}) =$
$\forall x_1.\cdots\forall x_{i-1}.\forall x_{i+1}.\cdots\forall x_n.(\boldsymbol{x}\{x_i/\boldsymbol{u}\} \wedge \boldsymbol{y}\{x_i/\boldsymbol{u}\} \longrightarrow \boldsymbol{v}\{x_i/\boldsymbol{u}\})$

% where x_i is not free in $\boldsymbol{u}$.

$\forall x_1.\cdots\forall x_n.(\boldsymbol{u} \vee \boldsymbol{v} \longrightarrow \boldsymbol{t}) = (\forall x_1.\cdots\forall x_n.(\boldsymbol{u} \longrightarrow \boldsymbol{t})) \wedge (\forall x_1.\cdots\forall x_n.(\boldsymbol{v} \longrightarrow \boldsymbol{t}))$

$\forall x_1.\cdots\forall x_n.((\mathit{if}\ \boldsymbol{u}\ \mathit{then}\ \top\ \mathit{else}\ \boldsymbol{v}) \longrightarrow \boldsymbol{t}) =$
$(\forall x_1.\cdots\forall x_n.(\boldsymbol{u} \longrightarrow \boldsymbol{t})) \wedge (\forall x_1.\cdots\forall x_n.(\boldsymbol{v} \longrightarrow \boldsymbol{t}))$

$\forall x_1.\cdots\forall x_n.((\mathit{if}\ \boldsymbol{u}\ \mathit{then}\ \bot\ \mathit{else}\ \boldsymbol{v}) \longrightarrow \boldsymbol{t}) = \forall x_1.\cdots\forall x_n.(\neg\, \boldsymbol{u} \wedge \boldsymbol{v} \longrightarrow \boldsymbol{t})$

For the definition of Σ, recall that $\exists x.t$ stands for $(\Sigma\ \lambda x.t)$. Thus, for example, $\exists x.\top$ stands for $(\Sigma\ \lambda x.\top)$.

The definition for the *if_then_else* function follows.

$\mathit{if\_then\_else} \ : \ \Omega \times a \times a \to a$

$\mathit{if}\ \top\ \mathit{then}\ u\ \mathit{else}\ v = u$
$\mathit{if}\ \bot\ \mathit{then}\ u\ \mathit{else}\ v = v$

The next two assumptions involve function application and the *if_then_else* function.

$(\boldsymbol{w}\ \ \mathit{if}\ \boldsymbol{x} = \boldsymbol{t}\ \mathit{then}\ u\ \mathit{else}\ v) = \mathit{if}\ \boldsymbol{x} = \boldsymbol{t}\ \mathit{then}\ (\boldsymbol{w}\{\boldsymbol{x}/\boldsymbol{t}\}\ u)\ \mathit{else}\ (\boldsymbol{w}\ v)$

% where $\boldsymbol{x}$ is a variable.

$(\mathit{if}\ \boldsymbol{x} = \boldsymbol{t}\ \mathit{then}\ u\ \mathit{else}\ v\ \ \boldsymbol{w}) = \mathit{if}\ \boldsymbol{x} = \boldsymbol{t}\ \mathit{then}\ (u\ \boldsymbol{w}\{\boldsymbol{x}/\boldsymbol{t}\}))\ \mathit{else}\ (v\ \boldsymbol{w})$

% where $\boldsymbol{x}$ is a variable.

The following assumption provides β-reduction.

$(\lambda x.\boldsymbol{u}\ t) = \boldsymbol{u}\{x/t\}$

Finally, there are three schemas that involve modalities.

$\Box_i \boldsymbol{t} = \boldsymbol{t}$

% where $\boldsymbol{t}$ is a rigid term and $i \in \{1, \ldots, m\}$.

$(\Box_i \mathbf{s}\ \mathbf{t}) = \Box_i(\mathbf{s}\ \mathbf{t})$

% where $\mathbf{t}$ is a rigid term and $i \in \{1, \ldots, m\}$.

$\Box_i \lambda x.\mathbf{t} = \lambda x.\Box_i \mathbf{t}$

% where $i \in \{1, \ldots, m\}$.

Definition 6.2. The *standard equality theory* is the theory consisting of the preceding definitions of equality and subsidiary functions, the connectives, the quantifiers, the *if_then_else* function, β-reduction, and the three modal schemas as global assumptions (and no local assumptions).

The soundness of computation is given by the next result.

Proposition 6.1. *Let $\mathcal{T}$ be a theory. Then the result of a computation of rank 0 using $\Box_{j_1} \cdots \Box_{j_r}$ with respect to $\mathcal{T}$ is a consequence of $\mathcal{T}$.*

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

Suppose now that the result holds for computations of length n. Consider a computation $\{t_i\}_{i=1}^{n+1}$ using $\Box_{j_1} \cdots \Box_{j_r}$ of length $n+1$ for which, at the nth step, the input equation is either the local assumption $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_n}} \forall(u_n = v_n)$ or the global assumption $\forall(u_n = v_n)$, the admissible substitution is θ_n, and the redex is s_n at occurrence o_n with modal path $k_1 \ldots k_{m_n}$ to o_n.

Let $\mathcal{T}$ be $(\mathcal{G}, \mathcal{L})$, I an interpretation, and w a world in I. Suppose that each member of $\mathcal{G}$ is valid in I and that each member of $\mathcal{L}$ is valid at w in I. By the induction hypothesis, $\Box_{j_1} \cdots \Box_{j_r} \forall(t_1 = t_n)$ is valid at w in I, that is, $\mathcal{V}(\Box_{j_1} \cdots \Box_{j_r} \forall(t_1 = t_n), I, w, \nu) = \top$, for each variable assignment ν. Thus, by Proposition 3.7, $\mathcal{V}(\forall(t_1 = t_n), I, w', \nu) = \top$, for each variable assignment ν and each w' such that $w\ R_{j_1} \circ \cdots \circ R_{j_r}\ w'$. Hence, by Part 3 of Proposition 3.5, $\mathcal{V}(t_1 = t_n, I, w', \nu) = \top$, for each variable assignment ν and each w' such that $w\ R_{j_1} \circ \cdots \circ R_{j_r}\ w'$.

By Proposition 3.40, for either case of input equation, $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_n}} \forall(u_n = v_n)$ is a consequence of $\mathcal{T}$. Thus, by Proposition 3.41, $\forall(\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_n}} (u_n = v_n))$ is a consequence of $\mathcal{T}$ and so, by Proposition 3.42, $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_n}} (u_n = v_n)$ is a consequence of $\mathcal{T}$. Hence, by Proposition 3.43, $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_n}} (u_n\theta_n = v_n\theta_n)$ is a consequence of $\mathcal{T}$. Thus $\mathcal{V}(\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_n}} (u_n\theta_n = v_n\theta_n), I, w, \nu) = \top$, for each variable assignment ν. Consequently, by Proposition 3.7, $\mathcal{V}(\Box_{k_1} \cdots \Box_{k_{m_n}} (u_n\theta_n = v_n\theta_n), I, w', \nu) = \top$, for each variable assignment ν and each w' such that $w\ R_{j_1} \circ \cdots \circ R_{j_r}\ w'$.

Now t_{n+1} is $t_n[s_n/v_n\theta_n]_{o_n}$. Hence, by Proposition 3.33, $\mathcal{V}(t_n = t_{n+1}, I, w', \nu) = \top$, for each variable assignment ν and each w' such that $w\ R_{j_1} \circ \cdots \circ R_{j_r}\ w'$. By Proposition 3.10, $\mathcal{V}(t_1 = t_{n+1}, I, w', \nu) = \top$, for each variable assignment ν and each w' such that $w\ R_{j_1} \circ \cdots \circ R_{j_r}\ w'$. By Part 3 of Proposition 3.5, $\mathcal{V}(\forall(t_1 = t_{n+1}), I, w', \nu) = \top$, for each variable assignment ν and each w' such that $w\ R_{j_1} \circ \cdots \circ R_{j_r}\ w'$. By Proposition 3.7, $\mathcal{V}(\Box_{j_1} \cdots \Box_{j_r} \forall(t_1 = t_{n+1}), I, w, \nu) = \top$, for each variable assignment ν. Thus $\Box_{j_1} \cdots \Box_{j_r} \forall(t_1 = t_{n+1})$ is a consequence of $\mathcal{T}$. This completes the induction step. □

The next proposition provides a useful result about computing with basic terms.

Proposition 6.2. *Let s and t be basic terms of the same type, $\mathcal{E}$ the standard equality theory, and $\Box_{j_1} \cdots \Box_{j_r}$ any sequence of modalities. Then the following hold.*

1. *If $s \xleftrightarrow{*}_{\alpha} t$, then there is a computation of rank 0 using $\Box_{j_1} \cdots \Box_{j_r}$ with respect to $\mathcal{E}$ for the goal $s = t$ that has the result $\Box_{j_1} \cdots \Box_{j_r}((s = t) = \top)$.*

2. *If $s \not\xleftrightarrow{*}_{\alpha} t$, then there is a computation of rank 0 using $\Box_{j_1} \cdots \Box_{j_r}$ with respect to $\mathcal{E}$ for the goal $s = t$ that has the result $\Box_{j_1} \cdots \Box_{j_r}((s = t) = \bot)$.*

Proof. The proof of both parts together is by induction on the structure of s [Llo03, Proposition 3.5.1].

Suppose that s of type $T\ \alpha_1 \dots \alpha_k$ has the form $C\ s_1 \dots s_n$. If $s \stackrel{*}{\longleftrightarrow}_\alpha t$, then t has the form $C\ t_1 \dots t_n$ and $s_i \stackrel{*}{\longleftrightarrow}_\alpha t_i$, for $i = 1, \dots, n$. By the induction hypothesis, there is a computation using $\Box_{j_1} \cdots \Box_{j_r}$ with the goal $s_i = t_i$ and the answer $\top$, for $i = 1, \dots, n$. Then, using the global assumption

$$(\boldsymbol{C}\ x_1 \dots x_n = \boldsymbol{C}\ y_1 \dots y_n) = (x_1 = y_1) \wedge \cdots \wedge (x_n = y_n)$$

and other global assumptions in $\mathcal{E}$, a computation can be constructed using $\Box_{j_1} \cdots \Box_{j_r}$ with the goal $s = t$ and the answer $\top$, as required.

If $s \not\stackrel{*}{\longleftrightarrow}_\alpha t$, then there are two cases to consider. Suppose first that t has the form $D\ t_1 \dots t_m$, where $C \neq D$. Then the global assumption

$$(\boldsymbol{C}\ x_1 \dots x_n = \boldsymbol{D}\ y_1 \dots y_m) = \bot$$

can be used to construct a computation with the goal $s = t$ and the answer $\bot$. In the second case, t has the form $C\ t_1 \dots t_n$ and $s_j \not\stackrel{*}{\longleftrightarrow}_\alpha t_j$, for some $j \in \{1, \dots, n\}$. By the induction hypothesis, there is a computation using $\Box_{j_1} \cdots \Box_{j_r}$ with the goal $s_j = t_j$ and the answer $\bot$. Then, using the global assumption

$$(\boldsymbol{C}\ x_1 \dots x_n = \boldsymbol{C}\ y_1 \dots y_n) = (x_1 = y_1) \wedge \cdots \wedge (x_n = y_n)$$

and other global assumptions in $\mathcal{E}$, a computation can be constructed using $\Box_{j_1} \cdots \Box_{j_r}$ with the goal $s = t$ and the answer $\bot$, as required. This completes the case for basic structures.

Suppose next that s of type $\gamma \to \eta$ has the form

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

where $t_1, \dots, t_n \in \mathfrak{B}_\gamma$, $s_1, \dots, s_n \in \mathfrak{B}_\eta$, $t_1 < \cdots < t_n$, $s_i \notin \mathfrak{D}_\eta$, for $1 \leq i \leq n$ ($n \in \mathbb{N}$), and $s_0 \in \mathfrak{D}_\eta$. There are two cases to consider.

The first case to consider is when $s \stackrel{*}{\longleftrightarrow}_\alpha t$. In this case, t must have the form

$$\lambda y.\mathit{if}\ y = t'_1\ \mathit{then}\ s'_1\ \mathit{else}\ \dots\ \mathit{if}\ y = t'_n\ \mathit{then}\ s'_n\ \mathit{else}\ s_0,$$

where $t'_1, \dots, t'_n \in \mathfrak{B}_\gamma$, $s'_1, \dots, s'_n \in \mathfrak{B}_\eta$, $t'_1 < \cdots < t'_n$, $s_i \notin \mathfrak{D}_\eta$, for $1 \leq i \leq n$ ($n \in \mathbb{N}$), $s_0 \in \mathfrak{D}_\eta$, and $s_i \stackrel{*}{\longleftrightarrow}_\alpha s'_i$ and $t_i \stackrel{*}{\longleftrightarrow}_\alpha t'_i$, for $i = 1, \dots, n$. By the induction hypothesis, there is a computation using using $\Box_{j_1} \cdots \Box_{j_r}$ with respect to $\mathcal{E}$ for the goal $t_i = t'_i$ that has the result $\Box_{j_1} \cdots \Box_{j_r}((t_i = t'_i) = \top)$ and there is a computation using $\Box_{j_1} \cdots \Box_{j_r}$ with respect to $\mathcal{E}$ for the goal $s_i = s'_i$ that has the result $\Box_{j_1} \cdots \Box_{j_r}((s_i = s'_i) = \top)$, for $i = 1, \dots, n$. A computation is now constructed for the goal $s = t$. In this computation and the later ones in the proof, I use several times the easily established fact that, if s and t are basic terms and $s < t$, then $s \not\stackrel{*}{\longleftrightarrow}_\alpha t$. Let s_{rest} denote

$$\lambda x.\mathit{if}\ x = t_2\ \mathit{then}\ s_2\ \mathit{else}\ \dots\ \mathit{if}\ x = t_n\ \mathit{then}\ s_n\ \mathit{else}\ s_0.$$

An outline of this computation is as follows. (The redexes are underlined.)

$$
\begin{array}{l}
\underline{s = t} \\
\underline{(less\ s\ t)} \wedge (less\ t\ s) \\
(\underline{(\forall x.((x = t_1) \longrightarrow s_1 = (t\ x)))} \wedge (less\ (remove\ \{x \mid (x = t_1)\}\ s_{rest})\ t)) \wedge (less\ t\ s) \\
((s_1 = \underline{(t\ t_1)}) \wedge (less\ (remove\ \{x \mid (x = t_1)\}\ s_{rest})\ t)) \wedge (less\ t\ s) \\
((s_1 = (if\ \underline{t_1 = t'_1}\ then\ s'_1\ else\ \ldots\ if\ t_1 = t'_n\ then\ s'_n\ else\ s_0)) \wedge \\
\qquad\qquad (less\ (remove\ \{x \mid (x = t_1)\}\ s_{rest})\ t)) \wedge (less\ t\ s) \\
\quad \vdots \\
((s_1 = \underline{(if\ \top\ then\ s'_1\ else\ \ldots\ if\ t_1 = t'_n\ then\ s'_n\ else\ s_0)}) \wedge \\
\qquad\qquad (less\ (remove\ \{x \mid (x = t_1)\}\ s_{rest})\ t)) \wedge (less\ t\ s) \\
(\underline{(s_1 = s'_1)} \wedge (less\ (remove\ \{x \mid (x = t_1)\}\ s_{rest})\ t)) \wedge (less\ t\ s) \\
\quad \vdots \\
\underline{(\top \wedge (less\ (remove\ \{x \mid (x = t_1)\}\ s_{rest})\ t))} \wedge (less\ t\ s) \\
(less\ \underline{(remove\ \{x \mid (x = t_1)\}\ s_{rest})}\ t) \wedge (less\ t\ s) \\
\quad \vdots \\
\underline{(less\ s_{rest}\ t)} \wedge (less\ t\ s) \\
\quad \vdots \\
\underline{\top \wedge (less\ t\ s)} \\
\underline{less\ t\ s} \\
\quad \vdots \\
\top.
\end{array}
$$

The case when $n = 0$, that is, when s and t have the form $\lambda x.s_0$, is much easier.

The second case to consider for basic abstractions is when $s \not\xleftrightarrow{*}_\alpha t$. In this case, there exists $j \in \{1, \ldots, n\}$ such that $t_i \xleftrightarrow{*}_\alpha t'_i$ and $s_i \xleftrightarrow{*}_\alpha s'_i$, for $i < j$, and either $t_j \not\xleftrightarrow{*}_\alpha t'_j$ or $s_j \not\xleftrightarrow{*}_\alpha s'_j$. Alternatively, one of s and t is a strict "prefix" (up to α-equivalence) of the other; the details of this case are omitted. Suppose that $t_j \xleftrightarrow{*}_\alpha t'_j$, but $s_j \not\xleftrightarrow{*}_\alpha s'_j$. Let s_{rest_j} denote

$$\lambda x.if\ x = t_j\ then\ s_j\ else\ \ldots\ if\ x = t_n\ then\ s_n\ else\ s_0.$$

An outline of a computation for the goal $s = t$ in this case is as follows.

$$
\begin{array}{l}
\underline{s = t} \\
\underline{(\mathit{less}\ s\ t)} \wedge (\mathit{less}\ t\ s) \\
\qquad \vdots \\
\underline{(\mathit{less}\ s_{rest_j}\ t)} \wedge (\mathit{less}\ t\ s) \\
(\underline{(\forall x.((x = t_j) \longrightarrow s_j = (t\ x)))} \wedge (\mathit{less}\ (\mathit{remove}\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (\mathit{less}\ t\ s) \\
((s_j = \underline{(t\ t_j)}) \wedge (\mathit{less}\ (\mathit{remove}\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (\mathit{less}\ t\ s) \\
((s_j = (\mathit{if}\ \underline{t_j = t'_1}\ \mathit{then}\ s'_1\ \mathit{else}\ \ldots\ \mathit{if}\ t_j = t'_n\ \mathit{then}\ s'_n\ \mathit{else}\ s_0)) \wedge \\
\qquad\qquad (\mathit{less}\ (\mathit{remove}\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (\mathit{less}\ t\ s) \\
\qquad \vdots \\
((s_j = (\underline{\mathit{if}\ \bot\ \mathit{then}\ s'_1\ \mathit{else}\ \ldots\ \mathit{if}\ t_j = t'_n\ \mathit{then}\ s'_n\ \mathit{else}\ s_0})) \wedge \\
\qquad\qquad (\mathit{less}\ (\mathit{remove}\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (\mathit{less}\ t\ s) \\
((s_j = (\mathit{if}\ \underline{t_j = t'_2}\ \mathit{then}\ s'_2\ \mathit{else}\ \ldots\ \mathit{if}\ t_j = t'_n\ \mathit{then}\ s'_n\ \mathit{else}\ s_0)) \wedge \\
\qquad\qquad (\mathit{less}\ (\mathit{remove}\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (\mathit{less}\ t\ s) \\
\qquad \vdots \\
((s_j = (\mathit{if}\ \underline{t_j = t'_j}\ \mathit{then}\ s'_j\ \mathit{else}\ \ldots\ \mathit{if}\ t_j = t'_n\ \mathit{then}\ s'_n\ \mathit{else}\ s_0)) \wedge \\
\qquad\qquad (\mathit{less}\ (\mathit{remove}\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (\mathit{less}\ t\ s) \\
\qquad \vdots \\
((s_j = (\underline{\mathit{if}\ \top\ \mathit{then}\ s'_j\ \mathit{else}\ \ldots\ \mathit{if}\ t_j = t'_n\ \mathit{then}\ s'_n\ \mathit{else}\ s_0})) \wedge \\
\qquad\qquad (\mathit{less}\ (\mathit{remove}\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (\mathit{less}\ t\ s) \\
(\underline{(s_j = s'_j)} \wedge (\mathit{less}\ (\mathit{remove}\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (\mathit{less}\ t\ s) \\
\qquad \vdots \\
\underline{(\bot \wedge (\mathit{less}\ (\mathit{remove}\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t))} \wedge (\mathit{less}\ t\ s) \\
\underline{\bot \wedge (\mathit{less}\ t\ s)} \\
\bot.
\end{array}
$$

The remaining case is when $t_j \not\xrightarrow{*}_{\alpha} t'_j$. Suppose that $t_j < t'_j$. (The case when $t'_j < t_j$ is similar except that $(\mathit{less}\ t\ s)$ instead of $(\mathit{less}\ s\ t)$ is shown to evaluate to $\bot$.) An outline of a

computation for the goal $s = t$ in this case is as follows.

$$
\begin{array}{l}
\underline{s = t} \\
\underline{(less\ s\ t)} \wedge (less\ t\ s) \\
\quad \vdots \\
\underline{(less\ s_{rest_j}\ t)} \wedge (less\ t\ s) \\
(\underline{(\forall x.((x = t_j) \longrightarrow s_j = (t\ x)))} \wedge (less\ (remove\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (less\ t\ s) \\
((s_j = \underline{(t\ t_j)}) \wedge (less\ (remove\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (less\ t\ s) \\
((s_j = (if\ \underline{t_j = t'_1}\ then\ s'_1\ else\ \ldots\ if\ t_j = t'_n\ then\ s'_n\ else\ s_0)) \wedge \\
\qquad\qquad (less\ (remove\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (less\ t\ s) \\
\quad \vdots \\
((s_j = (\underline{if\ \bot\ then\ s'_1\ else\ \ldots\ if\ t_j = t'_n\ then\ s'_n\ else\ s_0})) \wedge \\
\qquad\qquad (less\ (remove\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (less\ t\ s) \\
\quad \vdots \\
((s_j = (if\ \underline{t_j = t'_j}\ then\ s'_j\ else\ \ldots\ if\ t_j = t'_n\ then\ s'_n\ else\ s_0)) \wedge \\
\qquad\qquad (less\ (remove\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (less\ t\ s) \\
\quad \vdots \\
((s_j = (\underline{if\ \bot\ then\ s'_j\ else\ \ldots\ if\ t_j = t'_n\ then\ s'_n\ else\ s_0})) \wedge \\
\qquad\qquad (less\ (remove\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (less\ t\ s) \\
\quad \vdots \\
(\underline{(s_j = s_0)} \wedge (less\ (remove\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (less\ t\ s) \\
\quad \vdots \\
\underline{(\bot \wedge (less\ (remove\ \{x \mid (x = t_j)\}\ s_{rest_j})\ t)) \wedge (less\ t\ s)} \\
\underline{\bot \wedge (less\ t\ s)} \\
\bot.
\end{array}
$$

This completes the case of basic abstractions.

Suppose finally that s of type $\sigma \equiv (\sigma_1, \ldots, \sigma_n)$ has the form $(s_1, \ldots, s_n)$. Thus t has the form $(t_1, \ldots, t_n)$. If $s \overset{*}{\longleftrightarrow}_\alpha t$, then $s_i \overset{*}{\longleftrightarrow}_\alpha t_i$, for $i = 1, \ldots, n$. By the induction hypothesis, there is a computation using $\Box_{j_1} \cdots \Box_{j_r}$ with the goal $s_i = t_i$ and the answer $\top$, for $i = 1, \ldots, n$. Then, using the global assumption

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

and other global assumptions in $\mathcal{E}$, a computation can be constructed using $\Box_{j_1} \cdots \Box_{j_r}$ with the goal $s = t$ and the answer $\top$, as required. If $s \not\overset{*}{\longleftrightarrow}_\alpha t$, then $s_j \not\overset{*}{\longleftrightarrow}_\alpha t_j$, for some $j \in \{1, \ldots, n\}$. By the induction hypothesis, there is a computation using $\Box_{j_1} \cdots \Box_{j_r}$ with the goal $s_j = t_j$ and the answer $\bot$. A similar argument to the previous case now shows that

there is a computation using $\Box_{j_1} \cdots \Box_{j_r}$ with the goal $s = t$ and the answer $\bot$, as required. This completes the case of basic tuples. $\Box$

Here are four examples to illustrate computation.

Example 6.1. Consider again the functions *sometime* and *upto_sometime* that were introduced in Section 3.8 and that have the following definitions.

$$sometime : \Omega \times Nat \to \Omega$$
$$(sometime\ (\boldsymbol{\varphi}, n)) = if\ n = 0\ then\ \boldsymbol{\varphi}\ else\ \bigcirc(sometime\ (\boldsymbol{\varphi}, n-1))$$

$$upto\_sometime : \Omega \times Nat \to \Omega$$
$$(upto\_sometime\ (\boldsymbol{\varphi}, n)) =$$
$$if\ n = 0\ then\ \boldsymbol{\varphi}\ else\ ((sometime\ (\boldsymbol{\varphi}, n)) \wedge (upto\_sometime\ (\boldsymbol{\varphi}, n-1))).$$

These schemas are taken to be global assumptions of the theory.

Now let φ be some biterm. Then Figure 3 gives a computation of $(upto\_sometime\ (\varphi, 2))$. The computation uses the leftmost selection rule and has the redexes underlined. It follows from the computation that

$$(upto\_sometime\ (\varphi, 2)) = \bigcirc^{\leq 2}\varphi$$

is a consequence of the theory.

Example 6.2. Consider the following polymorphic definitions of the functions *append*, *permute*, *delete*, and *sorted* which have been written in the relational style of logic programming.

$$append\ :\ List\ a \times List\ a \times List\ a \to \Omega$$
$$(append\ (u, v, w)) =$$
$$((u = []) \wedge (v = w)) \vee \exists r.\exists x.\exists y.((u = r \sharp x) \wedge (w = r \sharp y) \wedge (append\ (x, v, y)))$$

$$permute\ :\ List\ a \times List\ a \to \Omega$$
$$(permute\ ([], x)) = x = []$$
$$(permute\ (x \sharp y, w)) = \exists u.\exists v.\exists z.((w = u \sharp v) \wedge (delete\ (u, x \sharp y, z)) \wedge (permute\ (z, v)))$$

$$delete\ :\ a \times List\ a \times List\ a \to \Omega$$
$$(delete\ (x, [], y)) = \bot$$
$$(delete\ (x, y \sharp z, w)) = ((x = y) \wedge (w = z)) \vee \exists v.((w = y \sharp v) \wedge (delete\ (x, z, v)))$$

$$sorted\ :\ List\ Int \to \Omega$$
$$(sorted\ []) = \top$$
$$(sorted\ x \sharp y) = if\ y = []\ then\ \top\ else\ \exists u.\exists v.((y = u \sharp v) \wedge (x \leq u) \wedge (sorted\ y)).$$

The intended meaning of *append* is that it is true iff its third argument is the concatenation of its first two arguments. The intended meaning of *permute* is that it is true iff its second

$\underline{(\mathit{upto\_sometime}\ (\varphi, 2))}$

$\mathit{if}\ \underline{2 = 0}\ \mathit{then}\ \varphi\ \mathit{else}\ ((\mathit{sometime}\ (\varphi, 2)) \wedge (\mathit{upto\_sometime}\ (\varphi, 2-1)))$

$\underline{\mathit{if}\ \bot\ \mathit{then}\ \varphi\ \mathit{else}\ ((\mathit{sometime}\ (\varphi, 2)) \wedge (\mathit{upto\_sometime}\ (\varphi, 2-1)))}$

$\underline{(\mathit{sometime}\ (\varphi, 2))} \wedge (\mathit{upto\_sometime}\ (\varphi, 2-1))$

$(\mathit{if}\ \underline{2 = 0}\ \mathit{then}\ \varphi\ \mathit{else} \bigcirc(\mathit{sometime}\ (\varphi, 2-1))) \wedge (\mathit{upto\_sometime}\ (\varphi, 2-1))$

$(\underline{\mathit{if}\ \bot\ \mathit{then}\ \varphi\ \mathit{else} \bigcirc(\mathit{sometime}\ (\varphi, 2-1))}) \wedge (\mathit{upto\_sometime}\ (\varphi, 2-1))$

$\bigcirc(\underline{\mathit{sometime}\ (\varphi, 2-1)}) \wedge (\mathit{upto\_sometime}\ (\varphi, 2-1))$

$\bigcirc\,(\mathit{if}\ \underline{(2-1)} = 0\ \mathit{then}\ \varphi\ \mathit{else} \bigcirc(\mathit{sometime}\ (\varphi, (2-1)-1))) \wedge (\mathit{upto\_sometime}\ (\varphi, 2-1))$

$\bigcirc\,(\mathit{if}\ \underline{1 = 0}\ \mathit{then}\ \varphi\ \mathit{else} \bigcirc(\mathit{sometime}\ (\varphi, (2-1)-1))) \wedge (\mathit{upto\_sometime}\ (\varphi, 2-1))$

$\bigcirc\,(\underline{\mathit{if}\ \bot\ \mathit{then}\ \varphi\ \mathit{else} \bigcirc(\mathit{sometime}\ (\varphi, (2-1)-1))}) \wedge (\mathit{upto\_sometime}\ (\varphi, 2-1))$

$\bigcirc\bigcirc(\underline{\mathit{sometime}\ (\varphi, (2-1)-1)}) \wedge (\mathit{upto\_sometime}\ (\varphi, 2-1))$

$\bigcirc\bigcirc\,(\mathit{if}\ \underline{(2-1)} - 1 = 0\ \mathit{then}\ \varphi\ \mathit{else} \bigcirc(\mathit{sometime}\ (\varphi, ((2-1)-1)-1))) \wedge$
$(\mathit{upto\_sometime}\ (\varphi, 2-1))$

$\vdots$

$\bigcirc\bigcirc\,(\mathit{if}\ \underline{0 = 0}\ \mathit{then}\ \varphi\ \mathit{else} \bigcirc(\mathit{sometime}\ (\varphi, ((2-1)-1)-1))) \wedge (\mathit{upto\_sometime}\ (\varphi, 2-1))$

$\bigcirc\bigcirc\,(\underline{\mathit{if}\ \top\ \mathit{then}\ \varphi\ \mathit{else} \bigcirc(\mathit{sometime}\ (\varphi, ((2-1)-1)-1))}) \wedge (\mathit{upto\_sometime}\ (\varphi, 2-1))$

$\bigcirc\bigcirc\varphi \wedge \underline{(\mathit{upto\_sometime}\ (\varphi, 2-1))}$

$\vdots$

$\bigcirc\bigcirc\varphi \wedge \bigcirc\varphi \wedge (\underline{\mathit{upto\_sometime}\ (\varphi, (2-1)-1)})$

$\vdots$

$\bigcirc\bigcirc\varphi \wedge \bigcirc\varphi \wedge \varphi$

Figure 3: Computation of $(\mathit{upto\_sometime}\ (\varphi, 2))$

argument is a permutation of its first argument. The intended meaning of *delete* is that it is true iff its third argument is the result of deleting its first argument from its second argument. The intended meaning of *sorted* is that it is true iff its argument is an increasingly ordered list of integers.

The notable feature of the above definitions is the presence of existential quantifiers in the bodies of the statements, so not surprisingly the key statement that makes all this work is concerned with the existential quantifier. To motivate this, consider the computation in Figure 4 that results from the goal $(\mathit{append}\ (1 \sharp [], 2 \sharp [], x))$. At one point in the computation, the following term is reached:

$$\exists r'.\exists x'.\exists y'.((1 = r') \wedge ([] = x') \wedge (x = r' \sharp y') \wedge (\mathit{append}\ (x', 2 \sharp [], y'))).$$

An obviously desirable simplification that can be made to this term is to eliminate the local

variable r' since there is a 'value' (that is, 1) for it. This leads to the term

$$\exists x'.\exists y'.(([] = x') \wedge (x = 1 \sharp y') \wedge (\mathit{append}\ (x', 2 \sharp [], y'))).$$

Similarly, one can eliminate x' to obtain

$$\exists y'.((x = 1 \sharp y') \wedge (\mathit{append}\ ([], 2 \sharp [], y'))).$$

After some more computation, the answer $x = 1 \sharp 2 \sharp []$ results. Now the statements that make all this possible are

$$\exists x_1.\cdots\exists x_n.(\boldsymbol{x} \wedge (x_i = \boldsymbol{u}) \wedge \boldsymbol{y}) = \exists x_1.\cdots\exists x_{i-1}.\exists x_{i+1}.\cdots\exists x_n.(\boldsymbol{x}\{x_i/\boldsymbol{u}\} \wedge \boldsymbol{y}\{x_i/\boldsymbol{u}\})$$
$$\exists x_1.\cdots\exists x_n.(\boldsymbol{x} \wedge (\boldsymbol{u} = x_i) \wedge \boldsymbol{y}) = \exists x_1.\cdots\exists x_{i-1}.\exists x_{i+1}.\cdots\exists x_n.(\boldsymbol{x}\{x_i/\boldsymbol{u}\} \wedge \boldsymbol{y}\{x_i/\boldsymbol{u}\}),$$

which come from the definition of $\Sigma : (a \to \Omega) \to \Omega$ in standard equality theory and have λ-abstractions in their heads.

$\underline{(\mathit{append}\ (1 \sharp [], 2 \sharp [], x))}$
$((\underline{1 \sharp [] = []}) \wedge (2 \sharp [] = x)) \vee \exists r'.\exists x'.\exists y'.((1 \sharp [] = r' \sharp x') \wedge (x = r' \sharp y') \wedge (\mathit{append}\ (x', 2 \sharp [], y')))$
$\underline{(\bot \wedge (2 \sharp [] = x))} \vee \exists r'.\exists x'.\exists y'.((1 \sharp [] = r' \sharp x') \wedge (x = r' \sharp y') \wedge (\mathit{append}\ (x', 2 \sharp [], y')))$
$\underline{\bot \vee \exists r'.\exists x'.\exists y'.((1 \sharp [] = r' \sharp x') \wedge (x = r' \sharp y') \wedge (\mathit{append}\ (x', 2 \sharp [], y')))}$
$\exists r'.\exists x'.\exists y'.(\underline{(1 \sharp [] = r' \sharp x')} \wedge (x = r' \sharp y') \wedge (\mathit{append}\ (x', 2 \sharp [], y')))$
$\underline{\exists r'.\exists x'.\exists y'.((1 = r') \wedge ([] = x') \wedge (x = r' \sharp y') \wedge (\mathit{append}\ (x', 2 \sharp [], y')))}$
$\underline{\exists x'.\exists y'.(([] = x') \wedge (x = 1 \sharp y') \wedge (\mathit{append}\ (x', 2 \sharp [], y')))}$
$\exists y'.((x = 1 \sharp y') \wedge \underline{(\mathit{append}\ ([], 2 \sharp [], y'))})$
$\vdots$
$\underline{\exists y'.((x = 1 \sharp y') \wedge (y' = 2 \sharp []))}$
$x = 1 \sharp 2 \sharp []$

Figure 4: Computation of $(\mathit{append}\ (1 \sharp [], 2 \sharp [], x))$

This example illustrates how the definitions in the standard equality theory allow the traditional functional programming style to be extended to encompass the relational style of logic programming. The definitions of predicates look a little different to the way one would write them in, say, Prolog. A mechanical translation of a Prolog definition into one that runs in this style of programming simply involves using the completion of the Prolog definition. The definition here of *append* is essentially the completion of the Prolog version of *append*. Alternatively, one can specialise the completion to the [] and $\sharp$ cases, as has been done here for the definitions of *permute*, *delete*, and *sorted*. One procedural difference of note is that Prolog's method of returning answers one at a time via backtracking is replaced here by returning all answers together as a disjunction (or a set). Thus the computation in Figure 5 from the goal

$$(\mathit{append}\ (x, y, 1 \sharp 2 \sharp []))$$

has the answer

$$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee ((x = 1 \sharp 2 \sharp []) \wedge (y = [])).$$

The style of programming typified by this example is called *programming with abstractions* [Llo03].

Example 6.3. Consider a belief base for an agent that contains the definition

$$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ x = A\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0),$$

where $A, B, C : \sigma$, $f : \sigma \to \mathit{Nat}$ and $\boldsymbol{B}$ is the belief modality for the agent. With such a definition, it is straightforward to compute in the 'forward' direction. Thus $(f\ B)$ can be computed in the obvious way to produce the answer 21 and the result $\boldsymbol{B}((f\ B) = 21)$.

Less obviously, the definition can be used to compute in the 'reverse' direction. For example, consider the computation of $\{x \mid (f\ x) = 42\}$ in Figure 6, which produces the answer $\{A, C\}$. This computation makes essential use of the equations

$$(\boldsymbol{w}\ \ \mathit{if}\ \boldsymbol{x} = \boldsymbol{t}\ \mathit{then}\ u\ \mathit{else}\ v) = \mathit{if}\ \boldsymbol{x} = \boldsymbol{t}\ \mathit{then}\ (\boldsymbol{w}\{\boldsymbol{x}/\boldsymbol{t}\}\ u)\ \mathit{else}\ (\boldsymbol{w}\ v)$$

% where $\boldsymbol{x}$ is a variable.

$$(\mathit{if}\ \boldsymbol{x} = \boldsymbol{t}\ \mathit{then}\ u\ \mathit{else}\ v\ \ \boldsymbol{w}) = \mathit{if}\ \boldsymbol{x} = \boldsymbol{t}\ \mathit{then}\ (u\ \boldsymbol{w}\{\boldsymbol{x}/\boldsymbol{t}\})\ \mathit{else}\ (v\ \boldsymbol{w})$$

% where $\boldsymbol{x}$ is a variable.

from the standard equality theory.

Example 6.4. This example illustrates how typical database queries can be answered. Consider the definition

$$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ x = (A, Z)\ \mathit{then}\ (42, 11)\ \mathit{else}\ \mathit{if}\ x = (B, X)\ \mathit{then}\ (21, 7)\ \mathit{else}\ \mathit{if}\ x = (C, Y)\ \mathit{then}\ (42, 17)\ \mathit{else}\ (0, 0)),$$

where $A, B, C : \sigma$, $X, Y, Z : \tau$ and $f : \sigma \times \tau \to \mathit{Nat} \times \mathit{Nat}$.

Figure 7 shows the computation using $\boldsymbol{B}$ of

$$\exists y.((y = (f\ (C, v))) \wedge ((\mathit{proj}_1\ y) = 42)),$$

where proj_1 projects onto the first component of a pair of natural numbers. The result of the computation is $\boldsymbol{B}(\exists y.((y = (f\ (C, v))) \wedge ((\mathit{proj}_1\ y) = 42)) = (v = Y))$ and the answer is $v = Y$.

Figure 8 shows the computation using $\boldsymbol{B}$ of

$$\{y \mid \exists x.((y = (f\ x)) \wedge ((\mathit{proj}_1\ y) = 42))\}.$$

The result of the computation is $\boldsymbol{B}(\{y \mid \exists x.((y = (f\ x)) \wedge ((\mathit{proj}_1\ y) = 42))\} = \{(42, 11), (42, 17)\})$ and the answer is $\{(42, 11), (42, 17)\}$.

Many other (non-modal) examples of computation can be found in [Tho99], [Llo99], and [Llo03].

$\underline{(append\ (x, y, 1 \sharp 2 \sharp []))}$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee \exists r'.\exists x'.\exists y'.((x = r' \sharp x') \wedge \underline{(1 \sharp 2 \sharp [] = r' \sharp y')} \wedge (append\ (x', y, y')))$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee \underline{\exists r'.\exists x'.\exists y'.((x = r' \sharp x') \wedge (1 = r') \wedge (2 \sharp [] = y') \wedge (append\ (x', y, y')))}$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee \underline{\exists x'.\exists y'.((x = 1 \sharp x') \wedge (2 \sharp [] = y') \wedge (append\ (x', y, y')))}$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee \exists x'.((x = 1 \sharp x') \wedge \underline{(append\ (x', y, 2 \sharp []))})$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee \exists x'.\underline{((x = 1 \sharp x') \wedge (((x' = []) \wedge (y = 2 \sharp [])) \vee}$
$\underline{\exists r''.\exists x''.\exists y''.((x' = r'' \sharp x'') \wedge (2 \sharp [] = r'' \sharp y'') \wedge (append\ (x'', y, y'')))))}$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee \underline{\exists x'.(((x = 1 \sharp x') \wedge (x' = []) \wedge (y = 2 \sharp [])) \vee}$
$\underline{((x = 1 \sharp x') \wedge \exists r''.\exists x''.\exists y''.((x' = r'' \sharp x'') \wedge (2 \sharp [] = r'' \sharp y'') \wedge (append\ (x'', y, y'')))))}$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee \underline{\exists x'.((x = 1 \sharp x') \wedge (x' = []) \wedge (y = 2 \sharp []))} \vee$
$\exists x'.((x = 1 \sharp x') \wedge \exists r''.\exists x''.\exists y''.((x' = r'' \sharp x'') \wedge (2 \sharp [] = r'' \sharp y'') \wedge (append\ (x'', y, y''))))$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee$
$\exists x'.((x = 1 \sharp x') \wedge \exists r''.\exists x''.\exists y''.((x' = r'' \sharp x'') \wedge \underline{(2 \sharp [] = r'' \sharp y'')} \wedge (append\ (x'', y, y''))))$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee$
$\exists x'.((x = 1 \sharp x') \wedge \underline{\exists r''.\exists x''.\exists y''.((x' = r'' \sharp x'') \wedge (2 = r'') \wedge ([] = y'') \wedge (append\ (x'', y, y'')))})$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee$
$\exists x'.((x = 1 \sharp x') \wedge \underline{\exists x''.\exists y''.((x' = 2 \sharp x'') \wedge ([] = y'') \wedge (append\ (x'', y, y'')))})$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee$
$\exists x'.((x = 1 \sharp x') \wedge \exists x''.((x' = 2 \sharp x'') \wedge \underline{(append\ (x'', y, []))}))$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee \exists x'.((x = 1 \sharp x') \wedge \exists x''.((x' = 2 \sharp x'') \wedge$
$(((x'' = []) \wedge (y = [])) \vee \exists r'''.\exists x'''.\exists y'''.((x'' = r''' \sharp x''') \wedge \underline{([] = r''' \sharp y''')} \wedge (append\ (x''', y, y''')))))$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee \exists x'.((x = 1 \sharp x') \wedge \exists x''.((x' = 2 \sharp x'') \wedge$
$(((x'' = []) \wedge (y = [])) \vee \exists r'''.\exists x'''.\exists y'''.\underline{((x'' = r''' \sharp x''') \wedge \bot} \wedge (append\ (x''', y, y'''))))))$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee \exists x'.((x = 1 \sharp x') \wedge \exists x''.((x' = 2 \sharp x'') \wedge$
$(((x'' = []) \wedge (y = [])) \vee \exists r'''.\exists x'''.\exists y'''.\underline{(\bot \wedge (append\ (x''', y, y''')))})))$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee$
$\exists x'.((x = 1 \sharp x') \wedge \exists x''.((x' = 2 \sharp x'') \wedge (((x'' = []) \wedge (y = [])) \vee \underline{\exists r'''.\exists x'''.\exists y'''.\bot})))$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee$
$\exists x'.((x = 1 \sharp x') \wedge \exists x''.((x' = 2 \sharp x'') \wedge \underline{(((x'' = []) \wedge (y = [])) \vee \bot)}))$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee$
$\exists x'.((x = 1 \sharp x') \wedge \underline{\exists x''.((x' = 2 \sharp x'') \wedge (x'' = []) \wedge (y = []))})$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee$
$\underline{\exists x'.((x = 1 \sharp x') \wedge (x' = 2 \sharp []) \wedge (y = []))}$

$((x = []) \wedge (y = 1 \sharp 2 \sharp [])) \vee ((x = 1 \sharp []) \wedge (y = 2 \sharp [])) \vee ((x = 1 \sharp 2 \sharp []) \wedge (y = []))$

Figure 5: Computation of $(append\ (x, y, 1 \sharp 2 \sharp []))$

$\{x \mid \underline{(f\ x)} = 42\}$
$\{x \mid (\underline{(=\ \ \mathit{if}\ x = A\ \mathit{then}\ 42\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)}\ 42)\}$
$\{x \mid (\underline{\mathit{if}\ x = A\ \mathit{then}\ (=\ 42)\ \mathit{else}\ (=\ \ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)}\ 42)\}$
$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \underline{(42 = 42)}\ \mathit{else}\ ((=\ \ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)\ 42)\}$
$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ (\underline{(=\ \ \mathit{if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)}\ 42)\}$
$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \underline{(\mathit{if}\ x = B\ \mathit{then}\ (=\ 21)\ \mathit{else}\ (=\ \ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)\ 42)}\}$
$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \underline{(21 = 42)}\ \mathit{else}\ ((=\ \ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)\ 42)\}$
$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ (\underline{(=\ \ \mathit{if}\ x = C\ \mathit{then}\ 42\ \mathit{else}\ 0)}\ 42)\}$
$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \underline{(\mathit{if}\ x = C\ \mathit{then}\ (=\ 42)\ \mathit{else}\ (=\ 0)\ \ 42)}\}$
$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \underline{(42 = 42)}\ \mathit{else}\ (0 = 42)\}$
$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \top\ \mathit{else}\ \underline{(0 = 42)}\}$
$\{x \mid \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = B\ \mathit{then}\ \bot\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \top\ \mathit{else}\ \bot\}$

Figure 6: Computation of rank 0 using $\boldsymbol{B}$ of $\{x \mid (f\ x) = 42\}$

6.2 Proof

Next I turn to (pure) proof using a tableau system.

Consider the problem of determining whether a biterm φ is true or false in the intended pointed interpretation. Once again suppose that a formal definition of the intended pointed interpretation is not available, but that the theory $\mathcal{T}$ of the application is available. Then the problem can still be solved by showing that φ is a theorem of $\mathcal{T}$. In this case, φ is a consequence of $\mathcal{T}$, by the soundness of the proof system, so that φ must be true in the intended pointed interpretation. Theoremhood is, of course, undecidable in the case of higher-order logic, but it is still possible to prove theorems in many cases of practical interest.

More formally, the *proof problem* is as follows.

> Given a theory $\mathcal{T}$ and biterm φ, determine whether φ is a consequence of $\mathcal{T}$.

Here now are the details of a proof system that can determine consequence.

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

Definition 6.3. A *prefix* is a finite sequence of the form $1.\langle n_1, j_1\rangle.\ \ldots\ .\langle n_k, j_k\rangle$, where n_i is a positive integer and $j_i \in \{1, \ldots, m\}$, for $i = 1, \ldots, k$.

A *prefixed biterm* is an expression of the form $\sigma\ \varphi$, where σ is a prefix and φ is a biterm.

In the following, $\langle n, j\rangle$ is abbreviated to n_j.

The tableau system works as usual for modal logics. Given that the aim is to show that a biterm 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. In this paper, I concentrate on the (multi-modal) logic $\mathbf{K}_m$ (where the m refers to the number of modalities) which has the

$\underline{\exists y.((y = (f\ (C, v))) \wedge ((proj_1\ y) = 42))}$

$(proj_1\ \underline{(f\ (C, v))}) = 42$

$\underline{(proj_1\ \mathit{if}\ (C, v) = (A, Z)\ \mathit{then}\ (42, 11)\ \mathit{else\ if}\ (C, v) = (B, X)\ \mathit{then}\ (21, 7)}$
$\underline{\mathit{else\ if}\ (C, v) = (C, Y)\ \mathit{then}\ (42, 17)\ \mathit{else}\ (0, 0))} = 42$

$((= \ \mathit{if}\ (C, v) = (A, Z)\ \mathit{then}\ (proj_1\ (42, 11))\ \mathit{else}\ (proj_1\ \mathit{if}\ (C, v) = (B, X)\ \mathit{then}\ (21, 7)$
$\underline{\mathit{else\ if}\ (C, v) = (C, Y)\ \mathit{then}\ (42, 17)\ \mathit{else}\ (0, 0))}\ 42)$

$(\mathit{if}\ (C, v) = (A, Z)\ \mathit{then}\ (= \ (proj_1\ (42, 11)))\ \mathit{else}\ (= \ (proj_1\ \mathit{if}\ (C, v) = (B, X)\ \mathit{then}\ (21, 7)$
$\underline{\mathit{else\ if}\ (C, v) = (C, Y)\ \mathit{then}\ (42, 17)\ \mathit{else}\ (0, 0)))\ 42)}$

$\mathit{if}\ \underline{(C, v) = (A, Z)}\ \mathit{then}\ ((proj_1\ (42, 11)) = 42)\ \mathit{else}\ ((proj_1\ \mathit{if}\ (C, v) = (B, X)\ \mathit{then}\ (21, 7)$
$\mathit{else\ if}\ (C, v) = (C, Y)\ \mathit{then}\ (42, 17)\ \mathit{else}\ (0, 0)) = 42)$

$\mathit{if}\ \underline{(C = A)} \wedge (v = Z)\ \mathit{then}\ ((proj_1\ (42, 11)) = 42)\ \mathit{else}\ ((proj_1\ \mathit{if}\ (C, v) = (B, X)\ \mathit{then}\ (21, 7)$
$\mathit{else\ if}\ (C, v) = (C, Y)\ \mathit{then}\ (42, 17)\ \mathit{else}\ (0, 0)) = 42)$

$\mathit{if}\ \underline{\bot \wedge (v = Z)}\ \mathit{then}\ ((proj_1\ (42, 11)) = 42)\ \mathit{else}\ ((proj_1\ \mathit{if}\ (C, v) = (B, X)\ \mathit{then}\ (21, 7)$
$\mathit{else\ if}\ (C, v) = (C, Y)\ \mathit{then}\ (42, 17)\ \mathit{else}\ (0, 0)) = 42)$

$\underline{\mathit{if}\ \bot\ \mathit{then}\ ((proj_1\ (42, 11)) = 42)\ \mathit{else}\ ((proj_1\ \mathit{if}\ (C, v) = (B, X)\ \mathit{then}\ (21, 7)}$
$\underline{\mathit{else\ if}\ (C, v) = (C, Y)\ \mathit{then}\ (42, 17)\ \mathit{else}\ (0, 0)) = 42)}$

$\underline{(proj_1\ \mathit{if}\ (C, v) = (B, X)\ \mathit{then}\ (21, 7)\ \mathit{else\ if}\ (C, v) = (C, Y)\ \mathit{then}\ (42, 17)\ \mathit{else}\ (0, 0))} = 42$

$\vdots$

$\underline{(proj_1\ \mathit{if}\ (C, v) = (C, Y)\ \mathit{then}\ (42, 17)\ \mathit{else}\ (0, 0))} = 42$

$((= \ \mathit{if}\ (C, v) = (C, Y)\ \mathit{then}\ (proj_1\ (42, 17))\ \mathit{else}\ \underline{(proj_1\ (0, 0)))}\ 42)$

$\underline{(\mathit{if}\ (C, v) = (C, Y)\ \mathit{then}\ (= \ (proj_1\ (42, 17)))\ \mathit{else}\ (= \ (proj_1\ (0, 0)))\ 42)}$

$\mathit{if}\ \underline{(C, v) = (C, Y)}\ \mathit{then}\ ((proj_1\ (42, 17)) = 42)\ \mathit{else}\ ((proj_1\ (0, 0)) = 42)$

$\mathit{if}\ \underline{(C = C)} \wedge (v = Y)\ \mathit{then}\ ((proj_1\ (42, 17)) = 42)\ \mathit{else}\ ((proj_1\ (0, 0)) = 42)$

$\mathit{if}\ \underline{\top \wedge (v = Y)}\ \mathit{then}\ ((proj_1\ (42, 17)) = 42)\ \mathit{else}\ ((proj_1\ (0, 0)) = 42)$

$\mathit{if}\ (v = Y)\ \mathit{then}\ (\underline{(proj_1\ (42, 17))} = 42)\ \mathit{else}\ ((proj_1\ (0, 0)) = 42)$

$\mathit{if}\ (v = Y)\ \mathit{then}\ \underline{(42 = 42)}\ \mathit{else}\ ((proj_1\ (0, 0)) = 42)$

$\mathit{if}\ (v = Y)\ \mathit{then}\ \top\ \mathit{else}\ (\underline{(proj_1\ (0, 0))} = 42)$

$\mathit{if}\ (v = Y)\ \mathit{then}\ \top\ \mathit{else}\ \underline{(0 = 42)}$

$\mathit{if}\ (v = Y)\ \mathit{then}\ \top\ \mathit{else}\ \bot$

Figure 7: Computation of rank 0 using $\boldsymbol{B}$ of $\exists y.((y = (f\ (C, v))) \wedge ((proj_1\ y) = 42))$

$$\{y \mid \exists x.((y = \underline{(f\ x)}) \wedge ((proj_1\ y) = 42))\}$$
$$\{y \mid \exists x.(\underline{(y = \varphi) \wedge ((proj_1\ y) = 42)})\}$$
$$\{y \mid \exists x.((y = \varphi) \wedge (\underline{(proj_1\ \varphi)} = 42))\}$$
$$\vdots$$
$$\{y \mid \exists x.(\underline{(y = \varphi) \wedge (\mathit{if}\ x = (A, Z)\ \mathit{then}\ \top\ \mathit{else\ if}\ x = (B, X)\ \mathit{then}\ \bot}$$
$$\underline{\mathit{else\ if}\ x = (C, Y)\ \mathit{then}\ \top\ \mathit{else}\ \bot)})\}$$
$$\{y \mid \underline{\exists x.(\mathit{if}\ ((y = \varphi) \wedge (x = (A, Z)))\ \mathit{then}\ \top\ \mathit{else\ if}\ ((y = \varphi) \wedge (\mathit{if}\ x = (B, X)\ \mathit{then}\ \bot}$$
$$\underline{\mathit{else\ if}\ x = (C, Y)\ \mathit{then}\ \top\ \mathit{else}\ \bot)))})\}$$
$$\{y \mid \mathit{if}\ \underline{\exists x.((y = \varphi) \wedge (x = (A, Z)))}\ \mathit{then}\ \top\ \mathit{else}\ \exists x.(\mathit{if}\ ((y = \varphi) \wedge (\mathit{if}\ x = (B, X)\ \mathit{then}\ \bot$$
$$\mathit{else\ if}\ x = (C, Y)\ \mathit{then}\ \top\ \mathit{else}\ \bot)))\}$$
$$\vdots$$
$$\{y \mid \mathit{if}\ y = (42, 11)\ \mathit{then}\ \top\ \mathit{else\ if}\ \exists x.(\underline{(y = \varphi) \wedge (\mathit{if}\ x = (B, X)\ \mathit{then}\ \bot}$$
$$\underline{\mathit{else\ if}\ x = (C, Y)\ \mathit{then}\ \top\ \mathit{else}\ \bot)})\}$$
$$\vdots$$
$$\{y \mid \mathit{if}\ y = (42, 11)\ \mathit{then}\ \top\ \mathit{else\ if}\ y = (21, 7)\ \mathit{then}\ \bot\ \mathit{else\ if}\ y = (42, 17)\ \mathit{then}\ \top\ \mathit{else}\ \bot\}$$

Figure 8: Computation of rank 0 using $\boldsymbol{B}$ of $\{y \mid \exists x.((y = (f\ x)) \wedge ((proj_1\ y) = 42))\}$. Here $\varphi \equiv \mathit{if}\ x = (A, Z)\ \mathit{then}\ (42, 11)\ \mathit{else\ if}\ x = (B, X)\ \mathit{then}\ (21, 7)\ \mathit{else\ if}\ x = (C, Y)\ \mathit{then}\ (42, 17)\ \mathit{else}\ (0, 0)$

tableau system given by the rules in Figures 9 and 10 and for which the corresponding set of frames is K. Figure 9 gives the basic tableau rules. These rules handle the connectives, modalities, quantifiers, abstractions, and equality. Figure 10 gives the two rules for handling the global and local assumptions. Generally speaking, these rules are well known (see, for example, [FM98] and [Fit02]), but the versions here differ in some details, in particular, in the use of the admissibility assumption in the universal, abstraction, and substitutivity rules. The proof is completed when each branch is closed, that is, contains contradictory biterms.

Without loss of generality, all local and global assumptions are assumed to be closed, as is the theorem to be proved. Note, however, that during construction of a tableau, open biterms can be introduced by some tableau rules.

Definition 6.4. Let $\mathcal{T}$ be a theory. A *proof of rank* 0 *with respect to* $\mathcal{T}$ is a sequence $T_1, \ldots, T_n$ of trees labelled by prefixed biterms satisfying the following conditions.

1. T_1 consists of a single node labelled by $1\ \neg\varphi$, for some biterm φ.
2. For $i = 1, \ldots, n-1$, there is
   (a) a tableau rule R from Figure 9 or Figure 10 such that T_{i+1} is obtained from T_i,

i. if R is a conjunctive rule, by extending a branch with two nodes labelled by the prefixed biterms in the denominator of R,

ii. if R is a disjunctive rule, by splitting a branch so that the leaf node of the branch has two children each labelled by one of the prefixed biterms in the denominator of R,

iii. otherwise, by extending a branch with a node labelled by the prefixed biterm in the denominator of R,

provided that any prefixed biterms in the numerator of R already appear in the branch and any side-conditions of R are satisfied.

3. Each branch of T_n contains nodes labelled by $\sigma\ \psi$ and $\sigma\ \neg\psi$, for some prefix σ and biterm ψ.

Each T_i is called a *tableau of rank* 0.

A branch of a tableau of rank 0 is *closed* if it contains nodes labelled by $\sigma\ \psi$ and $\sigma\ \neg\psi$, for some prefix σ and biterm ψ; otherwise, the branch is *open*.

A tableau of rank 0 is *closed* if each branch is closed; otherwise, the tableau is *open*.

The biterm φ is called the *theorem* of the proof; this is denoted by $\mathcal{T} \vdash \varphi$.

The next task is to prove the soundness of the proof system. The key result that needs to be established is that, if a biterm has a proof using the rules in Figures 9 and 10, then the biterm is a K-consequence of the global and local assumptions. To this end, the concept of a satisfiable set of prefixed biterms is introduced.

Definition 6.5. Let L be a class of frames. A set S of prefixed biterms 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.

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. For every biterm type $\alpha \equiv \alpha_1 \to \cdots \to \alpha_n \to \Omega$, there exist $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$ such that if $\sigma\ \varphi$ is in S and φ has type α, then $\mathcal{V}(\varphi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$.

A tableau branch is said to be L-*satisfiable* if the set of prefixed biterms 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 6.3. *A closed tableau of rank 0 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 biterms 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 biterms $\sigma\ \psi$ and $\sigma\ \neg\psi$ on the branch. Let the type of ψ be $\alpha_1 \to \cdots \to \alpha_n \to \Omega$ and the domain elements satisfying Condition 4 of Definition 6.5 for this type be $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$. Thus $\mathcal{V}(\psi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$

(Conjunctive rules) For any prefix σ,

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

(Disjunctive rules) For any prefix σ,

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

(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 $i \in \{1, \ldots, m\}$,

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

(Σ rules) For any prefix σ, if $y_1, \ldots, y_n$ are variables new to the branch,

$$\frac{\sigma\ \ (\Sigma\ \varphi)}{\sigma\ \ \varphi\ y_1\ \ldots\ y_n} \qquad \frac{\sigma\ \ \neg(\Pi\ \varphi)}{\sigma\ \ \neg\varphi\ y_1\ \ldots\ y_n}$$

(Π rules) For any prefix σ, if $t_1, \ldots, t_k$ $(0 \leq k \leq n)$ are terms,

$$\frac{\sigma\ \ (\Pi\ \varphi)}{\sigma\ \ \varphi\ t_1\ \ldots\ t_k} \qquad \frac{\sigma\ \ \neg(\Sigma\ \varphi)}{\sigma\ \ \neg\varphi\ t_1\ \ldots\ t_k}$$

(Abstraction rules) For any prefix σ, if $\{x/t\}$ is admissible with respect to φ,

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

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

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

(Substitutivity rule) For any prefix σ, if φ is a biterm containing a free occurrence of the variable x, and $\{x/s\}$ and $\{x/t\}$ are admissible with respect to φ,

$$\frac{\begin{array}{l}\sigma\ \ s = t\\ \sigma\ \ \varphi\{x/s\}\end{array}}{\sigma\ \ \varphi\{x/t\}}$$

Figure 9: Basic tableau rules

(Global assumption rule) If the prefix σ already occurs on the branch, ψ is a global assumption, and $t_1, \ldots, t_k$ $(0 \leq k \leq n)$ are terms,

$$\frac{}{\sigma \;\; \psi \; t_1 \;\; \ldots \;\; t_k}$$

(Local assumption rule) If ψ is a local assumption and $t_1, \ldots, t_k$ $(0 \leq k \leq n)$ are terms,

$$\frac{}{1 \;\; \psi \; t_1 \;\; \ldots \;\; t_k}$$

Figure 10: Tableau rules for global and local assumptions

and $\mathcal{V}(\neg\psi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. However, it follows from Part 4 of Proposition 3.1 that $\mathcal{V}(\neg\psi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$ iff $\mathcal{V}(\psi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \bot$, which gives a contradiction. Thus the closed tableau is not satisfiable. □

Since, for any class L of frames, an L-satisfiable set of prefixed biterms is satisfiable, it follows from Proposition 6.3 that a closed tableau is not L-satisfiable.

Proposition 6.4. *If a tableau rule (from Figures 9 and 10) 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 it can be assumed that $\mathcal{B}$ is satisfiable. Suppose the set of prefixed biterms on $\mathcal{B}$ is satisfiable via the interpretation I, variable assignment ν, and mapping F. Also suppose the type of φ and ψ is $\alpha_1 \to \cdots \to \alpha_n \to \Omega$ and the domain elements satisfying Condition 4 of Definition 6.5 for this type are $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$. Each of the tableau rules is now examined in turn.

Conjunctive rules Consider the first of these. Since the original tableau is satisfiable, it follows that $\mathcal{V}(\varphi \wedge \psi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. Thus $\mathcal{V}(\varphi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$ and $\mathcal{V}(\psi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$, by Part 1 of Proposition 3.1. Thus the tableau extended by this rule is satisfiable.

The proofs of the other conjunctive rules are similar.

Disjunctive rules Consider the first of these. Since the original tableau is satisfiable, it follows that $\mathcal{V}(\varphi \vee \psi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. Thus $\mathcal{V}(\varphi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$ or $\mathcal{V}(\psi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$, by Part 2 of Proposition 3.1. Thus the tableau extended by this rule is satisfiable because at least one of the resulting branches is satisfiable.

The proofs of the other disjunctive rules are similar.

Double negation rule Since the original tableau is satisfiable, it follows that $\mathcal{V}(\neg\neg\varphi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. Then $\mathcal{V}(\varphi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$, by Parts 5 and 6 of Proposition 3.1. Thus the extended tableau is also satisfiable.

Possibility rules Consider the first of these. Since the original tableau is satisfiable, it follows that $\mathcal{V}(\Diamond_i\varphi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. By Part 2 of Proposition 3.6, for some w' such that $F(\sigma)\ R_i\ w'$, $\mathcal{V}(\varphi, I, w', \nu)\ d_1\ \ldots\ d_n = \top$. Now extend the definition of F so that $F(\sigma.n_i) = w'$. Thus $F(\sigma)\ R_i\ F(\sigma.n_i)$ and $\mathcal{V}(\varphi, I, F(\sigma.n_i), \nu)\ d_1\ \ldots\ d_n = \top$, 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.

Σ rules Consider the first of these. Since the original tableau is satisfiable, it follows that $\mathcal{V}((\Sigma\ \varphi), I, F(\sigma), \nu) = \top$. By Part 1 of Proposition 3.2, $\mathcal{V}((\Sigma\ \varphi), I, F(\sigma), \nu) = \top$ iff, for some $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$, $\mathcal{V}(\varphi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. Let y_i be a variable of type α_i that is new to the branch $(i = 1, \ldots, n)$. Define the variable assignment ν^* by $\nu^*(x) = \nu(x)$, for $x \notin \{y_1, \ldots, y_n\}$ and $\nu^*(y_i) = d_i$, for $i = 1, \ldots, n$. Since ν and ν^* agree on the free variables of φ, $\mathcal{V}(\varphi, I, F(\sigma), \nu) = \mathcal{V}(\varphi, I, F(\sigma), \nu^*)$, by Proposition 3.14. Thus $\mathcal{V}(\varphi\ y_1\ \ldots\ y_n, I, F(\sigma), \nu^*) = \mathcal{V}(\varphi, I, F(\sigma), \nu^*)\ d_1\ \ldots\ d_n = \mathcal{V}(\varphi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. Now, since ν and ν^* differ only on the y_i and the y_i are new to the branch, ν and ν^* agree on the free variables of each biterm on the original branch. By Proposition 3.14, it follows that the extended branch is satisfiable via the interpretation I, variable assignment ν^*, and mapping F.

The proof of the other Σ rule is similar, using Proposition 3.3.

Π rules Consider the first of these. Since the original tableau is satisfiable, it follows that $\mathcal{V}((\Pi\ \varphi), I, F(\sigma), \nu) = \top$. By Part 2 of Proposition 3.2, $\mathcal{V}((\Pi\ \varphi), I, F(\sigma), \nu) = \top$ iff, for each $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$, $\mathcal{V}(\varphi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. Let t_i be terms of type α_i and $\mathcal{V}(t_i, I, F(\sigma), \nu) = d_i$ $(i = 1, \ldots, k)$. Then $\mathcal{V}(\varphi\ t_1\ \ldots\ t_k, I, F(\sigma), \nu)\ d_{k+1}\ \ldots\ d_n = \mathcal{V}(\varphi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$, for each $d_{k+1}\ \ldots\ d_n$. If this is the first occurrence on the branch of a biterm having the same type as $\varphi\ t_1\ \ldots\ t_k$, then the $d_{k+1}\ \ldots\ d_n$ required by Condition 4 of Definition 6.5 can be chosen arbitrarily; otherwise, the $d_{k+1}\ \ldots\ d_n$ have already been fixed and they work here as well. Hence the extended branch is satisfiable via the interpretation I, variable assignment ν, and mapping F.

The proof for the other Π rule is similar, using Proposition 3.3.

Abstraction rules Consider the first of these. To show satisfiability of the tableau extended by this rule, it suffices to show that $\mathcal{V}(\varphi\{x/t\}, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. Since the original tableau is satisfiable, it follows that $\mathcal{V}((\lambda x.\varphi\ t), I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. The result now follows immediately, since by Proposition 3.35, $\mathcal{V}((\lambda x.\varphi\ t), I, F(\sigma), \nu) = \mathcal{V}(\varphi\{x/t\}, I, F(\sigma), \nu)$.

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 $\mathcal{V}((t = t), I, F(\sigma), \nu) = \top$. But this follows immediately since the meaning of $=$ is the identity mapping.

Substitutivity rule To show satisfiability of the tableau extended by this rule, it suffices to show that $\mathcal{V}(\varphi\{x/t\}, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. Since the original tableau is satisfiable, it follows that $\mathcal{V}(s = t, I, F(\sigma), \nu) = \top$ and $\mathcal{V}(\varphi\{x/s\}, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. The result now follows immediately from Proposition 3.31.

Global assumption rule Since the (original) tableau is satisfiable and ψ is a global assumption, ψ is valid in I. Thus $\mathcal{V}(\psi, I, F(\sigma), \nu) = \mathbb{T}$. Hence, for each $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$, $\mathcal{V}(\psi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. Let t_i be terms of type α_i and $\mathcal{V}(t_i, I, F(\sigma), \nu) = d_i$ $(i = 1, \ldots, k)$. Then $\mathcal{V}(\psi\ t_1\ \ldots\ t_k, I, F(\sigma), \nu)\ d_{k+1}\ \ldots\ d_n = \mathcal{V}(\psi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. If this is the first occurrence on the branch of a biterm having the same type as $\varphi\ t_1\ \ldots\ t_k$, then the $d_{k+1}\ \ldots\ d_n$ required by Condition 4 of Definition 6.5 can be chosen arbitrarily; otherwise, the $d_{k+1}\ \ldots\ d_n$ have already been fixed and they work here as well. Hence the extended branch is satisfiable via the interpretation I, variable assignment ν, and mapping F.

Local assumption rule Since the (original) tableau is satisfiable and ψ is a local assumption, ψ is valid at $F(1)$ in I. Thus $\mathcal{V}(\psi, I, F(1), \nu) = \mathbb{T}$. Hence, for each $d_i \in \mathcal{D}_{\alpha_i}$ $(i =$

$1, \ldots, n)$, $\mathcal{V}(\psi, I, F(1), \nu)\ d_1\ \ldots\ d_n = \top$. Let t_i be terms of type α_i and $\mathcal{V}(t_i, I, F(1), \nu) = d_i$ $(i = 1, \ldots, k)$. Then $\mathcal{V}(\psi\ t_1\ \ldots\ t_k, I, F(1), \nu)\ d_{k+1}\ \ldots\ d_n = \mathcal{V}(\psi, I, F(1), \nu)\ d_1\ \ldots\ d_n = \top$. If this is the first occurrence on the branch of a biterm having the same type as $\varphi\ t_1\ \ldots\ t_k$, then the $d_{k+1}\ \ldots\ d_n$ required by Condition 4 of Definition 6.5 can be chosen arbitrarily; otherwise, the $d_{k+1}\ \ldots\ d_n$ have already been fixed and they work here as well. Hence the extended branch is satisfiable via the interpretation I, variable assignment ν, and mapping F.

□

Now the soundness result can be proved.

Proposition 6.5. *Let $\mathcal{T}$ be a theory. Then the theorem of a proof of rank 0 with respect to $\mathcal{T}$ is a consequence of $\mathcal{T}$.*

Proof. Let $\mathcal{T}$ be $(\mathcal{G}, \mathcal{L})$. Suppose that φ is the theorem of a proof of rank 0 with respect to $\mathcal{T}$, but is not a consequence of $\mathcal{T}$. Since φ has a proof, there is a closed tableau T beginning with the prefixed biterm $1\ \neg\varphi$. Let T_1 be the tableau consisting of the single node $1\ \neg\varphi$. Thus T results from T_1 by applying various tableau rules in Figures 9 and 10.

Suppose that the type of φ is $\alpha_1 \to \cdots \to \alpha_n \to \Omega$. Since φ is not a consequence, there is an interpretation I, a world w in I, a variable assignment ν, and $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$ such that each $\psi \in \mathcal{G}$ is valid in I, each $\psi \in \mathcal{L}$ is valid at w in I, and $\mathcal{V}(\varphi, I, w, \nu)\ d_1\ \ldots\ d_n = \bot$. Define the mapping F by $F(1) = w$. Clearly, $\{1\ \neg\varphi\}$ is satisfiable, using I, ν, and F, where d_i $(i = 1, \ldots, n)$ are the domain elements associated with the type of φ, as required by Condition 4 of Definition 6.5. Thus T_1 is satisfiable.

Since T_1 is satisfiable, so is any tableau obtained from T_1 by applying tableau rules, by Proposition 6.4. Thus T is satisfiable. But this gives a contradiction, by Proposition 6.3, since T is closed. Thus φ is a consequence of $\mathcal{T}$. □

The next result is an immediate corollary of Proposition 6.5.

Proposition 6.6. *If a biterm φ has a proof of rank 0 with respect to the empty theory, then φ is valid.*

Figure 11 gives the derived rules for the quantifiers that follow directly from rules given in Figure 9. The most common case is given where φ is a formula. The rules can be formulated more generally, but I prefer to keep the simpler versions instead and go back to the rules in Figure 9 for more general situations.

(Existential rules) For any prefix σ, if φ is a formula and y is a variable new to the branch,

$$\frac{\sigma\ \ \exists x.\varphi}{\sigma\ \ \varphi\{x/y\}} \qquad \frac{\sigma\ \ \neg\forall x.\varphi}{\sigma\ \ \neg\varphi\{x/y\}}$$

(Universal rules) For any prefix σ, if $\{x/t\}$ is admissible with respect to the formula φ,

$$\frac{\sigma\ \ \forall x.\varphi}{\sigma\ \ \varphi\{x/t\}} \qquad \frac{\sigma\ \ \neg\exists x.\varphi}{\sigma\ \ \neg\varphi\{x/t\}}$$

Figure 11: Derived rules for existential and universal quantifiers

Sometimes it will be convenient to have a further rule available that introduces $\lambda x_1 \ldots \lambda x_n.\top$ or $\neg\lambda x_1 \ldots \lambda x_n.\bot$ ($n \geq 0$) into a proof, as in Figure 12. The proof of correctness of this rule is almost identical to that of the reflexivity rule. It enables a branch to be closed as soon as either of the prefixed biterms $\sigma \;\; \lambda x_1 \ldots \lambda x_n.\bot$ or $\sigma \;\; \neg\lambda x_1 \ldots \lambda x_n.\top$ appear in it.

($\top$ introduction rules) For any prefix σ,

$$\frac{}{\sigma \;\; \lambda x_1 \ldots \lambda x_n.\top} \qquad \frac{}{\sigma \;\; \neg\lambda x_1 \ldots \lambda x_n.\bot}$$

Figure 12: $\lambda x_1 \ldots \lambda x_n.\top$ introduction rules

To keep proofs as compact as possible, it is convenient to have some derived rules. The next result shows the correctness of the derived rules in Figure 13 that are useful since assumptions often have the implicational form $\varphi \longrightarrow \psi$.

Proposition 6.7.

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

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

2. *Use of the local assumption $\varphi \longrightarrow \psi$ is equivalent to use of either of the tableau rules:*

$$\frac{1 \;\; \varphi}{1 \;\; \psi} \qquad \frac{1 \;\; \neg\psi}{1 \;\; \neg\varphi}$$

Proof. 1. Suppose first that the tableau rule $\dfrac{\sigma \;\; \varphi}{\sigma \;\; \psi}$ 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 biterm

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

Since the biterm is equivalent to $\lambda x_1 \ldots \lambda x_n.\top$, this could be added (correctly) to the end of the branch. A disjunctive rule can then be used to produce two children

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

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 biterm $\sigma \;\; \varphi \longrightarrow \psi$, 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 $\dfrac{\sigma \;\; \varphi}{\sigma \;\; \psi}$ 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 6.4, I can assume that $\mathcal{B}$ is satisfiable. Suppose the set of prefixed biterms on $\mathcal{B}$ is satisfiable via the interpretation I, variable assignment ν, and mapping F. Since the (original) tableau is satisfiable, there exists $d_i \in \mathcal{D}_{\alpha_i}$ $(i = 1, \ldots, n)$ such that $\mathcal{V}(\varphi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$. Also $\mathcal{V}(\varphi \longrightarrow \psi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$, since $\varphi \longrightarrow \psi$ is a global assumption. Thus $\mathcal{V}(\psi, I, F(\sigma), \nu)\ d_1\ \ldots\ d_n = \top$, by Proposition 3.1, as required.

The proof for the rule $\dfrac{\sigma\ \ \neg\psi}{\sigma\ \ \neg\varphi}$ is similar.

2. The proof is similar to that of 1. □

(Derived rule for global implicational assumption) For any prefix σ, if $\varphi \longrightarrow \psi$ is a global assumption,

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

(Derived rule for local implicational assumption) If $\varphi \longrightarrow \psi$ is a local assumption,

$$\frac{1\ \ \varphi}{1\ \ \psi} \qquad \frac{1\ \ \neg\psi}{1\ \ \neg\varphi}$$

Figure 13: Derived rules for implicational assumptions

Note that, in the case of theorem proving, the definition of *if_then_else* given in standard equality theory can be replaced by the following more general theory.

$x \longrightarrow$ *if* x *then* y *else* $z = y$
$\neg x \longrightarrow$ *if* x *then* y *else* $z = z$.

Here are several examples to illustrate proof.

Example 6.5. Suppose a theory consists of just the local assumption $\forall x.\Box_i\varphi$, where φ is some biterm. Then the proof in Figure 14 shows that $\Box_i\forall x.\varphi$ is a consequence of the theory.

1	$\neg\Box_i\forall x.\varphi$	1.
1.1_i	$\neg\forall x.\varphi$	2.
1.1_i	$\neg\varphi\{x/y\}$	3.
1	$\forall x.\Box_i\varphi$	4.
1	$\Box_i\varphi\{x/y\}$	5.
1.1_i	$\varphi\{x/y\}$	6.

Figure 14: Proof of rank 0 of $\Box_i\forall x.\varphi$

An explanation of this proof is as follows. Item 1 is the negation of the biterm to be proved; 2 is from 1 by a possibility rule; 3 is from 2 by an existential rule; 4 is a local assumption; 5 is from 4 by a universal rule; 6 is from 5 by a necessity rule; now the branch closes by 3 and 6.

In effect, this proof establishes the Barcan biterm

$$\forall x.\Box_i\varphi \longrightarrow \Box_i\forall x.\varphi.$$

Being able to prove the Barcan biterm depends crucially on the constant domain assumption in the semantics, which leads to the particular form of the existential and universal rules in Figure 9. For the varying domain semantics, the corresponding rules are more complicated and the Barcan biterm fails to hold generally [FM98].

Example 6.6. Suppose a theory consists of just the local assumption $\Box_i\forall x.\varphi$, where φ is some biterm. Then the proof in Figure 15 shows that $\forall x.\Box_i\varphi$ is a consequence of the theory.

1	$\neg\forall x.\Box_i\varphi$	1.
1	$\neg\Box_i\varphi\{x/y\}$	2.
1.1_i	$\neg\varphi\{x/y\}$	3.
1	$\Box_i\forall x.\varphi$	4.
1.1_i	$\forall x.\varphi$	5.
1.1_i	$\varphi\{x/y\}$	6.

Figure 15: Proof of rank 0 of $\forall x.\Box_i\varphi$

An explanation of this proof is as follows. Item 1 is the negation of the biterm to be proved; 2 is from 1 by an existential rule; 3 is from 2 by a possibility rule; 4 is a local assumption; 5 is from 4 by a necessity rule; 6 is from 5 by a universal rule; now the branch closes by 3 and 6.

In effect, this proof establishes the converse Barcan biterm

$$\Box_i\forall x.\varphi \longrightarrow \forall x.\Box_i\varphi.$$

Example 6.7. Let α be a type, $p : \alpha \to \Omega$ a predicate, and t a term of type α. Suppose a theory consists of just the local assumption $\Box_j(p\ t)$. Then the proof in Figure 16 shows that $\Box_j\forall x.(((=\ t)\ x) \longrightarrow (p\ x))$ is a consequence of the theory.

1	$\neg\Box_j\forall x.(((=\ t)\ x) \longrightarrow (p\ x))$	1.
1.1_j	$\neg\forall x.(((=\ t)\ x) \longrightarrow (p\ x))$	2.
1.1_j	$\neg(((=\ t)\ y) \longrightarrow (p\ y))$	3.
1.1_j	$((=\ t)\ y)$	4.
1.1_j	$\neg(p\ y)$	5.
1.1_j	$\neg(p\ t)$	6.
1	$\Box_j(p\ t)$	7.
1.1_j	$(p\ t)$	8.

Figure 16: Proof of rank 0 of $\Box_j\forall x.(((=\ t)\ x) \longrightarrow (p\ x))$

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 from 4 and 5 by the substitutivity rule; 7 is a local assumption; 8 is from 7 by a necessity rule; now the branch closes by 6 and 8.

Example 6.8. Consider a formula φ containing the free variables $x_1, \ldots, x_n$ ($n \geq 0$). Suppose that a theory consists of the local assumptions

$$\Box_{l_1} \cdots \Box_{l_p} \forall(\varphi)$$
$$\forall(\Box_{l_1} \cdots \Box_{l_p} \varphi \longrightarrow \Box_{j_1} \cdots \Box_{j_r} \varphi).$$

The proof of rank 0 of $\Box_{j_1} \cdots \Box_{j_r} \forall(\varphi)$ in Figure 17 shows that $\Box_{j_1} \cdots \Box_{j_r} \forall(\varphi)$ is a consequence of the theory. In Figure 17, φ' denotes the formula $\varphi\{x_1/y_1, \ldots, x_n/y_n\}$, where $y_1, \ldots, y_n$ are variables new to the branch of the proof.

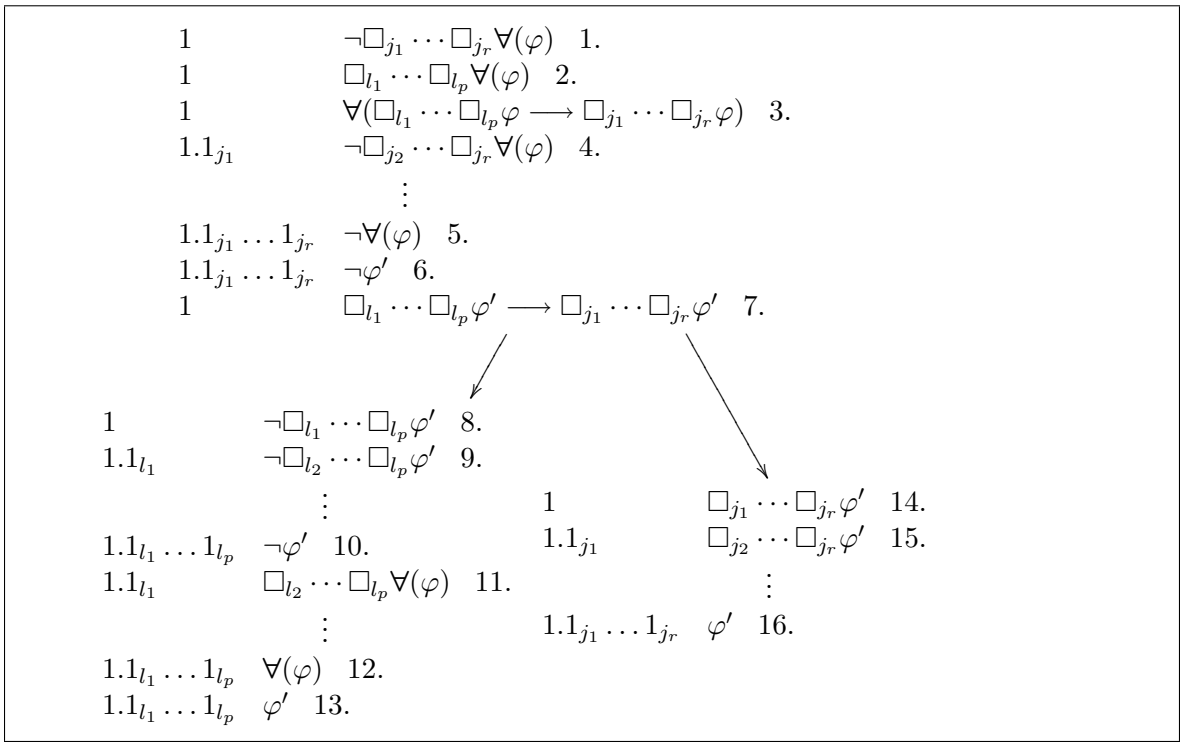

Figure 17: Proof of rank 0 of $\Box_{j_1} \cdots \Box_{j_r} \forall(\varphi)$

An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 is a local assumption; 3 is a local assumption; 4 is from 1 by a possibility rule; 5 is from 4 by possibility rules; 6 is from 5 by existential rules; 7 is from 3 by a universal rule; 8 and 14 are from 7 by a disjunctive rule; 9 is from 8 by a possibility rule; 10 is from 9 by possibility rules; 11 is from 2 by a necessity rule; 12 is from 11 by necessity rules; 13 is from 12 by universal rules; 15 is from 14 by a necessity rule; 16 is from 15 by necessity rules; now one branch closes by 10 and 13 and the other branch closes by 6 and 16.

Example 6.9. This example comes from [Bal98]. Consider the agents, Peter, John, and Wendy, with belief modalities $\boldsymbol{B}_p$, $\boldsymbol{B}_j$, and $\boldsymbol{B}_w$. 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.

$$\boldsymbol{B}_p time$$
$$\boldsymbol{B}_p\boldsymbol{B}_j place$$
$$\boldsymbol{B}_w(\boldsymbol{B}_p time \longrightarrow \boldsymbol{B}_j time)$$
$$\boldsymbol{B}_p\boldsymbol{B}_j(place \wedge time \longrightarrow appointment).$$

Suppose, in addition, that if Peter believes something, then Peter believes that he believes that thing; 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 local assumptions that capture all this are as follows.

$$\boldsymbol{B}_p\boldsymbol{\varphi} \longrightarrow \boldsymbol{B}_p\boldsymbol{B}_p\boldsymbol{\varphi}$$
$$\boldsymbol{B}_w\boldsymbol{\varphi} \longrightarrow \boldsymbol{B}_p\boldsymbol{\varphi}$$
$$\boldsymbol{B}_p\boldsymbol{B}_j\boldsymbol{\varphi} \longrightarrow \boldsymbol{B}_j\boldsymbol{B}_p\boldsymbol{\varphi},$$

where $\boldsymbol{\varphi}$ is a syntactical variable standing for an arbitrary formula. The proof below uses the derived rules that correspond to these local implicational assumptions.

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

$$\boldsymbol{B}_j\boldsymbol{B}_p appointment,$$

is a theorem of the belief base. The tableau proof of this formula is given in Figure 18.

An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 to 5 are local assumptions; 6 is from 1 by a derived rule from $\boldsymbol{B}_p\boldsymbol{B}_j\boldsymbol{\varphi} \longrightarrow \boldsymbol{B}_j\boldsymbol{B}_p\boldsymbol{\varphi}$; 7 is from 6 by a possibility rule; 8 is from 7 by a possibility rule; 9 is from 5 by a necessity rule; 10 is from 9 by a necessity rule; 11 and 25 are from 10 by a disjunctive rule; 12 and 15 are from 11 by a disjunctive rule; 13 is from 3 by a necessity rule; 14 is from 13 by a necessity rule; this branch now closes by 12 and 14; 16 is from 4 by a derived rule from $\boldsymbol{B}_w\boldsymbol{\varphi} \longrightarrow \boldsymbol{B}_p\boldsymbol{\varphi}$; 17 is from 16 by a necessity rule; 18 and 23 are from 17 by a disjunctive rule; 19 is from 18 by a possibility rule; 20 is from 2 by a derived rule from $\boldsymbol{B}_p\boldsymbol{\varphi} \longrightarrow \boldsymbol{B}_p\boldsymbol{B}_p\boldsymbol{\varphi}$; 21 is from 20 by a necessity rule; 22 is from 21 by a necessity rule; this branch now closes by 19 and 22; 24 is from 23 by a necessity rule; this branch now closes by 15 and 24; and the remaining branch closes by 8 and 25.

Example 6.10. Let $\mathfrak{T}$ be a theory, $q : \sigma \to \Omega$, and $t \in \mathfrak{B}_\sigma$. I show that

$$\mathfrak{T} \vdash \Box_{j_1} \cdots \Box_{j_r} \forall x.(((= t)\ x) \longrightarrow (q\ x)) \text{ iff } \mathfrak{T} \vdash \Box_{j_1} \cdots \Box_{j_r}(q\ t).$$

This example is relevant to belief acquisition.

It suffices to show that

$$\vdash \Box_{j_1} \cdots \Box_{j_r} \forall x.(((= t)\ x) \longrightarrow (q\ x)) \longrightarrow \Box_{j_1} \cdots \Box_{j_r}(q\ t)$$

and

$$\vdash \Box_{j_1} \cdots \Box_{j_r}(q\ t) \longrightarrow \Box_{j_1} \cdots \Box_{j_r} \forall x.(((= t)\ x) \longrightarrow (q\ x)).$$

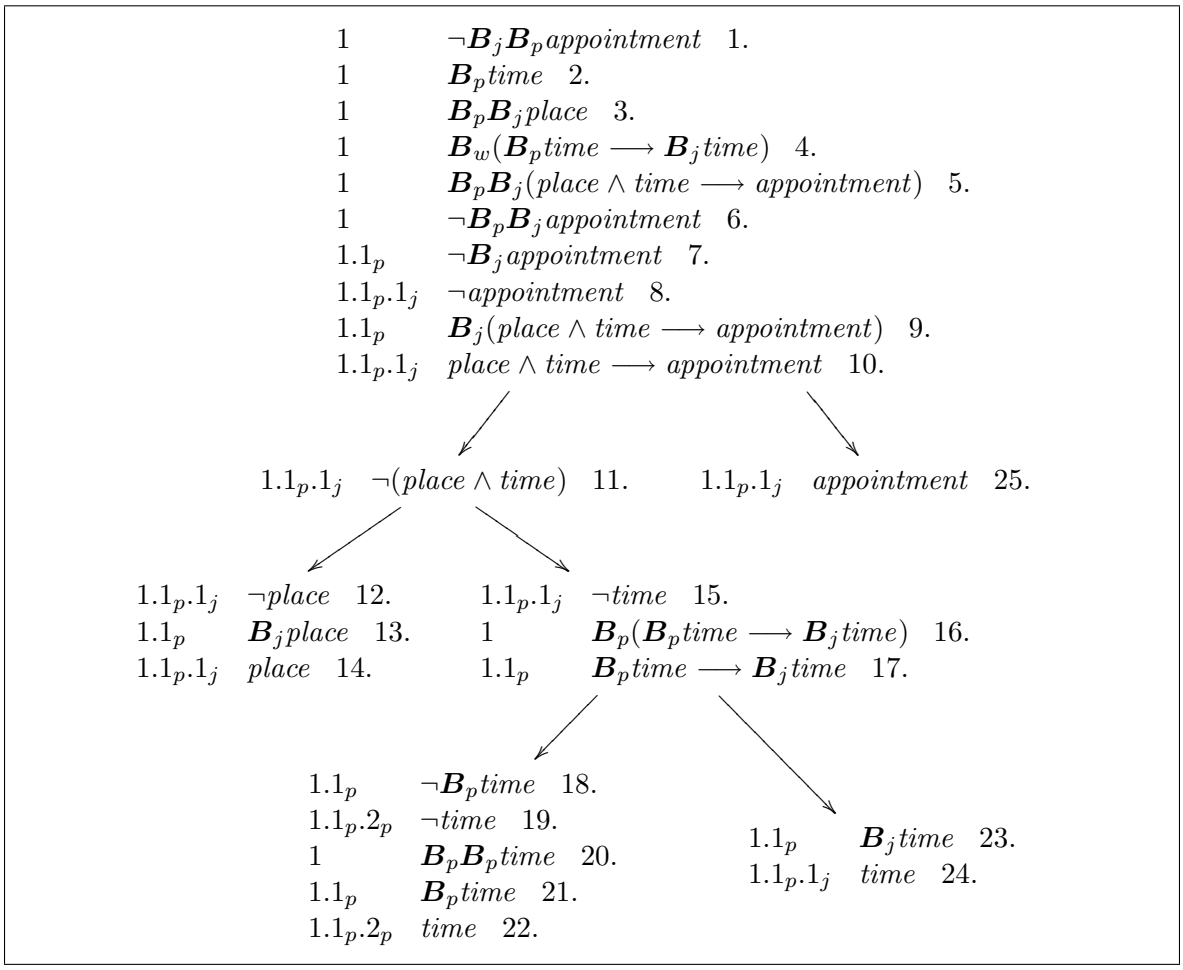

Figure 18: Proof of rank 0 of $\boldsymbol{B}_j \boldsymbol{B}_p appointment$

For the first of these, the proof is given in Figure 19. An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 and 3 are from 1 by a conjunctive rule; 4 is from 3 by a possibility rule; 5 is from 4 by possibility rules; 6 is from 2 by a necessity rule; 7 is from 6 by necessity rules; 8 is from 7 by a universal rule; 9 and 10 are from 8 by a disjunctive rule; 11 is the reflexivity rule; now the first branch closes by 9 and 11; and the second branch closes by 5 and 10.

For the second, the proof is given in Figure 20. An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 and 3 are from 1 by a conjunctive rule; 4 is from 3 by a possibility rule; 5 is from 4 by possibility rules; 6 is from 5 by an existential rule; 7 and 8 are from 6 by a conjunctive rule; 9 is from 8 and 7 by the substitutivity rule; 10 is from 2 by a necessity rule; 11 is from 10 by necessity rules; now the branch closes by 9 and 11.

So far I have considered the case where theories consist solely of formulas. Now I turn to the more general case where theories can contain biterms that are not formulas and show why this generalisation is useful.

Example 6.11. I start with a simple theory containing formulas. Consider the predicates

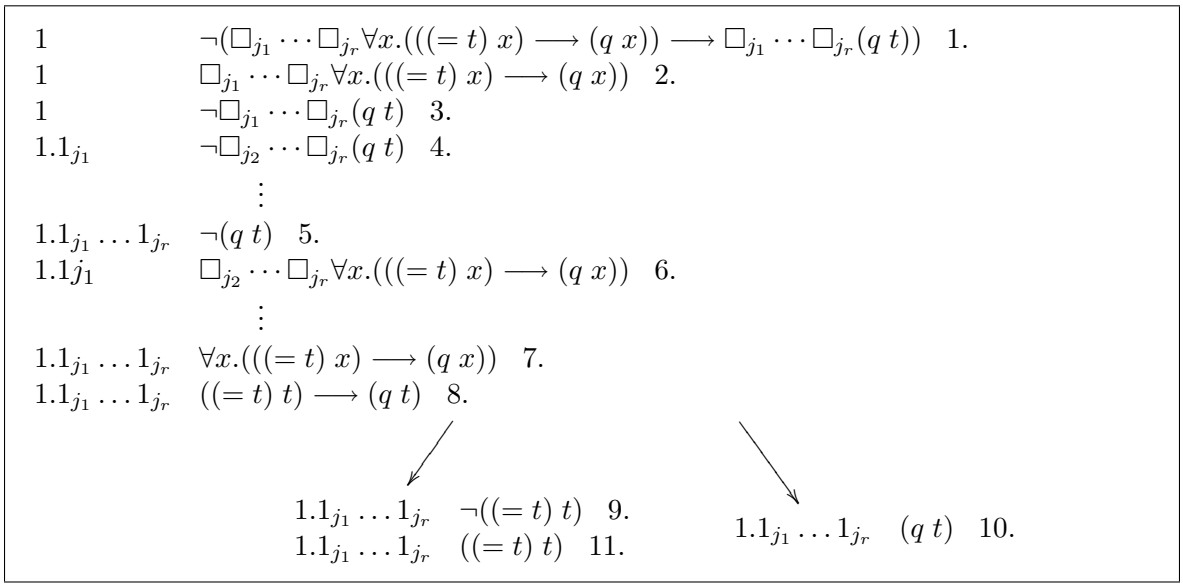

Figure 19: Proof of rank 0 of $\Box_{j_1}\cdots\Box_{j_r}\forall x.(((=t)\ x)\longrightarrow(q\ x))\longrightarrow\Box_{j_1}\cdots\Box_{j_r}(q\ t)$

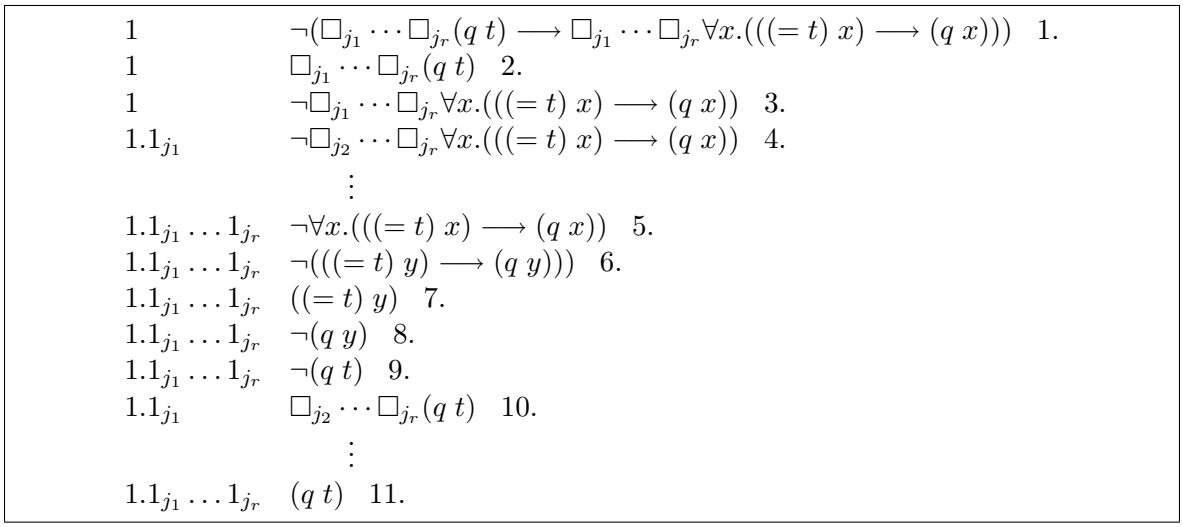

Figure 20: Proof of rank 0 of $\Box_{j_1}\cdots\Box_{j_r}(q\ t)\longrightarrow\Box_{j_1}\cdots\Box_{j_r}\forall x.(((=t)\ x)\longrightarrow(q\ x))$

$p,q,r:\sigma\to\Omega$ and the theory having the following local assumptions.

$$\forall x.((p\ x)\longrightarrow(q\ x))$$
$$\forall x.((q\ x)\longrightarrow(r\ x)).$$

Figure 21 gives the proof of the theorem $\forall x.((p\ x)\longrightarrow(r\ x))$. The soundness result, Proposition 6.5, shows that $\forall x.((p\ x)\longrightarrow(r\ x))$ is a consequence of theory.

The next step is to redo this example using biterms instead of formulas. Note that a biterm φ is valid at a world in an interpretation iff the formula $(\Pi\ \varphi)$ has the same property, by Proposition 3.2. Thus the implicit Π at the front of each formula in the preceding theory

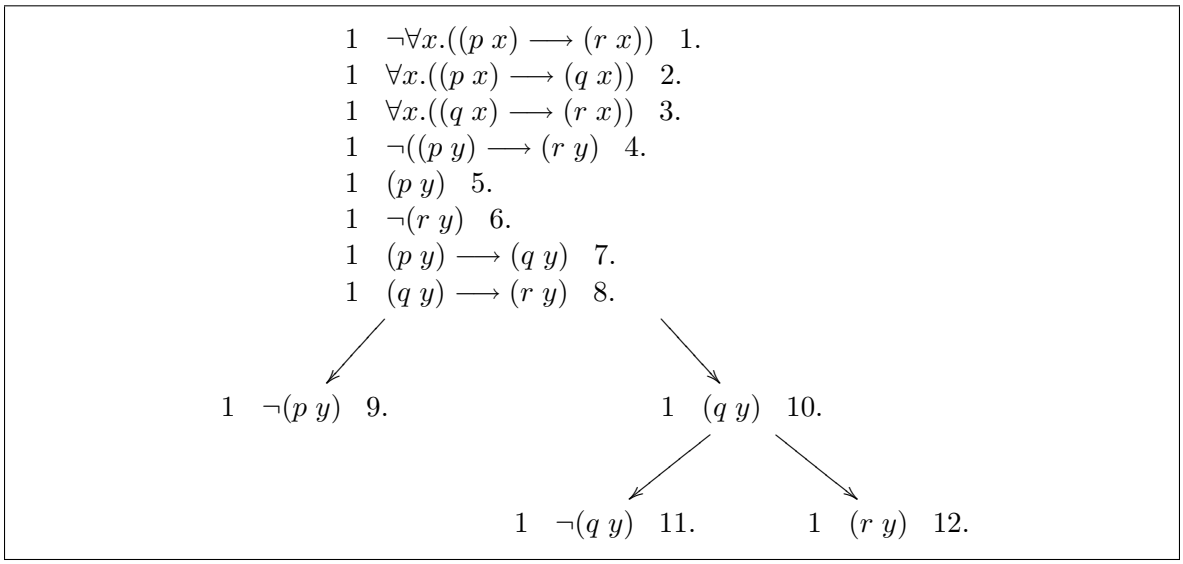

Figure 21: Proof of rank 0 of $\forall x.((p\ x) \longrightarrow (r\ x))$

can be removed to obtain the following (essentially) equivalent theory.

$\lambda x.((p\ x) \longrightarrow (q\ x))$
$\lambda x.((q\ x) \longrightarrow (r\ x))$.

Similarly, the preceding theorem can be replaced by the biterm $\lambda x.((p\ x) \longrightarrow (r\ x))$. Figure 22 now gives the proof of this theorem from the biterm theory. The step that gives the biterms in items 4 and 5 in this proof uses the equation

$$\neg\lambda x.(\varphi \longrightarrow \psi) = \lambda x.\varphi \wedge \neg\lambda x.\psi,$$

which can be used as a local assumption, by Proposition 3.1. Proposition 6.5 shows that $\lambda x.((p\ x) \longrightarrow (r\ x))$ is a consequence of biterm theory, a statement equivalent to the analogous statement for the preceding version using formulas.

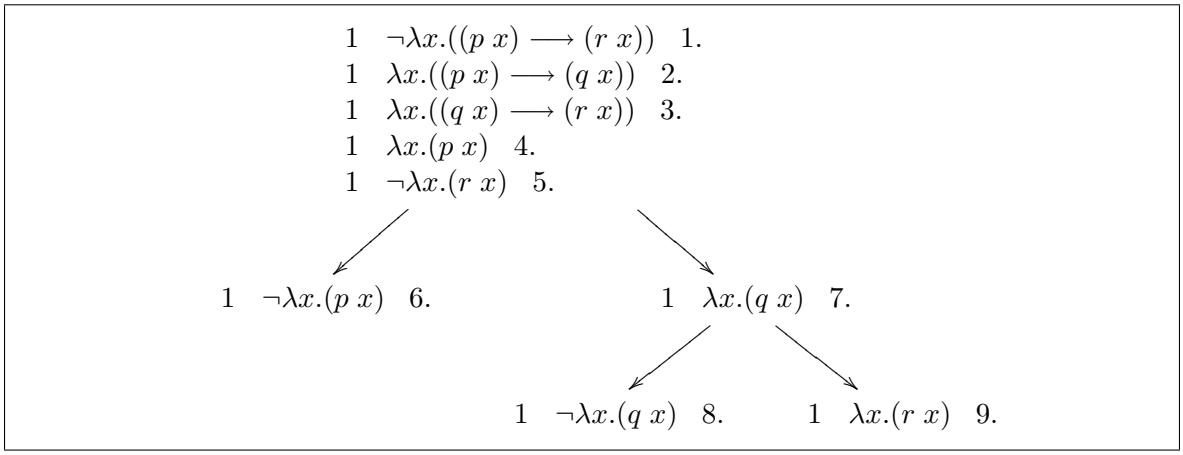

Figure 22: Proof of rank 0 of $\lambda x.((p\ x) \longrightarrow (r\ x))$

The biterm formulation is a little simpler than the formulation using formulas, but it can still be improved a lot. The point is that the use of variables complicates the theory and proof, and the variables are unnecessary. A better biterm formulation is the theory

$$p \longrightarrow q$$
$$q \longrightarrow r,$$

where $p \longrightarrow r$ is the theorem to be proved. Figure 23 contains this proof which is clearly as simple as any proof for this situation could possibly be. All reliance on variables has been eliminated and the proof is structurally equivalent to the proof for the case that p, q, and r are propositions (not predicates).

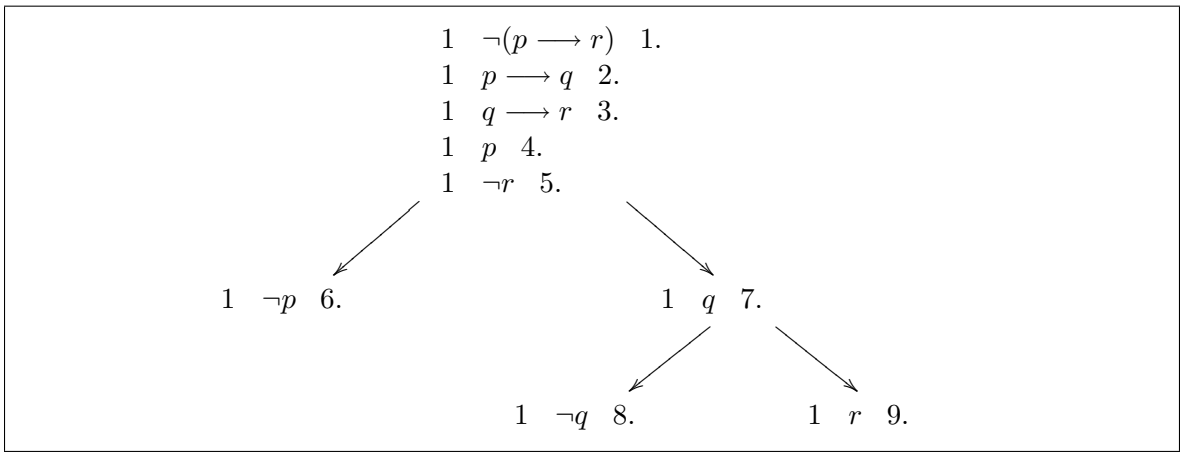

Figure 23: Proof of rank 0 of $p \longrightarrow r$

Biterms allow one to avoid the use of variables in knowledge representation and reasoning tasks when they are not really needed. Indeed, from the biterm perspective, many conventional uses of variables are simply gratuitous – they are only needed because, by the conventional definition, theories have to consist of formulas. Consequently, I advocate looking for opportunities to use biterms that make knowledge representation and reasoning tasks simpler. On the other hand, any situation that uses biterms can be forced into one that only uses formulas – simply apply Π to each assumption and to the biterm to be proved.

Here is an example of a propositional proof that extends immediately to the biterm case.

Example 6.12. Figure 24 gives a proof of the Kripke axiom

$$\Box_i(\varphi \longrightarrow \psi) \longrightarrow (\Box_i\varphi \longrightarrow \Box_i\psi),$$

for the case that φ and ψ are biterms.

An explanation of this proof is as follows. Item 1 is the negation of the biterm to be proved; 2 and 3 are from 1 by a conjunctive rule; 4 and 5 are from 3 by a conjunctive rule; 6 is from 5 by a possibility rule; 7 is from 4 by a necessity rule; 8 is from 2 by a necessity rule; 9 and 10 are from 8 by a disjunctive rule; now the first branch closes by 7 and 9, and the second branch closes by 6 and 10.

Other examples (for theories that consist of formulas) can be found in [FM98] and [Fit02].

Not surprisingly, the result of a computation can also be obtained by proving a theorem. This is shown by simulating a computation step using a proof.

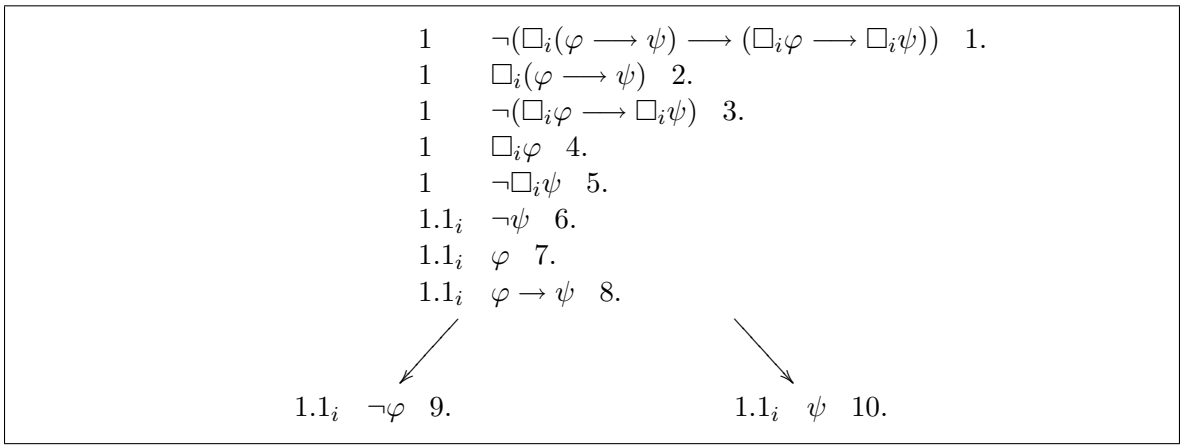

Figure 24: Proof of rank 0 of $\Box_i(\varphi \longrightarrow \psi) \longrightarrow (\Box_i\varphi \longrightarrow \Box_i\psi)$

Proposition 6.8. *Let $\mathcal{T} \equiv (\mathcal{G}, \mathcal{L})$ be a theory, t a term, and s a subterm of t at occurrence o, where the modal path to o in t is $k_1 \dots k_m$. Suppose there is*

1. *a formula $\Box_{j_1} \cdots \Box_{j_r}\Box_{k_1} \cdots \Box_{k_m}\forall(u = v)$ in $\mathcal{L}$, or*
2. *a formula $\forall(u = v)$ in $\mathcal{G}$, and*

a substitution θ that is admissible with respect to $u = v$ such that $u\theta$ is α-equivalent to s. Then $\mathcal{T} \vdash \Box_{j_1} \cdots \Box_{j_r}\forall(t = t[s/v\theta]_o)$.

Proof. First, the case when s is t is considered. Thus o is ε, the modal path is empty, and $u\theta$ is α-equivalent to t. It has to be shown that $\mathcal{T} \vdash \Box_{j_1} \cdots \Box_{j_r}\forall(u\theta = v\theta)$. The proof of this is given in Figure 25. An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 is a local assumption; 3 is from 1 by possibility rules; 4 is from 3 by existential rules (where φ is the substitution introducing the variables new to the branch); 5 is from 2 by necessity rules; 6 is from 5 by universal rules; now the branch closes by 4 and 6. The proof when the input equation is a global assumption is almost the same as this.

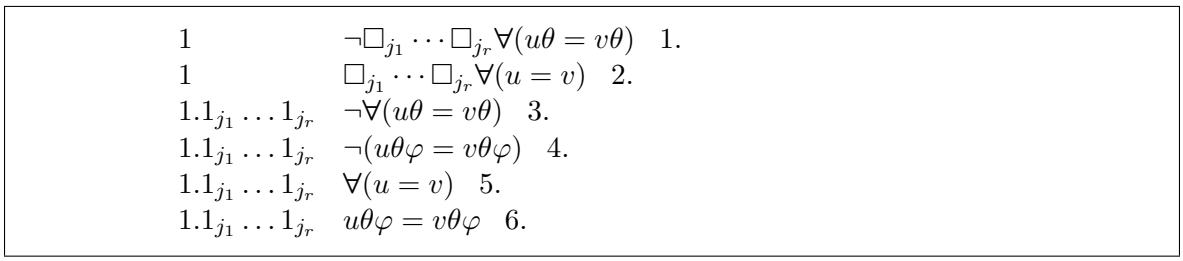

Figure 25: Proof of $\Box_{j_1} \cdots \Box_{j_r}\forall(u\theta = v\theta)$

Now the case when s is a *strict* subterm of t is considered. The proof for this case is by induction on the structure of t. If t is a variable or a constant, then there are no strict subterms and there is nothing more to consider.

Suppose that t is an abstraction $\lambda x.r$. Since s is a strict subterm of t, s must be a subterm of r at occurrence o', where $o = 1o'$. By either the induction hypothesis or the first part of the proof in case s is r, we have that $\mathcal{T} \vdash \Box_{j_1} \cdots \Box_{j_r}\forall(r = r[s/v\theta]_{o'})$. It has to be shown

that $\mathcal{T} \vdash \Box_{j_1} \cdots \Box_{j_r} \forall(\lambda x.r = (\lambda x.r)[s/v\theta]_o)$. The proof of this is given in Figure 26. An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 is from the induction hypothesis; 3 is from 1 by possibility rules; 4 is from 3 by existential rules (where φ is the substitution introducing the variables new to the branch); 5 is a global assumption (the axiom of extensionality); 6 is from 5 by universal rules; 7 is from 4 and 6 by the substitutivity rule; 8 is from 7 by an existential rule (z is a variable new to the branch); 9 is a global assumption (β-reduction); 10 is a global assumption (β-reduction); 11 is from 8 and 9 by the substitutivity rule; 12 is from 10 and 11 by the substitutivity rule; 13 is from 2 by necessity rules; 14 is from 13 by universal rules; now the branch closes by 12 and 14.

1	$\neg\Box_{j_1} \cdots \Box_{j_r} \forall(\lambda x.r = (\lambda x.r)[s/v\theta]_o)$ 1.
1	$\Box_{j_1} \cdots \Box_{j_r} \forall(r = r[s/v\theta]_{o'})$ 2.
$1.1_{j_1} \ldots 1_{j_r}$	$\neg\forall(\lambda x.r = (\lambda x.r)[s/v\theta]_o)$ 3.
$1.1_{j_1} \ldots 1_{j_r}$	$\neg((\lambda x.r)\varphi = ((\lambda x.r)[s/v\theta]_o)\varphi)$ 4.
$1.1_{j_1} \ldots 1_{j_r}$	$\forall f.\forall g.((f = g) = \forall y.((f\ y) = (g\ y)))$ 5.
$1.1_{j_1} \ldots 1_{j_r}$	$((\lambda x.r)\varphi = ((\lambda x.r)[s/v\theta]_o)\varphi) = \forall y.(((\lambda x.r)\varphi\ y) = (((\lambda x.r)[s/v\theta]_o)\varphi\ y))$ 6.
$1.1_{j_1} \ldots 1_{j_r}$	$\neg\forall y.(((\lambda x.r)\varphi\ y) = (((\lambda x.r)[s/v\theta]_o)\varphi\ y))$ 7.
$1.1_{j_1} \ldots 1_{j_r}$	$\neg(((\lambda x.r)\varphi\ z) = (((\lambda x.r)[s/v\theta]_o)\varphi\ z))$ 8.
$1.1_{j_1} \ldots 1_{j_r}$	$((\lambda x.r)\varphi\ z) = r\varphi\{x/z\}$ 9.
$1.1_{j_1} \ldots 1_{j_r}$	$(((\lambda x.r)[s/v\theta]_o)\varphi\ z) = r[s/v\theta]_{o'}\varphi\{x/z\}$ 10.
$1.1_{j_1} \ldots 1_{j_r}$	$\neg(r\varphi\{x/z\} = (((\lambda x.r)[s/v\theta]_o)\varphi\ z))$ 11.
$1.1_{j_1} \ldots 1_{j_r}$	$\neg(r\varphi\{x/z\} = r[s/v\theta]_{o'}\varphi\{x/z\})$ 12.
$1.1_{j_1} \ldots 1_{j_r}$	$\forall(r = r[s/v\theta]_{o'})$ 13.
$1.1_{j_1} \ldots 1_{j_r}$	$r\varphi\{x/z\} = r[s/v\theta]_{o'}\varphi\{x/z\}$ 14.

Figure 26: Proof of $\Box_{j_1} \cdots \Box_{j_r} \forall(\lambda x.r = (\lambda x.r)[s/v\theta]_o)$

Suppose that t is a modal term $(t_1\ t_2)$. Since s is a strict subterm of t, s either must be a subterm of t_1 at occurrence o', where $o = 1o'$, or a subterm of t_2 at occurrence o', where $o = 2o'$. The cases are essentially the same, so just the first is considered. By either the induction hypothesis or the first part of the proof in case s is t_1, we have that $\mathcal{T} \vdash \Box_{j_1} \cdots \Box_{j_r} \forall(t_1 = t_1[s/v\theta]_{o'})$. It has to be shown that $\mathcal{T} \vdash \Box_{j_1} \cdots \Box_{j_r} \forall((t_1\ t_2) = (t_1\ t_2)[s/v\theta]_o)$. The proof of this is given in Figure 27. An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 is from the induction hypothesis; 3 is from 1 by possibility rules; 4 is from 3 by existential rules (where φ is the substitution introducing the variables new to the branch); 5 is from 2 by necessity rules; 6 is from 5 by universal rules; 7 is by the reflexivity rule; 8 is from 6 and 7 by the substitutivity rule; now the branch closes by 4 and 8.

Suppose that t is a modal term $(t_1, \ldots, t_n)$. The proof for this case is similar to the preceding one.

Suppose that t is a modal term $\Box_{k_1} r$. Since s is a strict subterm of t, s must be a subterm of r at occurrence o', where $o = 1o'$. By either the induction hypothesis or the first part of the proof in case s is r, we have that $\mathcal{T} \vdash \Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \forall(r = r[s/v\theta]_{o'})$. It has to be shown that $\mathcal{T} \vdash \Box_{j_1} \cdots \Box_{j_r} \forall(\Box_{k_1} r = (\Box_{k_1} r)[s/v\theta]_o)$. The proof of this is given in Figure 28. An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 is from the induction hypothesis; 3 is from 1 by possibility rules; 4 is from 3 by existential

1	$\neg\Box_{j_1}\cdots\Box_{j_r}\forall((t_1\ t_2)=(t_1\ t_2)[s/v\theta]_o)$	1.
1	$\Box_{j_1}\cdots\Box_{j_r}\forall(t_1=t_1[s/v\theta]_{o'})$	2.
$1.1_{j_1}\ldots1_{j_r}$	$\neg\forall((t_1\ t_2)=(t_1\ t_2)[s/v\theta]_o)$	3.
$1.1_{j_1}\ldots1_{j_r}$	$\neg((t_1\ t_2)\varphi=(t_1\ t_2)[s/v\theta]_o\varphi)$	4.
$1.1_{j_1}\ldots1_{j_r}$	$\forall(t_1=t_1[s/v\theta]_{o'})$	5.
$1.1_{j_1}\ldots1_{j_r}$	$t_1\varphi=t_1[s/v\theta]_{o'}\varphi$	6.
$1.1_{j_1}\ldots1_{j_r}$	$(t_1\ t_2)\varphi=(t_1\ t_2)\varphi$	7.
$1.1_{j_1}\ldots1_{j_r}$	$(t_1\ t_2)\varphi=(t_1\ t_2)[s/v\theta]_o\varphi$	8.

Figure 27: Proof of $\Box_{j_1}\cdots\Box_{j_r}\forall((t_1\ t_2)=(t_1\ t_2)[s/v\theta]_o)$

rules (where φ is the substitution introducing the variables new to the branch); 5 is the global assumption $\Box_i(\boldsymbol{s}=\boldsymbol{t})\longrightarrow(\Box_i\boldsymbol{s}=\Box_i\boldsymbol{t})$; 6 and 10 are from 5 by a disjunctive rule; 7 is from 6 by a possibility rule; 8 is from 2 by necessity rules; 9 is from 8 by a universal rule; now the first branch closes by 7 and 9, and the second closes by 4 and 10.

1	$\neg\Box_{j_1}\cdots\Box_{j_r}\forall(\Box_{k_1}r=(\Box_{k_1}r)[s/v\theta]_o)$	1.
1	$\Box_{j_1}\cdots\Box_{j_r}\Box_{k_1}\forall(r=r[s/v\theta]_{o'})$	2.
$1.1_{j_1}\ldots1_{j_r}$	$\neg\forall(\Box_{k_1}r=(\Box_{k_1}r)[s/v\theta]_o)$	3.
$1.1_{j_1}\ldots1_{j_r}$	$\neg(\Box_{k_1}r\varphi=(\Box_{k_1}r)[s/v\theta]_o\varphi)$	4.
$1.1_{j_1}\ldots1_{j_r}$	$\Box_{k_1}(r\varphi=r[s/v\theta]_{o'}\varphi)\longrightarrow(\Box_{k_1}r\varphi=(\Box_{k_1}r)[s/v\theta]_o\varphi)$	5.
$1.1_{j_1}\ldots1_{j_r}$	$\neg\Box_{k_1}(r\varphi=r[s/v\theta]_{o'}\varphi)$	6.
$1.1_{j_1}\ldots1_{j_r}1_{k_1}$	$\neg(r\varphi=r[s/v\theta]_{o'}\varphi)$	7.
$1.1_{j_1}\ldots1_{j_r}1_{k_1}$	$\forall(r=r[s/v\theta]_{o'})$	8.
$1.1_{j_1}\ldots1_{j_r}1_{k_1}$	$r\varphi=r[s/v\theta]_{o'}\varphi$	9.
$1.1_{j_1}\ldots1_{j_r}$	$\Box_{k_1}r\varphi=(\Box_{k_1}r)[s/v\theta]_o\varphi$	10.

Figure 28: Proof of $\Box_{j_1}\cdots\Box_{j_r}\forall(\Box_{k_1}r=(\Box_{k_1}r)[s/v\theta]_o)$

□

An easy induction argument using Proposition 6.8 shows that the result of any computation is a theorem that can be proved without resorting to computation at all. In other words, from a theoretical point of view, computation is redundant. However, in practice, computation is essential, for at least two related reasons. The first is that in typical applications the most common reasoning task is a computational one, that of evaluating a function on some argument. The second is that a proof that simulates a computation is generally considerably more complicated than the computation.

Example 6.13. Consider a theory that contains the local assumption

$$\boldsymbol{B}_i\boldsymbol{B}_j\forall x.((p\ x)=(q\ x)\wedge(r\ x))$$

and the one step computation

$$\lambda x.\boldsymbol{B}_j\underline{(p\ x)}$$
$$\lambda x.\boldsymbol{B}_j((q\ x)\wedge(r\ x))$$

using $\boldsymbol{B}_i$ which shows that the result

$$\boldsymbol{B}_i(\lambda x.\boldsymbol{B}_j(p\ x) = \lambda x.\boldsymbol{B}_j((q\ x) \wedge (r\ x)))$$

is a consequence of the theory.

Figure 29 is a proof that simulates this computation. An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 is a local assumption; 3 is from 1 by a possibility rule; 4 is a global assumption (the axiom of extensionality); 5 is from 4 by universal rules; 6 is from 3 and 5 by the substitutivity rule; 7 is from 6 by an existential rule; 8 is a global assumption (β-reduction); 9 is a global assumption (β-reduction); 10 is from 7 and 8 by the substitutivity rule; 11 is from 9 and 10 by the substitutivity rule; 12 is the global assumption $\Box_i(\boldsymbol{s} = \boldsymbol{t}) \longrightarrow (\Box_i \boldsymbol{s} = \Box_i \boldsymbol{t})$; 13 and 17 are from 12 by a disjunctive rule; 14 is from 13 by a possibility rule; 15 is from 2 by necessity rules; 16 is from 15 by a universal rule; now the first branch closes by 14 and 16, and the second closes by 11 and 17.

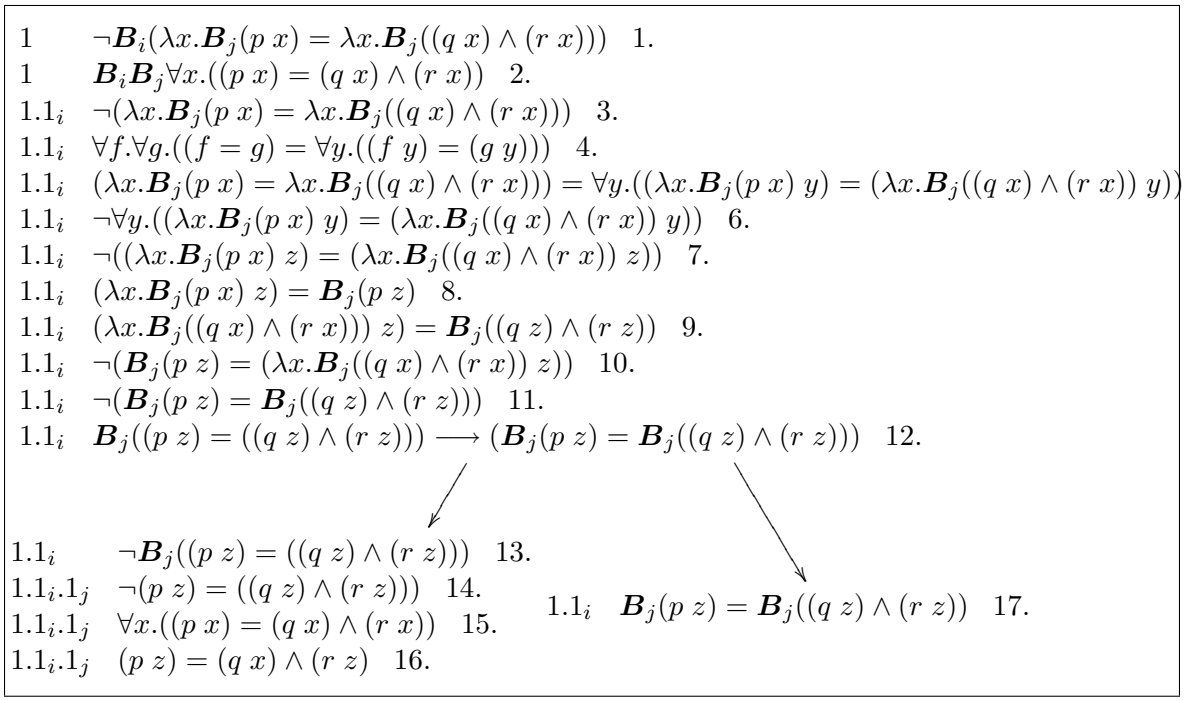

Figure 29: Proof of $\boldsymbol{B}_i(\lambda x.\boldsymbol{B}_j(p\ x) = \lambda x.\boldsymbol{B}_j((q\ x) \wedge (r\ x)))$

This example illustrates just how much more complicated the proof simulation of a computation is compared with the computation itself.

6.3 Computation and Proof

The next step is to combine computation and proof to give the full reasoning system. Before that, I point out some differences between computation and proof.

The primary difference is that computation is concerned with computing the meaning of arbitrary terms, whereas proof is concerned with finding the meaning of biterms. There is also a difference in the kind of theories that each requires: computation needs (primarily) an equational theory, whereas proof works with arbitrary theories. Typically, computation is

'efficient'; indeed, it is the basis of declarative programming languages. Proof usually involves much more search and therefore is likely to be much less efficient.

This subsection defines the combination of proof and computation, and shows the usefulness of this combination. Computation enhances proof with a powerful equational reasoning system; proof enhances computation by allowing some of the theory to be not in equational form. Here now are the details. By means of two mutually recursive definitions, the concepts of computation of rank k and proof of rank k are defined, for $k \geq 1$.

Definition 6.6. Let $\mathcal{T} \equiv (\mathcal{G}, \mathcal{L})$ be a theory and $k \geq 1$. A *computation of rank k using $\Box_{j_1} \cdots \Box_{j_r}$ with respect to $\mathcal{T}$* is a sequence $\{t_i\}_{i=1}^{n}$ of terms such that, for $i = 1, \ldots, n-1$, there is

1. a subterm s_i of t_i at occurrence o_i, where the modal path to o_i in t_i is $k_1 \ldots k_{m_i}$,
2. (a) a formula $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_i}} \forall(u_i = v_i)$ in $\mathcal{L}$, or
   (b) a formula $\forall(u_i = v_i)$ in $\mathcal{G}$, or
   (c) a formula $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_i}} \forall(u_i = v_i)$ that is the result of a computation of rank $k-1$ using $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_i}}$ with respect to $\mathcal{T}$, or
   (d) a formula $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_i}} \forall(u_i = v_i)$ that is the theorem of a proof of rank $k-1$ with respect to $\mathcal{T}$, and
3. a substitution θ_i that is admissible with respect to $u_i = v_i$

such that $u_i\theta_i$ is α-equivalent to s_i and t_{i+1} is $t_i[s_i/v_i\theta_i]_{o_i}$.

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

Each subterm s_i is called a *redex*.

Each formula $\Box_{j_1} \cdots \Box_{j_r} \Box_{k_1} \cdots \Box_{k_{m_i}} \forall(u_i = v_i)$ or $\forall(u_i = v_i)$ in Part 2 of the definition is called an *input equation*.

The formula $\Box_{j_1} \cdots \Box_{j_r} \forall(t_1 = t_n)$ is called the *result* of the computation.

Definition 6.7. Let $\mathcal{T} \equiv (\mathcal{G}, \mathcal{L})$ be a theory and $k \geq 1$. A *proof of rank k with respect to $\mathcal{T}$* is a sequence $T_1, \ldots, T_n$ of trees labelled by prefixed biterms satisfying the following conditions.

1. T_1 consists of a single node labelled by $1\ \neg\varphi$, for some biterm φ.
2. For $i = 1, \ldots, n-1$, there is either
   (a) a tableau rule R from Figure 9 or Figure 10 such that T_{i+1} is obtained from T_i,
      i. if R is a conjunctive rule, by extending a branch with two nodes labelled by the prefixed biterms in the denominator of R,
      ii. if R is a disjunctive rule, by splitting a branch so that the leaf node of the branch has two children each labelled by one of the prefixed biterms in the denominator of R,
      iii. otherwise, by extending a branch with a node labelled by the prefixed biterm in the denominator of R,

   provided that any prefixed biterms in the numerator of R already appear in the branch and any side-conditions of R are satisfied; or
   (b) there is a theorem η of a proof of rank $k-1$ and a branch is extended with the prefixed biterm $1\ \eta$; or

(c) there is a result η of a computation of rank $k-1$ and a branch is extended with the prefixed biterm $1\ \eta$.

3. Each branch of T_n contains nodes labelled by $\sigma\ \psi$ and $\sigma\ \neg\psi$, for some prefix σ and biterm ψ.

Each T_i is called a *tableau of rank k*.

A branch of a tableau of rank k is *closed* if it contains nodes labelled by $\sigma\ \psi$ and $\sigma\ \neg\psi$, for some prefix σ and biterm ψ; otherwise, the branch is *open*.

A tableau of rank k is *closed* if each branch is closed; otherwise, the tableau is *open*.

The biterm φ is called the *theorem* of the proof; this is denoted by $\mathcal{T} \vdash \varphi$.

Note that a computation of rank k is a computation of rank k' and a proof of rank k is a proof of rank k', for all $k' > k$.

Definition 6.8. Let $\mathcal{T}$ be a theory.

A *computation with respect to* $\mathcal{T}$ is a computation using $\Box_{j_1} \cdots \Box_{j_r}$ of rank k with respect to $\mathcal{T}$, for some $j_1 \ldots j_r$ and $k \geq 0$.

A *proof with respect to* $\mathcal{T}$ is a proof of rank k with respect to $\mathcal{T}$, for some $k \geq 0$.

Proposition 6.9. *Let $\mathcal{T}$ be a theory. Then the following hold.*

1. *The result of a computation with respect to $\mathcal{T}$ is a consequence of $\mathcal{T}$.*

2. *The theorem of a proof with respect to $\mathcal{T}$ is a consequence of $\mathcal{T}$.*

Proof. The proof is by induction on the rank of the computation and the rank of the proof.

If the rank of the computation is 0, then this is just Proposition 6.1, and if the rank of the proof is 0, then this is just Proposition 6.5.

Suppose next that the result holds for computations of rank $k-1$ and proofs of rank $k-1$. Consider first a computation of rank k. By the induction hypothesis, the input equation in Parts 2(c) and 2(d) of the definition of computation of rank k is consequence of $\mathcal{T}$. Using this fact, the proof that a computation of rank k is a consequence of $\mathcal{T}$ is now almost exactly the same as the proof of Proposition 6.1.

Finally, consider a proof of rank k. The first step is to establish that Proposition 6.4 can be extended to the Parts 2(b) and 2(c) in the definition of proof of rank k in which a branch is extended with a prefixed biterm $1\ \eta$, where η is either theorem of a proof of rank $k-1$ or a result of a computation of rank $k-1$. In either case, by the induction hypothesis, η is a consequence of $\mathcal{T}$, and so η is valid at $F(1)$ in I (where I is the interpretation in the proof of Proposition 6.4). Thus the tableau extended by a node labelled by $1\ \eta$ is satisfiable, which establishes Proposition 6.4 for these two extra cases. The remainder of the proof that a proof of rank k is a consequence of $\mathcal{T}$ now follows in a similar way to the proof of Proposition 6.5. □

In some applications, it is common for a proof to consist largely of a computation.

Proposition 6.10. *Let $\mathcal{T}$ be a theory, φ a biterm of type $\alpha_1 \to \cdots \to \alpha_n \to \Omega$, and $k \geq 0$.*

1. *Suppose there is a computation of rank k using $\Box_{j_1} \cdots \Box_{j_r}$ with respect to $\mathcal{T}$ of φ with result $\Box_{j_1} \cdots \Box_{j_r} \forall(\varphi = \lambda x_1 \ldots . \lambda x_n . \top)$. Then there is a proof of rank $k+1$ with respect to $\mathcal{T}$ of $\Box_{j_1} \cdots \Box_{j_r} \forall(\varphi)$.*

2. *Suppose there is a computation of rank k using $\Box_{j_1} \cdots \Box_{j_r}$ with respect to $\mathcal{T}$ of φ with result $\Box_{j_1} \cdots \Box_{j_r} \forall(\varphi = \lambda x_1. \ldots \lambda x_n.\bot)$. Then there is a proof of rank $k+1$ with respect to $\mathcal{T}$ of $\Box_{j_1} \cdots \Box_{j_r} \forall(\neg\varphi)$.*

Proof. 1. The proof of $\Box_{j_1} \cdots \Box_{j_r} \forall(\varphi)$ is given in Figure 30. Suppose that φ contains the free variables $y_1, \ldots, y_m$ $(m \geq 0)$. Then φ' denotes the biterm $\varphi\{y_1/z_1, \ldots, y_m/z_m\}$, where $z_1, \ldots, z_m$ are variables new to the branch of the proof. An explanation of this proof is as follows. Item 1 is the negation of the biterm to be proved; 2 is from 1 by a possibility rule; 3 is from 2 by possibility rules; 4 is from 3 by existential rules; 5 is a result from a computation of rank k; 6 is from 5 by a necessity rule; 7 is from 6 by necessity rules; 8 is from 7 by universal rules; 9 is from 4 and 8 by the substitutivity rule; now the branch closes by 9.

2. The proof of $\Box_{j_1} \cdots \Box_{j_r} \forall(\neg\varphi)$ is given in Figure 31. An explanation of this proof is as follows. Item 1 is the negation of the biterm to be proved; 2 is from 1 by a possibility rule; 3 is from 2 by possibility rules; 4 is from 3 by existential rules; 5 is from 4 by the double negation rule; 6 is a result from a computation of rank k; 7 is from 6 by a necessity rule; 8 is from 7 by necessity rules; 9 is from 8 by universal rules; 10 is from 5 and 9 by the substitutivity rule; now the branch closes by 10. □

1	$\neg\Box_{j_1} \cdots \Box_{j_r} \forall(\varphi)$	1.
1.1_{j_1}	$\neg\Box_{j_2} \cdots \Box_{j_r} \forall(\varphi)$	2.
	$\vdots$	
$1.1_{j_1} \ldots 1_{j_r}$	$\neg\forall(\varphi)$	3.
$1.1_{j_1} \ldots 1_{j_r}$	$\neg\varphi'$	4.
1	$\Box_{j_1} \cdots \Box_{j_r} \forall(\varphi = \lambda x_1. \ldots \lambda x_n.\top)$	5.
1.1_{j_1}	$\Box_{j_2} \cdots \Box_{j_r} \forall(\varphi = \lambda x_1. \ldots \lambda x_n.\top)$	6.
	$\vdots$	
$1.1_{j_1} \ldots 1_{j_r}$	$\forall(\varphi = \lambda x_1. \ldots \lambda x_n.\top)$	7.
$1.1_{j_1} \ldots 1_{j_r}$	$\varphi' = \lambda x_1. \ldots \lambda x_n.\top$	8.
$1.1_{j_1} \ldots 1_{j_r}$	$\neg\lambda x_1. \ldots \lambda x_n.\top$	9.

Figure 30: Proof of rank $k+1$ of $\Box_{j_1} \cdots \Box_{j_r} \forall(\varphi)$

The next result, which builds on Proposition 6.2, is used in the theoretical development of belief acquisition.

Proposition 6.11. *Let s and t be basic terms of the same type, $\mathcal{E}$ the standard equality theory, and $\Box_{j_1} \cdots \Box_{j_r}$ any sequence of modalities. Then the following hold.*

1. *If $s \overset{*}{\longleftrightarrow}_\alpha t$, then $\mathcal{E} \vdash \Box_{j_1} \cdots \Box_{j_r}(s = t)$.*

2. *If $s \not\overset{*}{\longleftrightarrow}_\alpha t$, then $\mathcal{E} \vdash \Box_{j_1} \cdots \Box_{j_r} \neg(s = t)$.*

Proof. 1. Suppose that $s \overset{*}{\longleftrightarrow}_\alpha t$. According to Proposition 6.2, there is a computation of rank 0 using $\Box_{j_1} \cdots \Box_{j_r}$ with respect to $\mathcal{E}$ for the goal $s = t$ that has the result $\Box_{j_1} \cdots \Box_{j_r}((s = t) = \top)$. Using this, the proof of $\Box_{j_1} \cdots \Box_{j_r}(s = t)$ is given in Figure 32. An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved;

1	$\neg\Box_{j_1}\cdots\Box_{j_r}\forall(\neg\varphi)$	1.
1.1_{j_1}	$\neg\Box_{j_2}\cdots\Box_{j_r}\forall(\neg\varphi)$	2.
	$\vdots$	
$1.1_{j_1}\ldots1_{j_r}$	$\neg\forall(\neg\varphi)$	3.
$1.1_{j_1}\ldots1_{j_r}$	$\neg\neg\varphi'$	4.
$1.1_{j_1}\ldots1_{j_r}$	φ'	5.
1	$\Box_{j_1}\cdots\Box_{j_r}\forall(\varphi = \lambda x_1.\ldots.\lambda x_n.\bot)$	6.
1.1_{j_1}	$\Box_{j_2}\cdots\Box_{j_r}\forall(\varphi = \lambda x_1.\ldots.\lambda x_n.\bot)$	7.
	$\vdots$	
$1.1_{j_1}\ldots1_{j_r}$	$\forall(\varphi = \lambda x_1.\ldots.\lambda x_n.\bot)$	8.
$1.1_{j_1}\ldots1_{j_r}$	$\varphi' = \lambda x_1.\ldots.\lambda x_n.\bot$	9.
$1.1_{j_1}\ldots1_{j_r}$	$\lambda x_1.\ldots.\lambda x_n.\bot$	10.

Figure 31: Proof of rank $k+1$ of $\Box_{j_1}\cdots\Box_{j_r}\forall(\neg\varphi)$

2 is the result from Proposition 6.2; 3 is from 1 by possibility rules; 4 is from 2 by necessity rules; 5 is from 3 and 4 by the substitutivity rule; the branch now closes by 5.

2. This proof is very similar to the proof of 1. □

1	$\neg\Box_{j_1}\cdots\Box_{j_r}(s = t)$	1.
1	$\Box_{j_1}\cdots\Box_{j_r}((s = t) = \top)$	2.
$1.1_{j_1}\ldots1_{j_r}$	$\neg(s = t)$	3.
$1.1_{j_1}\ldots1_{j_r}$	$((s = t) = \top)$	4.
$1.1_{j_1}\ldots1_{j_r}$	$\neg\top$	5.

Figure 32: Proof of rank 1 of $\Box_{j_1}\cdots\Box_{j_r}(s = t)$

Here is an example to illustrate the combination of computation and proof; more examples are given in Section 7.3 below.

Example 6.14. Consider an alphabet containing the functions

$$\begin{aligned}
&\mathit{setExists}_1 : (\alpha \to \Omega) \to \{\alpha\} \to \Omega\\
&\wedge_2 : (\alpha \to \Omega) \to (\alpha \to \Omega) \to \alpha \to \Omega\\
&\mathit{top} : \alpha \to \Omega\\
&\wedge : \Omega \to \Omega \to \Omega,
\end{aligned}$$

for some type α, and a theory $\mathfrak{T}$ that includes

$$\begin{aligned}
&(\mathit{setExists}_1\ p\ t) = \exists x.((p\ x) \wedge (x \in t))\\
&(\wedge_2\ p_1\ p_2\ x) = (p_1\ x) \wedge (p_2\ x)\\
&(\mathit{top}\ x) = \top\\
&x \wedge \top = x
\end{aligned}$$

as global assumptions.

Let p and q be specific predicates on items of type α, and consider the two predicates

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

and

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

I will show that

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

is a theorem of a proof of rank 1 with respect to $\mathcal{T}$ and, therefore, is a consequence of $\mathcal{T}$. Intuitively, this means that agent j believes that the first predicate is stronger than the second. The proof of this formula is given in Figure 33. In this proof, there are two subsidiary computations of rank 0: the computation of $(setExists_1\ (\wedge_2\ p\ q)\ y)$ is given in Figure 34 and the computation of $(setExists_1\ (\wedge_2\ p\ top)\ y)$ is given in Figure 35. Each computation uses the leftmost selection rule and has the redexes underlined.

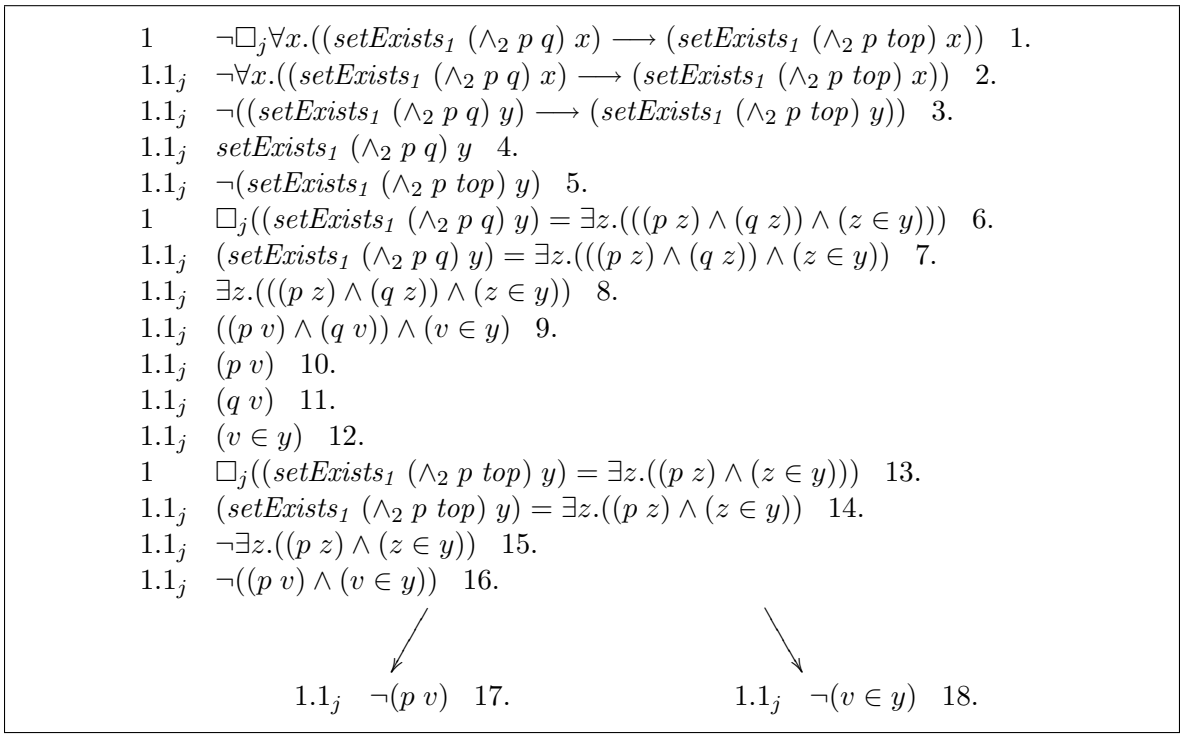

Figure 33: Proof of $\Box_j \forall x.((setExists_1\ (\wedge_2\ p\ q)\ x) \longrightarrow (setExists_1\ (\wedge_2\ p\ top)\ x))$

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 introduction of the result from Figure 34; 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 conjunctive rules; 13 is by introduction of the result from Figure 35; 14 is from 13 by a necessity rule; 15 is from 5 and 14 by the substitutivity rule; 16

$\underline{(setExists_1\ (\wedge_2\ p\ q)\ y)}$
$\exists z.(\underline{((\wedge_2\ p\ q)\ z)} \wedge (z \in y))$
$\exists z.(((p\ z) \wedge (q\ z)) \wedge (z \in y))$

Figure 34: Computation using $\Box_j$ of $(setExists_1\ (\wedge_2\ p\ q)\ y)$

$\underline{(setExists_1\ (\wedge_2\ p\ top)\ y)}$
$\exists z.(\underline{((\wedge_2\ p\ top)\ z)} \wedge (z \in y))$
$\exists z.(((p\ z) \wedge \underline{(top\ z)}) \wedge (z \in y))$
$\exists z.(\underline{((p\ z) \wedge \top)} \wedge (z \in y))$
$\exists z.((p\ z) \wedge (z \in y))$

Figure 35: Computation using $\Box_j$ of $(setExists_1\ (\wedge_2\ p\ top)\ y)$

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.

In Proposition 3.24, it was shown semantically that $f = \lambda x.t$ and $\forall x.((f\ x) = t)$ are equivalent. The next example shows this proof-theoretically.

Example 6.15. Figure 36 gives a proof of rank 1 of $(f = \lambda x.t) = \forall x.((f\ x) = t)$, where f is a function and t is a term. This proof uses the global assumption

$$\forall y.\forall z.((y = z) = \forall x.((y\ x) = (z\ x))),$$

which is the axiom of extensionality.

1 $\neg((f = \lambda x.t) = \forall x.((f\ x) = t))$ 1.
1 $\forall y.\forall z.((y = z) = \forall x.((y\ x) = (z\ x)))$ 2.
1 $(f = \lambda x.t) = \forall x.((f\ x) = (\lambda x.t\ x))$ 3.
1 $\forall x.((f\ x) = (\lambda x.t\ x)) = \forall x.((f\ x) = t)$ 4.
1 $(f = \lambda x.t) = \forall x.((f\ x) = t)$ 5.

Figure 36: Proof of rank 1 of $(f = \lambda x.t) = \forall x.((f\ x) = t)$

An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 is a global assumption; 3 is from 2 by a universal rule; 4 is by introduction of the result of a (one-step) computation; 5 is from 3 and 4 by the substitutivity rule; now the branch closes by 1 and 5.

The last example for this section illustrates reasoning with biterms.

Example 6.16. It is common to have to show that one standard predicate is stronger than another, because a standard predicate needs to be simplified or because the relationship needs to be established by a belief acquisition algorithm, for example.

Consider the standard predicates

$$\Box_2 f \circ \Box_1 p$$
$$\Box_2 f \circ \Box_1 q,$$

where $p : \sigma \to \Omega$ and $q : \sigma \to \Omega$ are defined by

$$\Box_1(p = r \wedge s \wedge t)$$
$$\Box_1(q = s \wedge t).$$

Figure 37 contains a proof that the first predicate is stronger than the second, that is, that

$$\Box_2 f \circ \Box_1 p \longrightarrow \Box_2 f \circ \Box_1 q$$

is a theorem. An explanation of this proof is as follows. Item 1 is the negation of the biterm to be proved; 2 and 3 are from 1 by a conjunctive rule; 4 is the result from the computation in Figure 38; 5 is from 2 and 4 by the substitutivity rule; 6 is the result from the computation in Figure 39; 7 is from 3 and 6 by the substitutivity rule; 8, 9, and 10 are from 5 by conjunctive rules; 11 and 12 are from 7 by a disjunctive rule; now the first branch closes by 9 and 11, and the second branch closes by 10 and 12.

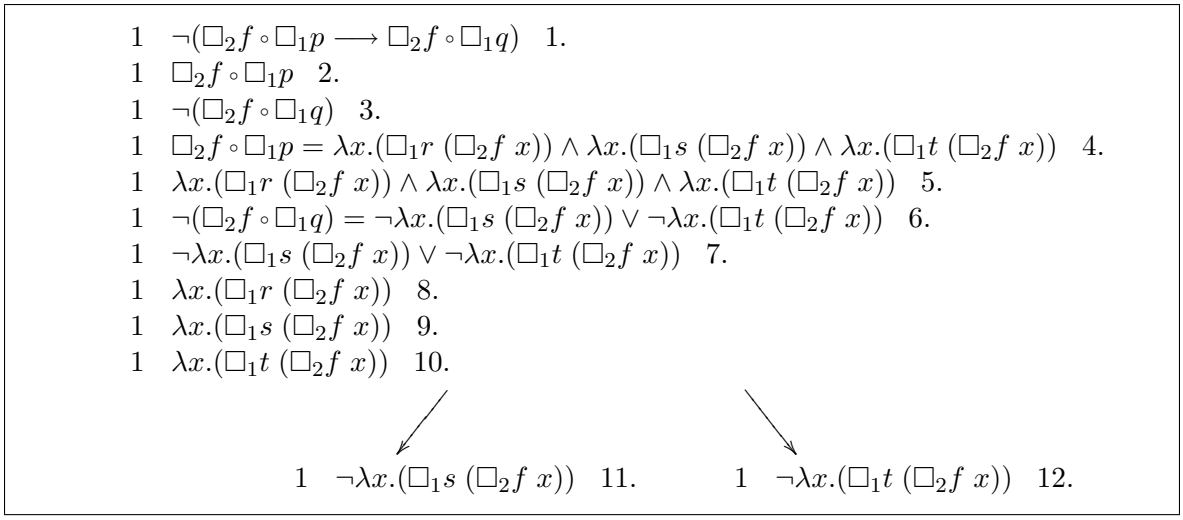

Figure 37: Proof of rank 1 of $\Box_2 f \circ \Box_1 p \longrightarrow \Box_2 f \circ \Box_1 q$

The computations in Figures 38 and 39 use (amongst others) equations having the following form.

$$f \circ g = \lambda x.(g\ (f\ x))$$
$$\Box_i(\varphi \wedge \psi) = \Box_i \varphi \wedge \Box_i \psi.$$

Actually, these two computations are not complete because the definitions for f, r, s and t have not been used. Clearly whether these definitions are used or not makes no essential difference to the proof, so the abbreviated computations correctly illustrate the ideas.

$\underline{\Box_2 f \circ \Box_1 p}$
$\lambda x.(\Box_1 \underline{p}\ (\Box_2 f\ x))$
$\lambda x.(\underline{\Box_1(r \wedge s \wedge t)}\ (\Box_2 f\ x))$
$\lambda x.(\underline{(\Box_1 r \wedge \Box_1 s \wedge \Box_1 t)\ (\Box_2 f\ x)})$
$\underline{\lambda x.((\Box_1 r\ (\Box_2 f\ x)) \wedge (\Box_1 s\ (\Box_2 f\ x)) \wedge (\Box_1 t\ (\Box_2 f\ x)))}$
$\lambda x.(\Box_1 r\ (\Box_2 f\ x)) \wedge \lambda x.(\Box_1 s\ (\Box_2 f\ x)) \wedge \lambda x.(\Box_1 t\ (\Box_2 f\ x))$

Figure 38: Computation of $\Box_2 f \circ \Box_1 p$

$\neg(\underline{\Box_2 f \circ \Box_1 q})$
$\neg\lambda x.(\Box_1 \underline{q}\ (\Box_2 f\ x))$
$\neg\lambda x.(\underline{\Box_1(s \wedge t)}\ (\Box_2 f\ x))$
$\neg\lambda x.(\underline{(\Box_1 s \wedge \Box_1 t)\ (\Box_2 f\ x)})$
$\neg\underline{\lambda x.((\Box_1 s\ (\Box_2 f\ x)) \wedge (\Box_1 t\ (\Box_2 f\ x)))}$
$\underline{\neg(\lambda x.(\Box_1 s\ (\Box_2 f\ x)) \wedge \lambda x.(\Box_1 t\ (\Box_2 f\ x)))}$
$\neg\lambda x.(\Box_1 s\ (\Box_2 f\ x)) \vee \neg\lambda x.(\Box_1 t\ (\Box_2 f\ x))$

Figure 39: Computation of $\neg(\Box_2 f \circ \Box_1 q)$

Example 6.16 shows the general approach to showing that one standard predicate is stronger than another: simplify each predicate using a computation and then give the results of the computations to the theorem prover.

6.4 Probabilistic Reasoning

In any significant application of computers, there is likely to be uncertainty of various kinds. Since the best way to treat uncertainty is to explicitly represent it and reason about it, this section contains a discussion of how uncertainty can be conveniently modelled and reasoned about in higher-order logic. This material is set in the context of the long-standing problem of integrating logic and probability, some references to which are given in Section 8. Appendix B gives some standard concepts from probability theory that will be needed.

Intuitively, the problem of integrating logic and probability is to find some way of effectively doing probabilistic reasoning in a logical formalism that may involve the invention of 'probabilistic logics'. So, for example, it is desirable that there be some way of representing the uncertainty of assumptions in a theory and also methods of computing the uncertainty of theorems that are proved from such assumptions. To set the scene, here is a discussion of some approaches to integrating logic and probability that have come recently from the machine learning community [Mug96, DK03, MMR+05, RD06, MR07, KD07].

The standard logical setting for these approaches is first-order logic. Imagine that an

agent is operating in some environment for which there is some uncertainty (for example, the environment might be partially observable). The environment is modelled as a probability distribution over the collection of first-order interpretations (over some suitable alphabet for the application at hand). The intuition is that any of these interpretations could be the actual environment but that some interpretations are more likely than others to correctly model the actual world and this information is given by the distribution on the interpretations. If the agent actually knew this distribution, then it could answer probabilistic questions of the form: if (closed) formula ψ holds, what is the probability that the (closed) formula φ holds? In symbols, the question is: what is $Pr(\varphi \,|\, \psi)$?

This situation is formalised as follows. Let $\mathfrak{I}$ be the set of interpretations and p a probability measure on the σ-algebra of all subsets of this set. Define the random variable $X_\varphi : \mathfrak{I} \to \mathbb{R}$ by

$$X_\varphi(I) = \begin{cases} 1 & \text{if } \varphi \text{ is true in } I \\ 0 & \text{otherwise,} \end{cases}$$

with a similar definition for X_ψ. Then $Pr(\varphi \,|\, \psi)$ can be written in the form

$$p(X_\varphi = 1 \,|\, X_\psi = 1)$$

which is equal to

$$\frac{p(X_\varphi = 1 \wedge X_\psi = 1)}{p(X_\psi = 1)}$$

and, knowing p, can be evaluated.

Of course, the real problem is to know the distribution on the interpretations. To make some progress on this, most systems intending to integrate logical and probabilistic reasoning in artificial intelligence make simplifying assumptions. For a start, most are based on Prolog. Thus theories are first-order Horn clause theories, maybe with negation as failure. Interpretations are limited to Herbrand interpretations and often function symbols are excluded so the Herbrand base (and therefore the number of Herbrand interpretations) is finite. Let $\mathfrak{I}$ denote the (finite) set of Herbrand interpretations and $\mathcal{B}$ the Herbrand base. One can identify $\mathfrak{I}$ with the product space $\{0,1\}^{\mathcal{B}}$ in the natural way. Thus the problem amounts to knowing the distribution on this product space. At this point, there is a wide divergence in the approaches. For example, either Bayesian networks or Markov random fields can be used to represent the product distribution. In [RD06], the occurrences of atoms in the same clause are used to give the arcs and the weights attached to clauses are used to give the potential functions in a Markov random field. In [KD07], conditional probability distributions are attached to clauses to give a Bayesian network. In [MMR$^+$05], a program is written that specifies a generative distribution for a Bayesian network. In all cases, the logic is exploited to give some kind of compact representation of what is usually a very large graphical model. Generally, the theory is only used to construct the graphical model and reasoning proceeds probabilistically, as described above. This completes the scene setting.

Here a different approach is followed [LN07b]. To begin with, a much more expressive logic, modal higher-order logic is used. The higher-orderness will be essential to achieve the desired integration of logic and probability. Also, the modalities will be important for agent applications. Furthermore, in the approach here, the theory plays a central role and probabilistic reasoning all takes place in the context of the theory.

To understand how this works, some preparation is necessary. Suppose an experiment is performed that randomly returns some element of a set Y (according to some distribution). This is modelled probabilistically by supposing that there is a σ-algebra $\mathcal{A}$ of measurable subsets (that is, events) of Y and a probability ν on $\mathcal{A}$ giving the probability space $(Y, \mathcal{A}, \nu)$. Then the probability that a randomly chosen element (that is, sample point) lies in a subset A of Y is given by $\nu(A)$.

Probabilities are not convenient to deal with in many applications (partly because the σ-algebra that is the domain of the probability would have to be made explicit); instead, it is easier to work with densities of probabilities. To set this up, suppose there is an underlying σ-finite measure μ on $\mathcal{A}$. Under a weak condition (the absolute continuity of ν with respect to μ), the Radon-Nikodym theorem (Proposition B.2) states that there exists a non-negative, integrable function h such that

$$\nu(A) = \int_A h \, d\mu,$$

for all $A \in \mathcal{A}$. Conversely, given such an h, the preceding equation defines a finite measure on $\mathcal{A}$ that is absolutely continuous with respect to μ.

There are two cases of particular interest for the underlying measure μ. In the first case, Y is a countable set and μ is the counting measure; this is the discrete case. In the second case, Y is $\mathbb{R}^n$ ($n \geq 1$) and μ is Lebesgue measure; this is the continuous case.

A density is a non-negative, integrable function h on a set Y such that $\int_Y h \, d\mu = 1$ (Definition B.8). In effect, in all the situations that are likely to be met in practice, the Radon-Nikodym theorem shows that one can work with densities instead of probabilities. Intuitively, the density for the probability shows where the 'mass' is most concentrated and, therefore, where a sample point is most likely to lie.

If $(Y, \mathcal{A}, \mu)$ is a measure space, then *Density* Y denotes the set of densities on $(Y, \mathcal{A}, \mu)$. Densities on a set are used to directly model the uncertainty in the identity of an element chosen according to some distribution from the set.

Example 6.17. Let Y be the set $\{\top, \bot\}$ and μ the counting measure on Y. Then $f : Y \to \mathbb{R}$ defined by

$$f(y) = \begin{cases} 0.4 & \text{if } y = \top \\ 0.6 & \text{if } y = \bot \end{cases}$$

is a density (with respect to μ) on Y. This is the density of a Bernoulli distribution [Was04].

Example 6.18. Let Y be the set $\{A, B, C, D\}$ and μ the counting measure on Y. Then $f : Y \to \mathbb{R}$ defined by

$$f(y) = \begin{cases} 0.6 & \text{if } y = A \\ 0.1 & \text{if } y = B \\ 0.2 & \text{if } y = C \\ 0.1 & \text{if } y = D \end{cases}$$

is a density (with respect to μ) on Y.

Other discrete densities include those for the Binomial, Geometric, and Poisson distributions [Was04].

Example 6.19. Let Y be $\mathbb{R}$ and μ the Lebesgue measure on Y. Then $f : Y \to \mathbb{R}$ defined by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2},$$

for $x \in \mathbb{R}$, is a density (with respect to μ) on $\mathbb{R}$. This is the density for the normal (or Gaussian) distribution with mean μ and variance σ^2.

Other continuous densities include those for the Exponential, Gamma, and Beta distributions [Was04].

Consider now a more complicated situation. Suppose that there is a function $f : X \to Y$ for which there is some uncertainty about its values that has to be modelled. This is done with a function

$$f' : X \to \mathit{Density}\ Y,$$

where, for each argument x, $(f'\ x)$ is a suitable density for modelling the uncertainty in the value of the function $(f\ x)$. Once again, the intuition is that the actual value of $(f\ x)$ is likely to be where the 'mass' of the density $(f'\ x)$ is most concentrated. While it is possible for the corresponding density to be different at each element x in X, typically X is divided into subsets (specified by predicates) on which the corresponding densities are identical.

Note that the case of uncertainty on a set that I started with is essentially a special case of uncertainty in the values of a function as in the preceding paragraph. Let $\star$ be a distinguished singleton set. (This is the domain for the type *1* in the logic.) Then each element y in the set Y can be identified with the function from $\star$ to Y that maps the (only) element of $\star$ to y. Thus one can recover the former case from the latter by putting X equal to $\star$.

Example 6.20. Consider a function f having domain Y and for which there is some uncertainty about its boolean values. This is modelled by giving f the signature $Y \to \mathit{Density}\ \{\top, \bot\}$. Then a typical definition for f might be as follows.

$$f(x) = \begin{cases} \lambda y.\mathit{if}\ y = \top\ \mathit{then}\ 0.1\ \mathit{else}\ 0.9 & \text{if } p_1(x) \text{ holds} \\ \lambda y.\mathit{if}\ y = \top\ \mathit{then}\ 0.7\ \mathit{else}\ 0.3 & \text{if } p_2(x) \text{ holds} \\ \lambda y.\mathit{if}\ y = \top\ \mathit{then}\ 0.8\ \mathit{else}\ 0.2 & \text{if } p_3(x) \text{ holds} \\ \lambda y.\mathit{if}\ y = \top\ \mathit{then}\ 0\ \mathit{else}\ 1 & \text{otherwise.} \end{cases}$$

Note how the predicates p_1, p_2, and p_3 divide up the domain into regions on each of which the function has the same density as a value – this is typical of such functions that are often learned.

Thus the key idea is to model uncertainty by using densities in the definitions of (some) functions in theories. This simple idea turns out to be a powerful and convenient way of modelling uncertainty with logical theories in diverse applications, especially agent applications. Note carefully the use that has been made of the higher-orderness of the logic; this approach is not available with first-order logic.

Now I turn to the formulation of these ideas in the logic. First, there is a type synonym

$$\mathit{Density}\ a \equiv a \to \mathit{Real}.$$

A type $\mathit{Density}\ \tau$ is the type of densities whose domain has type τ. Any term of type $\mathit{Density}\ \tau$, for some τ, is called a *density*.

Example 6.21. Consider again the situation of Example 6.20. Suppose the type of elements in Y is σ. Then, in the logic, the signature of f is $\sigma \to \mathit{Density}\ \Omega$. Formulated in the logic, the definition for f is as follows.

$$\begin{aligned}
&\forall x.((f\ x) = \\
&\quad \mathit{if}\ (p_1\ x)\ \mathit{then}\ \lambda y.\mathit{if}\ y = \top\ \mathit{then}\ 0.1\ \mathit{else}\ \mathit{if}\ y = \bot\ \mathit{then}\ 0.9\ \mathit{else}\ 0 \\
&\quad \mathit{else}\ \mathit{if}\ (p_2\ x)\ \mathit{then}\ \lambda y.\mathit{if}\ y = \top\ \mathit{then}\ 0.7\ \mathit{else}\ \mathit{if}\ y = \bot\ \mathit{then}\ 0.3\ \mathit{else}\ 0 \\
&\quad \mathit{else}\ \mathit{if}\ (p_3\ x)\ \mathit{then}\ \lambda y.\mathit{if}\ y = \top\ \mathit{then}\ 0.8\ \mathit{else}\ \mathit{if}\ y = \bot\ \mathit{then}\ 0.2\ \mathit{else}\ 0 \\
&\quad \mathit{else}\ \lambda y.\mathit{if}\ y = \top\ \mathit{then}\ 0\ \mathit{else}\ \mathit{if}\ y = \bot\ \mathit{then}\ 1\ \mathit{else}\ 0).
\end{aligned}$$

It will be convenient because of the operations that will be applied to *if_then_else* expressions to be a little pedantic about how they are written by explicitly having an '*else* 0' case at the end.

The next task is to express the definitions of the functions ♮, ♭, §, and \$ from Appendix B in the logic. For this, I confine attention in the following to the discrete case for which densities have finite support (that is, take a non-zero value on at most finitely many elements of their domain). For this case, several summation functions whose arguments are functions with finite support are needed. (In the discrete case when the support is not finite, summation functions are needed that suitably approximate infinite sums. In the continuous case, functions that do (approximate) integration are needed. In both these cases, densities are likely to be standard ones such as the Poisson, Gaussian, and so on, for which special-purpose functions for performing the necessary computations can be written.)

First comes the function *sum1* that computes $\sum_x f(x)$, where f is a real-valued function having finite support (that is, takes a non-zero value on at most finitely many elements of its domain) and x ranges over the domain of f.

$$\mathit{sum1} : (a \to \mathit{Real}) \to \mathit{Real}$$

$$(\mathit{sum1}\ \lambda x.\mathit{if}\ x = u\ \mathit{then}\ v\ \mathit{else}\ \boldsymbol{w}) = v + (\mathit{sum1}\ \lambda x.\boldsymbol{w})$$

$$(\mathit{sum1}\ \lambda x.0) = 0$$

Here it is assumed that the argument to *sum1* has the syntactic form of an abstraction over a nested *if_then_else* for which there are no repeated occurrences of the tests $x = u$. If there are such repeats, it is easy enough to remove them.

Example 6.22. Figure 40 gives the computation of

$$(\mathit{sum1}\ \lambda y.\mathit{if}\ y = \top\ \mathit{then}\ 0.1\ \mathit{else}\ \mathit{if}\ y = \bot\ \mathit{then}\ 0.5\ \mathit{else}\ 0),$$

which has the answer 0.6.

Also needed will be a function *sum2* that computes $\sum_{x \in S} f(x)$, where f is a real-valued function having finite support and S is a subset of the domain of f.

$$\mathit{sum2} : \{a\} \to (a \to \mathit{Real}) \to \mathit{Real}$$

$$(\mathit{sum2}\ s\ \lambda x.\mathit{if}\ x = u\ \mathit{then}\ v\ \mathit{else}\ \boldsymbol{w}) = (\mathit{if}\ u \in s\ \mathit{then}\ v\ \mathit{else}\ 0) + (\mathit{sum2}\ s\ \lambda x.\boldsymbol{w})$$

$$(\mathit{sum2}\ s\ \lambda x.0) = 0$$

Note that *sum1* and *sum2* are closely related in that

$$(\mathit{sum2}\ \lambda y.\top\ f) = (\mathit{sum1}\ f).$$

$\underline{(sum1\ \lambda y.if\ y = \top\ then\ 0.1\ else\ if\ y = \bot\ then\ 0.5\ else\ 0)}$
$0.1 + \underline{(sum1\ \lambda y.if\ y = \bot\ then\ 0.5\ else\ 0)}$
$0.1 + (0.5 + \underline{(sum1\ \lambda y.0)})$
$0.1 + (\underline{0.5 + 0})$
$\underline{0.1 + 0.5}$
0.6

Figure 40: Computation of $(sum1\ \lambda y.if\ y = \top\ then\ 0.1\ else\ if\ y = \bot\ then\ 0.5\ else\ 0)$

Example 6.23. Figure 41 gives the computation of

$(sum2\ \lambda x.(x = \bot)\ \lambda y.if\ y = \top\ then\ 0.1\ else\ if\ y = \bot\ then\ 0.5\ else\ 0)$,

which has the answer 0.5.

$\underline{(sum2\ \lambda x.(x = \bot)\ \lambda y.if\ y = \top\ then\ 0.1\ else\ if\ y = \bot\ then\ 0.5\ else\ 0)}$
$if\ \underline{\top \in \lambda x.(x = \bot)}\ then\ 0.1\ else\ 0 + (sum2\ \lambda x.(x = \bot)\ \lambda y.if\ y = \bot\ then\ 0.5\ else\ 0)$
$if\ \underline{\top = \bot}\ then\ 0.1\ else\ 0 + (sum2\ \lambda x.(x = \bot)\ \lambda y.if\ y = \bot\ then\ 0.5\ else\ 0)$
$\underline{if\ \bot\ then\ 0.1\ else\ 0} + (sum2\ \lambda x.(x = \bot)\ \lambda y.if\ y = \bot\ then\ 0.5\ else\ 0)$
$\underline{0 + (sum2\ \lambda x.(x = \bot)\ \lambda y.if\ y = \bot\ then\ 0.5\ else\ 0)}$
$\underline{(sum2\ \lambda x.(x = \bot)\ \lambda y.if\ y = \bot\ then\ 0.5\ else\ 0)}$
$\vdots$
$0.5 + \underline{(sum2\ \lambda y.0)}$
$\underline{0.5 + 0}$
0.5

Figure 41: Computation of $(sum2\ \lambda x.(x = \bot)\ \lambda y.if\ y = \top\ then\ 0.1\ else\ if\ y = \bot\ then\ 0.5\ else\ 0)$

Now the definitions of the following four functions for the discrete case can be defined in the logic.

$\natural : (a \to Density\ b) \to (b \to Density\ c) \to (a \to Density\ c)$
$\flat : (a \to Density\ b) \to (b \to c) \to (a \to Density\ c)$
$\S : Density\ b \to (b \to Density\ c) \to Density\ c$
$\$: Density\ b \to (b \to c) \to Density\ c$

The intuitive meaning of the function $\natural$ is that it is the 'composition' of two functions each of which has uncertainty in its values giving a 'composite' function that has the accumulated

uncertainty of these two functions in its values; the other functions have similar meanings that are essentially special cases of this.

Here are the definitions in the logic of these functions for the discrete case that correspond to the mathematical definitions in Appendix B.

$$\natural : (a \to \mathit{Density}\ b) \to (b \to \mathit{Density}\ c) \to (a \to \mathit{Density}\ c)$$
$$f \natural g = \lambda x.\lambda z.(\mathit{sum1}\ \lambda y.(((f\ x)\ y) \times ((g\ y)\ z)))$$

$$\flat : (a \to \mathit{Density}\ b) \to (b \to c) \to (a \to \mathit{Density}\ c)$$
$$f \flat g = \lambda x.\lambda z.(\mathit{sum2}\ \lambda y.((g\ y) = z)\ \lambda y.((f\ x)\ y))$$

$$\S : \mathit{Density}\ b \to (b \to \mathit{Density}\ c) \to \mathit{Density}\ c$$
$$f \S g = \lambda z.(\mathit{sum1}\ \lambda y.((f\ y) \times ((g\ y)\ z)))$$

$$\$: \mathit{Density}\ b \to (b \to c) \to \mathit{Density}\ c$$
$$f \$ g = \lambda z.(\mathit{sum2}\ \lambda y.((g\ y) = z)\ f)$$

An extended example of probabilistic reasoning using these functions is given in Example 7.8.

Conditionalisation and marginalisation (see Appendix B) are also easily expressed in the logic using the summation functions introduced (for the discrete case with finite support).

The literature on integrating logic and probability contains a variety of probabilistic logics. I now examine the syntactic features of two of these logics and show how they can be easily expressed using higher-order logic. In the logic of [FH94], one can write expressions such as $w(\varphi) < 1/3$ and $w(\varphi) \geq 2w(\psi)$, where the first means that "φ has probability less than 1/3" and the second means "φ is at least twice as probable as ψ". To write such expressions in higher-order logic, one replaces the formula φ in the logic of [FH94] by a density on the booleans denoted by φ'. (Some of) the function symbols in φ would have definitions involving densities. Then $w(\varphi) < 1/3$ is written in the logic of this paper as

$$(\varphi'\ \top) < 1/3$$

and $w(\varphi) \geq 2w(\psi)$ is written as

$$(\varphi'\ \top) \geq 2(\psi'\ \top).$$

The notation in [FH94] is extended so that one can write $w_i(\varphi) < 1/3$ with the meaning "according to agent i, φ holds with probability less than 1/3". In the logic of this paper, this can be expressed by

$$(\boldsymbol{B}_i\varphi'\ \top) < 1/3$$

or, equivalently by Equation 3.3 in Section 3.8 since $\top$ is rigid,

$$\boldsymbol{B}_i(\varphi'\ \top) < 1/3.$$

(An alternative to using $\boldsymbol{B}_i$ in the preceding two formulas is to use $\boldsymbol{K}_i$.) By the way, note carefully the difference in meaning between $\boldsymbol{B}_i(\varphi'\ \top) < 1/3$ and

$$\boldsymbol{B}_i((\varphi'\ \top) < 1/3).$$

Easy examples show that there are interpretations for which the first formula is true whereas the second is false. Even formulas such as $a_1 w_i(\varphi_1) + \cdots + a_k w_i(\varphi_k) \geq b$, where $a_1, \ldots, a_k, b$ are numbers, are admitted in [FH94]. This can be written as

$$a_1 \boldsymbol{B}_i(\varphi'_1\ \top) + \cdots + a_k \boldsymbol{B}_i(\varphi'_k\ \top) \geq b$$

in the logic of this paper.

In the logic of [Hal89], expressions such as $w_x(\varphi) \geq 1/2$ are admitted, where this expression has the meaning that "the probability that a randomly chosen x in the domain satisfies φ is greater than or equal to 1/2". Let h be the relevant density (with finite support) and S the set $\{x \mid \varphi\}$, Then, mathematically, the required expression is $\sum_{x \in S} h(x) \geq 1/2$, where $\sum_{x \in S} h(x)$ is the measure of the set of x that satisfy φ. In the logic of this paper, this is expressed as

$$(\mathit{sum2}\ \{x \mid \varphi\}\ h) \geq 1/2.$$

6.5 Decidability and Termination

I now turn to the related issues of the decidability of the proof system and termination of the computation system.

First, I discuss the proof system. Many *propositional* sublogics of the logic considered here are decidable; for example, the fusion logics $\mathbf{K}_m$, $\mathbf{T}_m$, $\mathbf{K4}_m$, $\mathbf{S4}_m$, $\mathbf{S5}_m$, and $\mathbf{KD45}_m$ ($m \geq 1$) are all decidable [FHMV95]. For such logics, interest naturally turns to the complexity of their decision procedure. For example, the satisfiability problems of $\mathbf{S5}_m$ and $\mathbf{KD45}_m$ ($m \geq 2$) are PSPACE-complete; thus the corresponding decision problems are also PSPACE-complete [FHMV95].

However, even propositional modal logics can be undecidable; for example, product logics of dimension three or more are often undecidable [GKWZ03]. In any case, for the class of intended applications, the higher-order (or, at least, first-order) versions of the logic are needed, and these are undecidable. As Church proved, 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. 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.

So how does one cope with the undecidability of the decision problem of the logic? The answer is that for each application the theorem-proving tasks that will arise in that application

have to be examined and shown to terminate (or else the application has to be redesigned and re-implemented in order to achieve this). What I have in mind here is that each application is a substantial system and that it makes sense to expend effort on ensuring the desired termination, in the same way as it makes sense to use conventional software engineering techniques to ensure the correctness of the software that implements an application. Furthermore, this suggestion is feasible: for a particular application the range of theorem-proving tasks (even for an adaptive system) is likely to be tightly circumscribed, so it should be possible to provide termination proofs for the theorem-proving tasks that arise.

Some design decisions help to ease the problem of undecidability. To begin with, I avoid (full and, therefore, undecidable) higher-order unification altogether for a decidable form of matching that is quite powerful enough for most applications. Also, experience with a variety of applications indicates that the crucial facility offered by the logic is computation, as in Section 6.1. That is, the most common reasoning task is to evaluate a function given some arguments; in many cases, the use of the theorem prover is subsidiary to the computation and is sometimes even essentially propositional and decidable.

This leads to the issue of the termination of the computation component of the reasoning system. The computation component in effect provides a declarative programming language. Even though the logic is modal, programming in this language is no more complicated than programming in, say, Haskell. Programs have to be written, modal or not, and the usual concerns of writing programs that efficiently compute what is needed and, especially, avoid infinite loops are much the same as for less expressive languages. So the answer to the problem of non-termination is that, for a particular application, the programming tasks that will arise in that application have to be analysed and shown to terminate (or else the application has to be redesigned and re-implemented in order to achieve this). There are well-established techniques for doing this such as the use of well-founded orderings [DM79].

In summary, for logics as expressive as the one proposed in this paper there is no hope for *general* decidability and termination results. Instead, for each application, it is necessary to show that the proof and computation tasks that arise in the application do indeed terminate; in the case where a proof or computation is intrinsically potentially non-terminating, resource-bounded techniques have to be brought to bear.

7 Agents

Multi-modal, polymorphically-typed, higher-order logic is highly expressive, actually much more expressive than most logics that are used in applications, so it is not surprising that the range of applications of the logic presented here is substantial. The expressive power of the logic makes it extremely attractive for applications; generally, one can accurately model the application in a language that is intuitive and comprehensible.

While a whole array of applications does come within the purview of the logic presented here, the particular class of agent applications strongly motivated the development and largely explains the choice of topics in the paper. In the next section, a brief overview of architectures for rational agents that can exploit the expressive power of the logic of this paper is given. In Section 7.2, particular syntactic forms for beliefs are discussed. In Section 7.3, applications of these ideas to agents are considered.

7.1 Architectures for Rational Agents

This subsection provides an overview of architectures for rational agents.

The first issue to address is what exactly it might mean for an agent to be rational. Here is a widely accepted definition of rationality [RN02, p.36]:

> "For each possible percept sequence, a rational agent should select an action that is expected to maximise its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has".

The understanding is that the performance measure is external to the agent and some method of informing the agent of how well it is doing is available; the details of how this is handled will depend on the application. What follows is an outline of architectures for agents that are rational in the above sense.

Consider an agent situated in some environment that can receive percepts from the environment and can apply actions that generally have a non-deterministic effect on the environment. Thus the agent can be considered to be a function from the sequence of percepts so far to actions; an agent architecture provides the definition of this function.

The dynamics of agents can be modelled in a standard way [RN02, Section 17.4]. Suppose that the agent maintains some (internal) state that records information about the environment to help it choose suitable actions. In general, the environment is only partially observable so that information recorded about it in the state is incomplete and uncertain; this is modelled by the agent maintaining a *density* on the states that the environment could be in. In each cycle, the agent receives a percept and updates the state density according to the information provided by the percept; using the updated state density, it then selects an action using some rationality principle; finally, it applies the action to the environment and updates the state density according to the action selected. Summarising the description so far, here is a high-level view of the main loop of the agent algorithm.

```
loop forever
  get percept
  select action
  put action.
```

The percept received by the agent provides information about two aspects of the environment. The first is information that refines what effect the last selected action had on the environment. The second is information about changes that may have occurred in the environment other than those that resulted from that action; for example, the environment may change autonomously irrespective of the behaviour of the agent. Both kinds of information are combined into a single percept.

This description is formalised as follows. Let S be the set of states of the agent, A the set of actions that the agent can apply, and P the set of percepts that the agent can receive. The set S is the underlying set of a measure space that has a σ-algebra $\mathcal{A}$ and a measure μ on $\mathcal{A}$. Often $(S, \mathcal{A}, \mu)$ is discrete so that $\mathcal{A}$ is the powerset of S and μ is the counting measure. The sensor model for the agent is given by a function

$$observe : S \to Density\ P.$$

If the environment is in a certain state, then the sensor model gives a distribution over the percepts that the agent could receive. Also the dynamics of the agent is captured by a function

$$\mathit{transition} : A \to S \to \mathit{Density}\ S.$$

In the discrete case, $\mathit{transition}(a)(s)(s')$ is the conditional probability that, given the state is s and action a is applied, there will be a transition to state s'. I now look in some detail at each step of the algorithm.

First comes the *get percept* step. If the current state density is $\boldsymbol{s}$ and percept p is received, then the next state density is

$$\lambda s.\alpha \boldsymbol{s}(s) \times \mathit{observe}(s)(p)$$

where α is a normalising constant, using Bayes rule.

Next comes the *select action* step. Various specific rationality principles could be used; a widely used one is the principle of maximum expected utility [RN02, p.585] (namely, a rational agent should choose an action that maximises the agent's expected utility, where the utility is a real-valued function on the set of states). If this principle is used, it is assumed that the utility function is known to the agent and that the maximum expected value of the utility corresponds closely to the external performance measure. Under the principle of maximum expected utility, the policy function

$$\mathit{policy} : \mathit{Density}\ S \to A$$

is defined by

$$\mathit{policy}(\boldsymbol{s}) = \operatorname*{argmax}_{a \in A}\ \mathbb{E}_{\boldsymbol{s}\ \S\ \mathit{transition}(a)}(\mathit{utility}),$$

for each $s \in S$. Here $\mathbb{E}_{\boldsymbol{s}\ \S\ \mathit{transition}(a)}(\mathit{utility})$ denotes the expectation of the random variable $\mathit{utility} : S \to \mathbb{R}$ with respect to the density $\boldsymbol{s}\ \S\ \mathit{transition}(a)$. This density is defined by

$$(\boldsymbol{s}\ \S\ \mathit{transition}(a))(s') = \int_S \boldsymbol{s}(s) \times \mathit{transition}(a)(s)(s')\ ds,$$

for each $s' \in S$. If the current state density is $\boldsymbol{s}$, then the action selected is thus the one given by $\mathit{policy}(\boldsymbol{s})$.

Finally, there is the *put action* step. One part of this is applying the action to the environment. The other part is updating the state density as a result of applying the action. Suppose that the current state density is $\boldsymbol{s}$ and the action selected is a_{max}. Then the next state density is $\boldsymbol{s}\ \S\ \mathit{transition}(a_{max})$.

Sometimes it is not feasible to use a utility function because it is too difficult to specify precisely enough what should be the utility of each state. In such cases, it is common to give the agent a reward (positive or negative) after each action [Put94, RN02]. This is a kind of 'local' utility; the 'global' utility employed above can be defined in terms of rewards and generally needs to be learned by the agent. Since this aspect of agent architectures is not relevant for the material of this paper, it is not discussed further.

In many complex applications, the agent is not given the definition of the function *transition* (or cannot be given this definition because it is impractical to specify it precisely enough); in such cases, the agent essentially has to learn the transition function from its

experience in the environment. A common complication then is that the number of states may be very large, so large in fact that it is quite impractical to attempt to learn *directly* the definition of the function *transition*. Instead, an obvious idea is to partition the set of states into a much smaller number of subsets of states such that the states in each subset can be treated uniformly and learn a transition function over equivalence classes of states. I now explore this idea in some detail and, in the subsequent two subsections, show how the expressivity of the logic can be exploited to turn this idea into a practical method that assists in the construction of agents.

It will be important to employ two different ways of partitioning the states. For this reason, two collections of functions on states are introduced. These are

$$e_i : S \to V_i,$$

where $i = 1, \ldots, n$, and

$$r_j : S \to W_j,$$

where $j = 1, \ldots, m$. Each function e_i, for $i = 1, \ldots, n$, is called an *evidence feature* and each function r_j, for $j = 1, \ldots, m$, is called a *result feature*. Each e_i and r_j is a feature that picks out a specific property of a state that is relevant to selecting actions. Evidence features are so-called because they pick out properties of the state that suggest the action that ought to be selected. Result features are so-called because they pick out properties of the state which results from applying an action that can be usefully employed in the calculation of its utility. An interesting fact that emerges from applying these ideas to practical applications is how different the evidence features are compared with the result features [LS06].

Typically, the cardinality of the product spaces $V_1 \times \cdots \times V_n$ and $W_1 \times \cdots \times W_m$ are much smaller than the cardinality of S. Often, some V_i and W_j are the simply the set of booleans. In an application that is at least partly continuous, they could be $\mathbb{R}$. There are two key functions associated with this construction. The function

$$(e_1, \ldots, e_n) : S \to V_1 \times \cdots \times V_n$$

is defined by

$$(e_1, \ldots, e_n)(s) = (e_1(s), \ldots, e_n(s)),$$

for each $s \in S$. The space $V_1 \times \cdots \times V_n$ is assumed to have a suitable σ-algebra of measurable sets on it, so that $(e_1, \ldots, e_n)$ is a measurable function. If μ is the measure on S, then $(e_1, \ldots, e_n)^{-1} \circ \mu$ is the measure imposed on $V_1 \times \cdots \times V_n$. (Note that $(f \circ g)(x)$ means $g(f(x))$.) Similarly, the function

$$(r_1, \ldots, r_m) : S \to W_1 \times \cdots \times W_m$$

is defined by

$$(r_1, \ldots, r_m)(s) = (r_1(s), \ldots, r_m(s)),$$

for each $s \in S$. The space $W_1 \times \cdots \times W_m$ is also assumed to have a suitable σ-algebra of measurable sets on it, so that $(r_1, \ldots, r_m)$ is a measurable function. Similarly, $(r_1, \ldots, r_m)^{-1} \circ \mu$ is the measure imposed on $W_1 \times \cdots \times W_m$.

Now, instead of the transition function

$$\mathit{transition} : A \to S \to \mathit{Density}\ S,$$

one works with a function

$$\mathit{transition}' : A \to V_1 \times \cdots \times V_n \to \mathit{Density}\ W_1 \times \cdots \times W_m.$$

The motivation for doing this is that $\mathit{transition}'$ should be more amenable to being learned because of the much smaller cardinalities of its domain and codomain. Some precision is lost in working with $V_1 \times \cdots \times V_n$ and $W_1 \times \cdots \times W_m$ instead of S in this way, as discussed below; to make up for this, informative choices of the evidence and result features need to be made. The agent cycle is now reconsidered using these features.

The three key functions have signatures

$$\begin{aligned}
&\mathit{observe} : S \ \to \mathit{Density}\ P\\
&\mathit{transition}' : A \to V_1 \times \cdots \times V_n \to \mathit{Density}\ W_1 \times \cdots \times W_m\\
&\mathit{policy} : \mathit{Density}\ S \ \to A.
\end{aligned}$$

The function *observe* is the same as before, as is the update of the state density after receiving the percept.

The policy is now defined by

$$\mathit{policy}(\boldsymbol{s}) = \operatorname*{argmax}_{a \in A} \mathbb{E}_{\boldsymbol{s}\,\S\,(e_1,\ldots,e_n)\,\circ\,\mathit{transition}'(a)}(\mathit{utility}'),$$

for each $s \in S$. Here

$$\mathit{utility}' : W_1 \times \cdots \times W_m \to \mathbb{R}.$$

The result features are intended to be chosen so that $(r_1, \ldots, r_m) \circ \mathit{utility}'$ gives the utility of each state. Note that

$$(e_1, \ldots, e_n) \circ \mathit{transition}'(a) : S \to \mathit{Density}\ W_1 \times \cdots \times W_m,$$

so that $\boldsymbol{s}\,\S\,(e_1, \ldots, e_n) \circ \mathit{transition}'(a)$ is a density on $W_1 \times \cdots \times W_m$.

If the action selected by the policy is a_{max}, then the state density that results by applying this action is

$$\boldsymbol{s}\,\S\,((e_1, \ldots, e_n) \circ \mathit{transition}'(a_{max}) \circ \overline{(r_1, \ldots, r_m)}).$$

Here, since $(r_1, \ldots, r_m) : S \to W_1 \times \cdots \times W_m$, it follows from a result in Appendix B that one can define the function

$$\overline{(r_1, \ldots, r_m)} : \mathit{Density}\ W_1 \times \cdots \times W_m \to \mathit{Density}\ S$$

by

$$\overline{(r_1, \ldots, r_m)}(h) = (r_1, \ldots, r_m) \circ h,$$

for each $h \in \mathit{Density}\ W_1 \times \cdots \times W_m$. Thus

$$(e_1, \ldots, e_n) \circ \mathit{transition}'(a_{max}) \circ \overline{(r_1, \ldots, r_m)} : S \to \mathit{Density}\ S$$

and so

$$\boldsymbol{s} \S ((e_1, \ldots, e_n) \circ \mathit{transition}'(a_{max}) \circ \overline{(r_1, \ldots, r_m)}) : \mathit{Density}\ S,$$

as required. This completes the description of the agent cycle modified to handle features.

At this point, the advantages and disadvantages of introducing features are clearer. The main advantage is that they can make learning the transition function feasible when otherwise it wouldn't be because of the vast size of the state space. The main disadvantage is that some precision in the update of the state density during the agent cycle is lost since this is now mediated by passing through $V_1 \times \cdots \times V_n$ and $W_1 \times \cdots \times W_m$. This shows the crucial importance of finding suitable features; if these can be found, and experience has shown that this can be hard for some applications, then the loss of precision is likely to be small and there is everything to gain. Choosing the 'right' (general form of) features is generally a problem that has to be solved before deployment by the designer of the agent, although some techniques are known that allow agents to discover such information autonomously. Agents can learn the precise definition of a feature (whose general form is given by an hypothesis language) during deployment by machine learning techniques. This eases the task of the designer since only the general forms of the features need specifying before deployment. In a dynamic environment, this adaptive capability of an agent is essential, of course. A further complication is that, if the definitions of some features change from time to time, then the process that is learning the transition function must take this into account because earlier training examples may need some adjustment.

The ideas of this subsection can now be connected back to the main theme of this paper. As the preceding discussion has made clear, the success of the whole enterprise of using features depends upon them being able to capture quite subtle properties of the environment. For example, it is likely that the features will need to capture beliefs about the beliefs of other agents, beliefs about temporal aspects of the environment, and will have to cope with uncertainty. The logic of this paper is ideal for the representation of such properties. This topic is taken up in the next two subsections.

7.2 Agent Beliefs

This subsection gives some detail about the syntactic form that beliefs might take [LN07c, LN07b].

The term 'belief' has two meanings in artificial intelligence: in robotics and vision [TBF05], a 'belief' is generally a probability distribution; in logical artificial intelligence, a 'belief' is a logical formula. I now give a definition of belief that encompasses both meanings and investigate the use of this concept for agent applications.

There are no generally agreed forms for beliefs in the literature, other than the basic requirement that they be formulas. For the purpose of constructing multi-agent systems, the following definition is proposed.

Definition 7.1. A *belief* is the definition of a function $f : \sigma \to \tau$ having the form

$$\Box \forall x.((f\ x) = t),$$

where $\Box$ is a (possibly empty) sequence of modalities and t is a term of type τ.

The function f thus defined is called a *belief function*. In case τ has the form *Density* ν, for some ν, we say the belief is *probabilistic*.

A *belief base* is a set of beliefs.

Typically, for agent j, beliefs have the form $\boldsymbol{B}_j\varphi$, with the intuitive meaning 'agent j believes φ', where φ is $\forall x.((f\ x) = t)$. Other typical beliefs have the form $\boldsymbol{B}_j\boldsymbol{B}_i\varphi$, meaning 'agent j believes that agent i believes φ'. If there is a temporal component to beliefs, this is often manifested by temporal modalities at the front of beliefs. Then, for example, there could be a belief of the form $\bullet^2\boldsymbol{B}_j\boldsymbol{B}_i\varphi$, whose intuitive meaning is 'at the second last time, agent j believed that agent i believed φ'. (Here, $\bullet^2$ is a shorthand for $\bullet\bullet$.)

The above description generalises the agent architecture presented in [LS06] by admitting probabilistic beliefs in addition to non-probabilistic ones. Using a density to model the uncertain value of a function on some argument is better than using a single value (such as the mean of the density). For example, if the density is a normal distribution, then it may be important that the variance is large or small: if it is large, intuitively, there is less confidence about its actual value; if it is small, then there could be confidence that the actual value is the mean. Such subtleties can assist in the selection of one action over another. Similarly, learning tasks can exploit the existence of the density by including features based on the mean, variance, higher moments, or other parameters of the density in hypothesis languages.

The form that beliefs can take is now examined in more detail. Some beliefs can be specified directly by the programmer and the body of the definition can be any term of the appropriate type. In particular, some of these beliefs may be compositions of other probabilistic beliefs in which case the composition operators introduced in Section 6.4 will be useful. Some beliefs, however, need to be acquired from training examples, usually during deployment. A particular form for beliefs of this latter type is proposed. I consider beliefs that, for a function $f : \sigma \to \tau$, are definitions of the following form.

$$
\begin{aligned}
&\Box\forall x.((f\ x) = \\
&\qquad \mathit{if}\ (p_1\ x)\ \mathit{then}\ v_1 \\
&\qquad \mathit{else\ if}\ (p_2\ x)\ \mathit{then}\ v_2 \\
&\qquad\quad \vdots \\
&\qquad \mathit{else\ if}\ (p_n\ x)\ \mathit{then}\ v_n \\
&\qquad \mathit{else}\ v_0),
\end{aligned}
\tag{7.1}
$$

where $\Box$ is a (possibly empty) sequence of modalities, $p_1, \ldots, p_n$ are predicates that can be modal and/or higher order, and $v_0, v_1, \ldots, v_n$ are suitable values. Such a belief is a definition for the function f in the context of the modal sequence $\Box$. Note that in the case when τ has the form *Density* ν, for some ν, the values $v_0, v_1, \ldots, v_n$ are densities.

While the above form for acquired beliefs may appear to be rather specialised, it turns out to be convenient and general, and easily encompasses beliefs in many other forms [Riv87, EIM02]. Also the decision-list form of the definitions is highly convenient for acquisition using some kind of learning algorithm [LN07a, Llo03, Ng05a].

Now consider this question: what makes up the belief base of an agent? Clearly the most crucial belief is the definition of the function *policy* that an agent uses to select actions. Working backwards from this, the belief base should also contain the definitions of the functions *observe*, *transition′* and *utility′*, the definitions of the evidence and result features, and the definitions of any subsidiary functions needed for the preceding definitions. These are the *only* beliefs the agent needs to maintain in order to act rationally. This fact provides strong guidance during the design of the agent.

7.3 Applications to Agents

From the point of view of this paper, what is most relevant in the preceding discussion about agent architectures is the collection of evidence and result features that are defined using the logic. Several examples that illustrate this are now given. Attention is focussed on evidence features; examples of result features can be found in [LS06].

Example 7.1. This example concerns an infotainment agent, which is a multi-agent system that contains a number of agents with functionalities for recommending movies, TV programs, music and the like, as well as information agents with functionalities for searching for information on the Internet. Here I concentrate on the TV recommender as a typical such agent. Much more detail is given in [CGLN05] and [LS06].

First I introduce the types that will be needed and the data constructors corresponding to these types. Several standard types will be needed: Ω (the type of the booleans), *Nat* (the type of natural numbers), *Int* (the type of integers), and *String* (the type of strings). Also *List* denotes the (unary) list type constructor.

The following type synonyms are introduced.

$$State = Occurrence \times Status$$
$$Occurrence = Date \times Time \times Channel$$
$$Date = Day \times Month \times Year$$
$$Time = Hour \times Minute$$
$$Program = Title \times Subtitle \times Duration \times (List\ Genre) \times Classification \times Synopsis$$
$$Text = List\ String.$$

In addition, *Title*, *Subtitle*, and *Synopsis* are all defined to be *String*, and *Year*, *Month*, *Day*, *Hour*, *Minute* and *Duration* are all defined to be *Nat*.

The data constructors for the type *Status* are as follows.

$$Unknown, Yes, No : Status.$$

The meaning of *Unknown* is that a recommendation (about a program having a particular occurrence) hasn't yet been made, *Yes* means that it has a positive recommendation, and *No* means that it has a negative recommendation.

In the following, basic terms are used to represent individuals. So, for example, basic terms of type *Program* are used to represent TV programs and basic terms of type *Occurrence* are used to represent occurrences (of programs).

The function of primary interest in this example is

$$uaccept : State \to \Omega,$$

whose value is true if and only if the user is willing to watch television during the time when the program having the occurrence component of the state is on. As shown in [LS06], *uaccept* is an evidence feature that is used by the TV agent with two other evidence features to determine whether or not to recommend a TV program. The presentation here of the representation and computation of *uaccept* differs somewhat from, and provides much more detail than, the presentation given in [LS06].

Suppose that the theory $\mathfrak{T}$, the belief base of the TV agent, includes the following definitions as global assumptions.

$$
\begin{array}{l}
add : Time \times Duration \rightarrow Time \\
\forall h.\ \forall m.\ \forall d. \\
\quad ((add\ ((h, m), d)) = \\
\quad\quad ((60 \times h + m + d)\ div\ 60, (60 \times h + m + d)\ mod\ 60)).
\end{array}
$$

$$
\begin{array}{l}
proj_{\,Occurrence} : State \rightarrow Occurrence \\
\forall o.\ \forall s. \\
\quad ((proj_{\,Occurrence}\ (o, s)) = o).
\end{array}
$$

$$
\begin{array}{l}
proj_{\,Duration} : Program \rightarrow Duration \\
\forall t.\ \forall t'. \forall d. \forall g. \forall c. \forall s. \\
\quad ((proj_{\,Duration}\ (t, t', d, g, c, s)) = d).
\end{array}
$$

Other global assumptions are definitions for the functions *weekday* (a predicate to determine whether a day is a weekday or not), $proj_{\,Hour}$ (the projection onto the *Hour* component of *Time*), $proj_{\,Time}$ (the projection onto the *Time* component of *Occurrence*), and *if_then_else*. Each of the global assumptions is true at each world of the intended pointed interpretation.

Suppose that $\mathfrak{T}$ includes the following definitions as local assumptions. For these, $\boldsymbol{B}_u$ is the belief modality for the user and $\boldsymbol{B}_t$ is the belief modality for the TV agent.

$$
\begin{array}{l}
uaccept : State \rightarrow \Omega \\
\boldsymbol{B}_t\ \forall x. \\
\quad ((uaccept\ x) = \\
\quad\quad \boldsymbol{B}_u(tv\_time\_acceptable\ (period\ (proj_{\,Occurrence}\ x)))).
\end{array}
$$

$$
\begin{array}{l}
period : Occurrence \rightarrow Date \times Time \times Time \\
\boldsymbol{B}_t\ \forall x. \\
\quad ((period\ x) = \\
\quad\quad ((proj_{\,Date}\ x), (proj_{\,Time}\ x), (add\ ((proj_{\,Time}\ x), (proj_{\,Duration}\ (tv\_guide\ x)))))).
\end{array}
$$

$$
\begin{array}{l}
tv\_guide : Occurrence \rightarrow Program \\
\boldsymbol{B}_t\ \forall x. \\
\quad ((tv\_guide\ x) = \\
\quad\quad if\ ((= ((21, 7, 2004), (19, 30), Win))\ x) \\
\quad\quad\quad\quad\quad\quad then\ (\text{“}Seinfeld\text{”}, \text{“”}, 30, [Sitcom], PG, \text{“}Kramer \ldots\text{”}) \\
\quad\quad else\ if\ ((= ((20, 7, 2004), (20, 30), ABC))\ x) \\
\quad\quad\quad\quad\quad\quad then\ (\text{“}The\ Bill\text{”}, \text{“”}, 50, [Drama], M, \text{“}Sun\ Hill \ldots\text{”}) \\
\quad\quad\quad\quad \vdots \\
\quad\quad else\ (\text{“ ”}, \text{“ ”}, 0, [], NA, \text{“ ”})).
\end{array}
$$

$tv\_time\_acceptable : Date \times Time \times Time \to \Omega$
$\boldsymbol{B}_t\boldsymbol{B}_u\, \forall x.$
$((tv\_time\_acceptable\ x) =$
$\quad if\ (weekday\ (proj_{Day}\ x)) \wedge (((proj_{Hour}\ (proj_{Start}\ x)) \geq 20) \wedge$
$\qquad ((proj_{Hour}\ (proj_{End}\ x)) \leq 23))\ then\ \top$
$\quad else\ if\ \neg(weekday\ (proj_{Day}\ x)) \wedge (((proj_{Hour}\ (proj_{Start}\ x)) \geq 12) \wedge$
$\qquad ((proj_{Hour}\ (proj_{End}\ x)) \leq 25))\ then\ \top$
$\quad else\ \bot).$

The term $(\text{“ ”}, \text{“ ”}, 0, [], NA, \text{“ ”})$ is the default value of type *Program*. The function *tv_time_acceptable* is obtained by a belief acquisition process (similar to the learning approach of [CGLN05]) using training examples provided by the user. Hence the modal sequence $\boldsymbol{B}_t\boldsymbol{B}_u$ at the front of the definition: the TV agent believes that the user believes that certain times are acceptable for watching television. The function *tv_guide* is similarly obtained, but from an agent (the provider of television guides) that I choose not to model. Hence there is just the modality $\boldsymbol{B}_t$ at the front of the definition.

Suppose also that there is a local assumption

$$\boldsymbol{B}_t\forall x.((tv\_guide\ x) = \mathbf{u}) \longrightarrow \boldsymbol{B}_t\boldsymbol{B}_u\forall x.((tv\_guide\ x) = \mathbf{u}),$$

where $\mathbf{u}$ is a syntactical variable ranging over terms of type *Program*. The intuitive meaning of this schema is that “if the TV agent believes the TV guide, then the TV agent believes the user believes the TV guide”. This assumption is justified because it is natural for the TV agent to assume that the user has the same beliefs about the TV guide as the TV agent does; after all, the TV guide is publicly available on the Web and in newspapers, and all versions of it are (nearly) identical.

Now let φ be the scope of the modality in the definition of *tv_guide*, that is, φ is the formula

$\forall x.$
$((tv\_guide\ x) =$
$\quad if\ ((= ((21, 7, 2004), (19, 30), Win))\ x)$
$\qquad then\ (\text{“}Seinfeld\text{”}, \text{“”}, 30, [Sitcom], PG, \text{“}Kramer \ldots\text{”})$
$\quad else\ if\ ((= ((20, 7, 2004), (20, 30), ABC))\ x)$
$\qquad then\ (\text{“}The\ Bill\text{”}, \text{“”}, 50, [Drama], M, \text{“}Sun\ Hill \ldots\text{”})$
$\qquad \vdots$
$\quad else\ (\text{“ ”}, 0, \text{“ ”}, \text{“ ”}, [], NA, \text{“ ”})).$

Then, using as local assumptions the definition of *tv_guide* (that is, $\boldsymbol{B}_t\varphi$) and the instance $\boldsymbol{B}_t\varphi \longrightarrow \boldsymbol{B}_t\boldsymbol{B}_u\varphi$ of the above local assumption, Figure 42 gives a proof of rank 0 of the

theorem $\boldsymbol{B}_t\boldsymbol{B}_u\varphi$, that is,

$$\begin{array}{l}\boldsymbol{B}_t\boldsymbol{B}_u\,\forall x.\\ \quad((tv\_guide\ x) =\\ \quad\ \ if\ ((=((21,7,2004),(19,30),\ Win))\ x)\\ \qquad\qquad\qquad\qquad then\ (\text{“}Seinfeld\text{”},\text{“”},30,[Sitcom],PG,\text{“}Kramer\ldots\text{”})\\ \quad\ \ else\ if\ ((=((20,7,2004),(20,30),ABC))\ x)\\ \qquad\qquad\qquad\qquad then\ (\text{“}The\ Bill\text{”},\text{“”},50,[Drama],M,\text{“}Sun\ Hill\ldots\text{”})\\ \qquad\vdots\\ \quad\ \ else\ (\text{“ ”},\text{“ ”},0,[],NA,\text{“ ”})).\end{array}$$

1	$\neg\boldsymbol{B}_t\boldsymbol{B}_u\varphi$	1.
1	$\boldsymbol{B}_t\varphi$	2.
1	$\neg\boldsymbol{B}_t\varphi$	3.

Figure 42: Proof of rank 0 of $\boldsymbol{B}_t\boldsymbol{B}_u\varphi$

An explanation of the proof in Figure 42 is as follows. Item 1 is the negation of the formula to be proved; 2 is a local assumption; 3 is from 1 by a derived rule from $\boldsymbol{B}_t\varphi \longrightarrow \boldsymbol{B}_t\boldsymbol{B}_u\varphi$; now the branch closes by 2 and 3.

Of course, as an alternative to this approach to handling the TV guide, it would be possible to explicitly maintain both $\boldsymbol{B}_t\varphi$ and $\boldsymbol{B}_t\boldsymbol{B}_u\varphi$ in the belief base of the TV agent. But this is not a good solution because the TV guide is regularly updated, so keeping both of these would lead to duplicated work and consistency problems. Thus it is better to maintain just $\boldsymbol{B}_t\varphi$ and infer $\boldsymbol{B}_t\boldsymbol{B}_u\varphi$ from this.

Suppose finally that there is a local assumption

$$\boldsymbol{B}_t\forall x.((period\ x) = \mathbf{u}) \longrightarrow \boldsymbol{B}_t\boldsymbol{B}_u\forall x.((period\ x) = \mathbf{u}),$$

where $\mathbf{u}$ is a syntactical variable ranging over terms of type $Date \times Time \times Time$. In a similar way, there is a proof of rank 0 of the theorem

$$\begin{array}{l}\boldsymbol{B}_t\boldsymbol{B}_u\,\forall x.\\ \quad((period\ x) =\\ \quad\ ((proj_{\,Date}\ x),(proj_{\,Time}\ x),(add\ ((proj_{\,Time}\ x),(proj_{\,Duration}\ (tv\_guide\ x))))))).\end{array}$$

Now, using these two theorems, Figure 43 gives a computation of rank 1 using $\boldsymbol{B}_t$ of $(uaccept\ (((20,7,2004),(20,30),ABC),Unknown))$. It follows from Proposition 6.9 that

$$\boldsymbol{B}_t((uaccept\ (((20,7,2004),(20,30),ABC),Unknown)) = \top)$$

is a consequence of $\mathcal{T}$. Informally, this states that the TV agent believes that the user is willing to watch television during the time when the program on channel ABC that starts at 8.30pm on 20th July, 2004, is on.

$\underline{(uaccept\ (((20,7,2004),(20,30),ABC),Unknown))}$

$\boldsymbol{B}_u(tv\_time\_acceptable\ (period\ \underline{(proj_{Occurrence}\ (((20,7,2004),(20,30),ABC),Unknown))}))$

$\boldsymbol{B}_u(tv\_time\_acceptable\ \underline{(period\ ((20,7,2004),(20,30),ABC))})$

$\vdots$

$\boldsymbol{B}_u(tv\_time\_acceptable\ ((20,7,2004),(20,30),$
$(add\ ((20,30),(proj_{Duration}\ \underline{(tv\_guide\ ((20,7,2004),(20,30),ABC))})))))$

$\vdots$

$\boldsymbol{B}_u(tv\_time\_acceptable\ ((20,7,2004),(20,30),$
$(add\ ((20,30),\underline{(proj_{Duration}\ (\text{“}The\ Bill\text{”},\text{“”},50,[Drama],M,\text{“}Sun\ Hill\ldots\text{”}))})))))$

$\boldsymbol{B}_u\ (tv\_time\_acceptable\ ((20,7,2004),(20,30),\underline{(add\ ((20,30),50))}))$

$\vdots$

$\boldsymbol{B}_u\underline{(tv\_time\_acceptable\ ((20,7,2004),(20,30),(21,20)))}$

$\boldsymbol{B}_u(if\ \underline{(weekday\ ((20,7,2004)))} \wedge (((proj_{Hour}\ (20,30)) \geq 20) \wedge ((proj_{Hour}\ (21,20)) \leq 23))\ then\ \top$
$else\ if\ \neg(weekday\ (20,7,2004)) \wedge (((proj_{Hour}\ (20,30)) \geq 12) \wedge ((proj_{Hour}\ (21,20)) \leq 25))\ then\ \top$
$else\ \bot))$

$\vdots$

$\boldsymbol{B}_u(if\ \underline{\top \wedge (((proj_{Hour}\ (20,30)) \geq 20) \wedge ((proj_{Hour}\ (21,20)) \leq 23))}\ then\ \top$
$else\ if\ \neg(weekday\ (20,7,2004)) \wedge (((proj_{Hour}\ (20,30)) \geq 12) \wedge ((proj_{Hour}\ (21,20)) \leq 25))\ then\ \top$
$else\ \bot))$

$\vdots$

$\boldsymbol{B}_u\underline{(if\ \top\ then\ \top}$
$\underline{else\ if\ \neg(weekday\ (20,7,2004)) \wedge ((proj_{Hour}\ (20,30)) \geq 12) \wedge ((proj_{Hour}\ (21,20)) \leq 25)\ then\ \top}$
$\underline{else\ \bot))}$

$\underline{\boldsymbol{B}_u\top}$

$\top$

Figure 43: Computation of rank 1 using $\boldsymbol{B}_t$ of $(uaccept\ (((20,7,2004),(20,30),ABC),Unknown))$

Example 7.2. This example illustrates the use of temporal modalities in evidence features.

Suppose the belief base of an agent is based on an alphabet that includes the following constants.

$$p : \sigma \to \Omega$$
$$q : \sigma \to \Omega$$
$$A, B, \ldots, Z : \sigma,$$

where $A, B, \ldots, Z$, are rigid. Suppose the belief base includes the following local assumptions.

$$\boldsymbol{B}\ \forall x.((p\ x) = \ \textit{if}\ x = A\ \textit{then}\ \top\ \textit{else if}\ x = \ \textit{then}\ \top\ \textit{else}\ \bot)$$
$$\boldsymbol{B}\ \forall x.((q\ x) = \ \textit{if}\ x = A\ \textit{then}\ \top\ \textit{else if}\ x = C\ \textit{then}\ \top\ \textit{else}\ \bot)$$
$$\bullet\boldsymbol{B}\ \forall x.((p\ x) = \ \textit{if}\ x = A\ \textit{then}\ \top\ \textit{else if}\ x = C\ \textit{then}\ \top\ \textit{else}\ \bot)$$
$$\bullet\boldsymbol{B}\ \forall x.((q\ x) = \ \textit{if}\ x = A\ \textit{then}\ \top\ \textit{else}\ \bot)$$
$$\bullet^2\boldsymbol{B}\ \forall x.((p\ x) = \ \textit{if}\ x = C\ \textit{then}\ \top\ \textit{else}\ \bot)$$
$$\bullet^2\boldsymbol{B}\ \forall x.((q\ x) = \ \textit{if}\ x = A\ \textit{then}\ \top\ \textit{else if}\ x = B\ \textit{then}\ \top\ \textit{else}\ \bot)$$
$$\bullet^3\boldsymbol{B}\ \forall x.((p\ x) = \bot)$$
$$\bullet^3\boldsymbol{B}\ \forall x.((q\ x) = \bot).$$

The above is the kind of belief base of an agent that remembers (some of) the past and uses a belief acquisition process to acquire new beliefs. At time 0, the definitions of p and q make them false everywhere. At time 1, some training examples arrive and the new definitions of p and q that result from a belief acquisition process using these training examples are

$$\boldsymbol{B}\ \forall x.((p\ x) = \ \textit{if}\ x = C\ \textit{then}\ \top\ \textit{else}\ \bot)$$
$$\boldsymbol{B}\ \forall x.((q\ x) = \ \textit{if}\ x = A\ \textit{then}\ \top\ \textit{else if}\ x = B\ \textit{then}\ \top\ \textit{else}\ \bot).$$

Meanwhile, at time 1, the old definitions for p and q have a $\bullet$ modality put in front of them to indicate that they are true at the last time. Thus they become as follows.

$$\bullet\boldsymbol{B}\ \forall x.((p\ x) = \bot)$$
$$\bullet\boldsymbol{B}\ \forall x.((q\ x) = \bot).$$

After two more time steps of belief acquisition, one arrives at the belief base given at the beginning of this example.

Suppose also that the belief base includes the following global assumptions.

$$\boldsymbol{\varphi}\,\boldsymbol{S}\,\boldsymbol{\psi} = \boldsymbol{\psi} \vee (\boldsymbol{\varphi} \wedge \bullet(\boldsymbol{\varphi}\,\boldsymbol{S}\,\boldsymbol{\psi}))$$
$$\bullet\boldsymbol{B}\boldsymbol{\varphi} \longrightarrow \boldsymbol{B}\bullet\boldsymbol{\varphi}.$$

Now using the global assumption

$$\bullet\boldsymbol{B}\boldsymbol{\varphi} \longrightarrow \boldsymbol{B}\bullet\boldsymbol{\varphi}$$

and the local assumption

$$\bullet\boldsymbol{B}\ \forall x.((q\ x) = \ \textit{if}\ x = A\ \textit{then}\ \top\ \textit{else}\ \bot),$$

it is easy to show that

$$\boldsymbol{B}\bullet\, \forall x.((q\ x) = \ \mathit{if}\ x = \ \mathit{then}\ \top\ \mathit{else}\ \bot)$$

is a theorem of the belief base.

Similarly, from the local assumption

$$\bullet^2\boldsymbol{B}\ \forall x.((q\ x) = \ \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else\ if}\ x = B\ \mathit{then}\ \top\ \mathit{else}\ \bot),$$

Figure 44 shows that

$$\boldsymbol{B}\bullet^2\ \forall x.((q\ x) = \ \mathit{if}\ x = \ \mathit{then}\ \top\ \mathit{else\ if}\ x = B\ \mathit{then}\ \top\ \mathit{else}\ \bot)$$

is a theorem of the belief base. An explanation of this proof is as follows. Item 1 is the negation of the formula to be proved; 2 is a local assumption; 3 is from 1 by a derived rule from the global implicational assumption $\bullet\boldsymbol{B}\boldsymbol{\varphi} \longrightarrow \boldsymbol{B}\bullet\boldsymbol{\varphi}$; 4 is from 3 by a possibility rule; 5 is from 4 by a derived rule from $\bullet\boldsymbol{B}\boldsymbol{\varphi} \longrightarrow \boldsymbol{B}\bullet\boldsymbol{\varphi}$; 6 is from 2 by a necessity rule; the branch now closes by 5 and 6.

1	$\neg\boldsymbol{B}\bullet^2\varphi$ 1.
1	$\bullet^2\boldsymbol{B}\varphi$ 2.
1	$\neg\bullet\boldsymbol{B}\bullet\varphi$ 3.
1.1$_\bullet$	$\neg\boldsymbol{B}\bullet\varphi$ 4.
1.1$_\bullet$	$\neg\bullet\boldsymbol{B}\varphi$ 5.
1.1$_\bullet$	$\bullet\boldsymbol{B}\varphi$ 6.

Figure 44: Proof of rank 0 of $\boldsymbol{B}\bullet^2\varphi$, where φ is $\forall x.((q\ x) = \ \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else\ if}\ x = B\ \mathit{then}\ \top\ \mathit{else}\ \bot)$

Using theorems such as these, Figure 45 gives a computation of rank 1 using $\boldsymbol{B}$ of $(p\ A)\ \boldsymbol{S}\ (q\ B)$. This computation shows that the result $\boldsymbol{B}((p\ A)\ \boldsymbol{S}\ (q\ B) = \top)$ is a consequence of the belief base.

It is easy to imagine the formula $\boldsymbol{B}((p\ A)\ \boldsymbol{S}\ (q\ B))$ being (part of) an evidence feature in some application: if the agent believes that $(p\ A)$ has held ever since $(q\ B)$ held at some time in the past, then perform a certain action; else perform a different action.

By way of contrast, Figure 46 shows a computation that ends in $\bot$.

One potential problem in using a global assumption such as $\boldsymbol{\varphi}\,\boldsymbol{S}\,\boldsymbol{\psi} = \boldsymbol{\psi} \vee (\boldsymbol{\varphi} \wedge \bullet(\boldsymbol{\varphi}\,\boldsymbol{S}\,\boldsymbol{\psi}))$ in computations is that it might allow them to run forever. Thus one must be careful to ensure that the belief base is such that any computation using this assumption will indeed terminate. For computations similar to Figures 45 and 46 that go into the past, one can ensure termination by having beliefs such as

$$\bullet^n\boldsymbol{B}\ \forall x.((p\ x) = \bot)$$
$$\bullet^n\boldsymbol{B}\ \forall x.((q\ x) = \bot),$$

for some $n \geq 0$, in the belief base.

Example 7.3. In Example 7.2, it was assumed that it was appropriate to maintain explicit definitions for the predicate p at the various times. If these definitions of p are large, it may

$\underline{(p\ A)\ \boldsymbol{S}\ (q\ B)}$
$\underline{(q\ B)} \vee ((p\ A) \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ B)))$
$\vdots$
$\underline{\bot \vee ((p\ A) \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ B)))}$
$\underline{(p\ A)} \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ B))$
$\vdots$
$\underline{\top \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ B))}$
$\bullet\underline{((p\ A)\ \boldsymbol{S}\ (q\ B))}$
$\bullet(\underline{(q\ B)} \vee ((p\ A) \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ B))))$
$\vdots$
$\bullet\underline{(\bot \vee ((p\ A) \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ B))))}$
$\bullet(\underline{(p\ A)} \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ B)))$
$\vdots$
$\bullet\underline{(\top \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ B)))}$
$\bullet^2\underline{((p\ A)\ \boldsymbol{S}\ (q\ B))}$
$\bullet^2(\underline{(q\ B)} \vee ((p\ A) \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ B))))$
$\vdots$
$\bullet^2\underline{(\top \vee ((p\ A) \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ B))))}$
$\underline{\bullet\bullet\top}$
$\underline{\bullet\top}$
$\top$

Figure 45: Computation of rank 1 using $\boldsymbol{B}$ of $(p\ A)\ \boldsymbol{S}\ (q\ B)$

be better to give the definition at each time by an incremental change to the one at the last time. This example shows how to do this and provides an interesting use of modalities applied to predicates.

Suppose that the belief base for an agent includes the following local assumptions.

$\boldsymbol{B}\ \forall x.((p\ x) = \mathit{if}\ x = B\ \mathit{then}\ \top\ \mathit{else\ if}\ x = C\ \mathit{then}\ \bot\ \mathit{else}\ (\bullet p\ x))$
$\bullet\boldsymbol{B}\ \forall x.((p\ x) = \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else\ if}\ x = C\ \mathit{then}\ \top\ \mathit{else}\ \bot).$

Here the definition of p at the current time (which is the same as for Example 7.2) is given as an incremental change to the definition of p at the last time. Note the occurrence of $\bullet p$ in the first formula in which the modality $\bullet$ is applied to the predicate p. It is also assumed

$\underline{(p\ A)\ \boldsymbol{S}\ (q\ D)}$
$\underline{(q\ D)} \vee ((p\ A) \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ D)))$
$\vdots$
$\underline{\bot \vee ((p\ A) \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ D)))}$
$\underline{(p\ A)} \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ D))$
$\vdots$
$\underline{\top \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ D))}$
$\bullet\underline{((p\ A)\ \boldsymbol{S}\ (q\ D))}$
$\bullet(\underline{(q\ D)} \vee ((p\ A) \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ D))))$
$\vdots$
$\bullet\underline{(\bot \vee ((p\ A) \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ D))))}$
$\bullet(\underline{(p\ A)} \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ D)))$
$\vdots$
$\bullet\underline{(\top \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ D)))}$
$\bullet^2\underline{((p\ A)\ \boldsymbol{S}\ (q\ D))}$
$\bullet^2(\underline{(q\ D)} \vee ((p\ A) \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ D))))$
$\vdots$
$\bullet^2\underline{(\bot \vee ((p\ A) \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ D))))}$
$\bullet^2(\underline{(p\ A)} \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ D)))$
$\vdots$
$\bullet^2\underline{(\bot \wedge \bullet((p\ A)\ \boldsymbol{S}\ (q\ D)))}$
$\underline{\bullet\bullet\bot}$
$\underline{\bullet\bot}$
$\bot$

Figure 46: Computation of rank 1 using $\boldsymbol{B}$ of $(p\ A)\ \boldsymbol{S}\ (q\ D)$

that the belief base contains the global assumption

$$\bullet \boldsymbol{B}\varphi \longrightarrow \boldsymbol{B}\bullet\varphi.$$

Figure 47 gives the computation using $\boldsymbol{B}$ of $(p\ A)$, in which use is made of the global assumptions (3.1) and (3.3). It follows from this that $\boldsymbol{B}((p\ A) = \top)$ is a consequence of the belief base.

$$\underline{(p\ A)}$$
$$\textit{if } \underline{A = B} \textit{ then } \top \textit{ else if } A = C \textit{ then } \bot \textit{ else } (\bullet p\ A)$$
$$\underline{\textit{if } \bot \textit{ then } \top \textit{ else if } A = C \textit{ then } \bot \textit{ else } (\bullet p\ A)}$$
$$\textit{if } \underline{A = C} \textit{ then } \bot \textit{ else } (\bullet p\ A)$$
$$\underline{\textit{if } \bot \textit{ then } \bot \textit{ else } (\bullet p\ A)}$$
$$\underline{(\bullet p\ A)}$$
$$\bullet\underline{(p\ A)}$$
$$\bullet\ \textit{if } \underline{A = A} \textit{ then } \top \textit{ else if } A = C \textit{ then } \top \textit{ else } \bot$$
$$\bullet\ \underline{\textit{if } \top \textit{ then } \top \textit{ else if } A = C \textit{ then } \top \textit{ else } \bot}$$
$$\underline{\bullet\top}$$
$$\top$$

Figure 47: Computation of rank 1 using $\boldsymbol{B}$ of $(p\ A)$

If this chain of incrementally modified definitions goes back too far, it can become cumbersome. In this case, it may be better to cut the chain off at some time in the recent past. For instance, for the belief base of this example, one can essentially recover the explicit definition of p at the current time by means of the computation of rank 1 in Figure 48 in which there is a substitution of the definition of p at the last time. The last step in this computation uses the following instance of the global assumption (3.1).

$$\bullet\varphi = \varphi,$$

where $\varphi \equiv \textit{if } x = A \textit{ then } \top \textit{ else if } x = C \textit{ then } \top \textit{ else } \bot$ is rigid. The last formula in the computation can be simplified, if desired, to recover

$$\boldsymbol{B}\ \forall x.((p\ x) = \textit{ if } x = B \textit{ then } \top \textit{ else if } x = A \textit{ then } \top \textit{ else } \bot).$$

Note that the computation in Figure 48 does not use the leftmost selection rule. Instead the computation can be thought of as a partial evaluation of the definition of p at the current time that uses an appropriate selection of redexes and stops at an appropriate point (to produce the desired outcome from the partial evaluation).

Even though equality of the first and last formulas in Figure 48 has been shown to be a consequence of the (original) belief base, a further restriction is needed to ensure their computational equivalence. This restriction is that the last formula is only ever used to compute terms of the form $(p\ t)$, where t is rigid. Figures 49 and 50 give the respective

$$\boldsymbol{B}\ \forall x.((p\ x) = \mathit{if}\ x = B\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \bot\ \mathit{else}\ \underline{(\bullet p\ x)})$$
$$\boldsymbol{B}\ \forall x.((p\ x) = \mathit{if}\ x = B\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \bot\ \mathit{else}\ \bullet\underline{(p\ x)})$$
$$\boldsymbol{B}\ \forall x.((p\ x) = \mathit{if}\ x = B\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \bot$$
$$\mathit{else}\ \underline{\bullet(\mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \top\ \mathit{else}\ \bot)})$$
$$\boldsymbol{B}\ \forall x.((p\ x) = \mathit{if}\ x = B\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \bot$$
$$\mathit{else}\ \mathit{if}\ x = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \top\ \mathit{else}\ \bot)$$

Figure 48: Computation of $\boldsymbol{B}\ \forall x.((p\ x) = \mathit{if}\ x = B\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = C\ \mathit{then}\ \bot\ \mathit{else}\ (\bullet p\ x))$

computations of $(p\ t)$ in each case. The rigidity condition is needed at several places in the computation in Figure 49. Therefore, to maintain the computational equivalence of the two belief bases, it needs to be imposed in Figure 50. In practice, arguments to functions in belief bases are usually rigid, so this restriction is unlikely to cause any inconvenience.

$$\underline{(p\ t)}$$
$$\mathit{if}\ t = B\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ t = C\ \mathit{then}\ \bot\ \mathit{else}\ \underline{(\bullet p\ t)}$$
$$\mathit{if}\ t = B\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ t = C\ \mathit{then}\ \bot\ \mathit{else}\ \bullet\underline{(p\ t)}$$
$$\mathit{if}\ t = B\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ t = C\ \mathit{then}\ \bot$$
$$\mathit{else}\ \underline{\bullet(\mathit{if}\ t = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ t = C\ \mathit{then}\ \top\ \mathit{else}\ \bot)}$$
$$\mathit{if}\ t = B\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ t = C\ \mathit{then}\ \bot$$
$$\mathit{else}\ \mathit{if}\ t = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ t = C\ \mathit{then}\ \top\ \mathit{else}\ \bot$$

Figure 49: Partial computation using $\boldsymbol{B}$ of $(p\ t)$ for the original belief base

$$\underline{(p\ t)}$$
$$\mathit{if}\ t = B\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ t = C\ \mathit{then}\ \bot$$
$$\mathit{else}\ \mathit{if}\ t = A\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ t = C\ \mathit{then}\ \top\ \mathit{else}\ \bot$$

Figure 50: Partial computation using $\boldsymbol{B}$ of $(p\ t)$ for new belief base

Use of modalities applied to predicates can be avoided in Example 7.3 by working throughout with $\bullet(p\ x)$ instead of $(\bullet p\ x)$. However, in the next example, modal terms will become essential when the predicate p is replaced by a function f (because $(f\ x)$ will not generally be a formula).

Example 7.4. In Example 7.3, I showed how it is possible to maintain incremental changes

of definitions of a predicate at successive times. In this example, the technique is extended to functions (that are not predicates).

Suppose that the function $f : \sigma \to Nat$ has definitions at the current and last times as follows.

$$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ x = C\ \mathit{then}\ 3\ \mathit{else}\ (\bullet f\ x))$$
$$\bullet\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ x = A\ \mathit{then}\ 42\ \mathit{else\ if}\ x = B\ \mathit{then}\ 21\ \mathit{else\ if}\ x = C\ \mathit{then}\ 15\ \mathit{else}\ 0)$$

The value of f at the current time on some argument can be computed using the definition of f as in Figure 51. It follows from this computation that $\boldsymbol{B}((f\ A) = 42)$ is a consequence of the belief base.

$\underline{(f\ A)}$
$\mathit{if}\ \underline{A = C}\ \mathit{then}\ 3\ \mathit{else}\ (\bullet f\ A)$
$\underline{\mathit{if}\ \bot\ \mathit{then}\ 3\ \mathit{else}\ (\bullet f\ A)}$
$\underline{(\bullet f\ A)}$
$\bullet\underline{(f\ A)}$
$\bullet\mathit{if}\ \underline{A = A}\ \mathit{then}\ 42\ \mathit{else\ if}\ A = B\ \mathit{then}\ 21\ \mathit{else\ if}\ A = C\ \mathit{then}\ 15\ \mathit{else}\ 0$
$\bullet\underline{\mathit{if}\ \top\ \mathit{then}\ 42\ \mathit{else\ if}\ A = B\ \mathit{then}\ 21\ \mathit{else\ if}\ A = C\ \mathit{then}\ 15\ \mathit{else}\ 0}$
$\bullet 42$
42

Figure 51: Computation using $\boldsymbol{B}$ of $(f\ A)$

The definition of f at the current time can be written equivalently as

$$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ x = C\ \mathit{then}\ 3\ \mathit{else\ if}\ x = A\ \mathit{then}\ 42\ \mathit{else\ if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ 0).$$

This follows from the partial evaluation of Figure 52 which is a computation of rank 1 of $\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ x = C\ \mathit{then}\ 3\ \mathit{else}\ (\bullet f\ x))$. In a similar way to Example 7.3, this formula should only be used to compute terms of the form $(f\ t)$, where t is rigid.

Example 7.5. This example shows how to handle beliefs whose body is an abstraction. Consider an agent with belief modality $\boldsymbol{B}$ which has a belief base that includes

$$\bullet\boldsymbol{B}(f = \lambda x.\mathit{if}\ (p\ x)\ \mathit{then}\ A\ \mathit{else\ if}\ (q\ x)\ \mathit{then}\ B\ \mathit{else}\ C),$$

where A, B, and C are rigid constants, and $f : \alpha \to \beta$. Now let t be a rigid term of type α and suppose the agent wants to compute the value of $(\bullet f\ t)$. This computation may then proceed as in Figure 53. It follows from Figure 53 that $\boldsymbol{B}((\bullet f\ t) = B)$ is a consequence of the belief base.

This computation has used the following instance of global assumption (3.5).

$$\bullet\lambda x.s = \lambda x.\bullet s,$$

$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ x = C\ \mathit{then}\ 3\ \mathit{else}\ \underline{(\bullet f\ x)})$
$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ x = C\ \mathit{then}\ 3\ \mathit{else}\ \bullet\underline{(f\ x)})$
$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ x = C\ \mathit{then}\ 3$
$\quad \mathit{else}\ \bullet(\mathit{if}\ x = A\ \mathit{then}\ 42\ \mathit{else\ if}\ x = B\ \mathit{then}\ 21\ \mathit{else\ if}\ x = C\ \mathit{then}\ 15\ \mathit{else}\ 0))$
$\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ x = C\ \mathit{then}\ 3$
$\quad \mathit{else\ if}\ x = A\ \mathit{then}\ 42\ \mathit{else\ if}\ x = B\ \mathit{then}\ 21\ \mathit{else\ if}\ x = C\ \mathit{then}\ 15\ \mathit{else}\ 0)$

Figure 52: Computation of rank 1 of $\boldsymbol{B}\ \forall x.((f\ x) = \mathit{if}\ x = C\ \mathit{then}\ 3\ \mathit{else}\ (\bullet f\ x))$

$(\bullet\underline{f}\ t)$
$(\underline{\bullet\lambda x.\mathit{if}\ (p\ x)\ \mathit{then}\ A\ \mathit{else\ if}\ (q\ x)\ \mathit{then}\ B\ \mathit{else}\ C}\ \ t)$
$\underline{(\lambda x.\bullet\mathit{if}\ (p\ x)\ \mathit{then}\ A\ \mathit{else\ if}\ (q\ x)\ \mathit{then}\ B\ \mathit{else}\ C\ \ t)}$
$\bullet\mathit{if}\ \underline{(p\ t)}\ \mathit{then}\ A\ \mathit{else\ if}\ (q\ t)\ \mathit{then}\ B\ \mathit{else}\ C$
$\quad\vdots$
$\bullet\underline{\mathit{if}\ \bot\ \mathit{then}\ A\ \mathit{else\ if}\ (q\ t)\ \mathit{then}\ B\ \mathit{else}\ C}$
$\bullet\mathit{if}\ \underline{(q\ t)}\ \mathit{then}\ B\ \mathit{else}\ C$
$\quad\vdots$
$\bullet\underline{\mathit{if}\ \top\ \mathit{then}\ B\ \mathit{else}\ C}$
$\underline{\bullet B}$
B

Figure 53: Computation with abstractions and modalities

where $s \equiv \mathit{if}\ (p\ x)\ \mathit{then}\ A\ \mathit{else\ if}\ (q\ x)\ \mathit{then}\ B\ \mathit{else}\ C$. It has also used the global assumption

$$\bullet\boldsymbol{B}\varphi \longrightarrow \boldsymbol{B}\bullet\varphi,$$

where $\varphi \equiv f = \lambda x.\mathit{if}\ (p\ x)\ \mathit{then}\ A\ \mathit{else\ if}\ (q\ x)\ \mathit{then}\ B\ \mathit{else}\ C$, to switch the $\bullet$ and $\boldsymbol{B}$.

Example 7.6. This example shows the usefulness of modal terms for predicate rewrite systems discussed in Section 5.2.

Consider the following predicate rewrite system.

$\mathit{top} \rightarrowtail \boldsymbol{B}_i(\mathit{setExists}_1\ (\wedge_2\ \mathit{top}\ \mathit{top}))$
$\mathit{top} \rightarrowtail \bullet\boldsymbol{B}_j\mathit{top}$
$\mathit{top} \rightarrowtail \blacklozenge\boldsymbol{B}_j\mathit{top}$
$\mathit{top} \rightarrowtail p$
$\mathit{top} \rightarrowtail q$

$$top \rightarrowtail r.$$

The following is a path in the predicate space defined by the rewrite system.

$$\begin{aligned} top \;&\rightsquigarrow\; \boldsymbol{B}_i(setExists_1\ (\wedge_2\ top\ top)) \rightsquigarrow\; \boldsymbol{B}_i(setExists_1\ (\wedge_2\ \bullet\boldsymbol{B}_j top\ \ top)) \\ &\rightsquigarrow\; \boldsymbol{B}_i(setExists_1\ (\wedge_2\ \bullet\boldsymbol{B}_j p\ \ top)) \\ &\rightsquigarrow \cdots \rightsquigarrow\; \boldsymbol{B}_i(setExists_1\ (\wedge_2\ \bullet\boldsymbol{B}_j p\ \blacklozenge\boldsymbol{B}_j q)). \end{aligned}$$

Thus the predicate $\boldsymbol{B}_i(setExists_1\ (\wedge_2\ \bullet\boldsymbol{B}_j p\ \blacklozenge\boldsymbol{B}_j q))$ is in the space of predicates generated by the predicate rewrite system. If s is a set, the informal meaning of the formula

$$(\boldsymbol{B}_i(setExists_1\ (\wedge_2\ \bullet\boldsymbol{B}_j p\ \blacklozenge\boldsymbol{B}_j q))\ \ s)$$

is that agent i believes that there is an item in the set s such that at the last time step agent j believed the item satisfied p and that at some time in the past agent j believed that the item satisfied q.

Example 7.7. This example illustrates how modal terms might be useful for learning applications. Consider an agent i that selects an action partly in response to the perceived beliefs of another agent j. Suppose that it matters whether the agent j believes something now or believed something at the last time or always in the past. Then a suitable predicate rewrite system for the hypothesis language associated with this learning task might be something similar to the following.

$$\begin{aligned} &top \rightarrowtail \boldsymbol{B}_j(p\ (\wedge_2\ top\ top)) \\ &top \rightarrowtail \boldsymbol{B}_j\bullet(p\ (\wedge_2\ top\ top)) \\ &top \rightarrowtail \boldsymbol{B}_j\blacksquare(p\ (\wedge_2\ top\ top)) \\ &top \rightarrowtail r \\ &top \rightarrowtail s \\ &top \rightarrowtail t \\ &\quad\vdots \end{aligned}$$

Thus predicates such as

$$\boldsymbol{B}_j\bullet(p\ (\wedge_2\ r\ s))$$

and

$$\boldsymbol{B}_j\blacksquare(p\ (\wedge_2\ t\ r))$$

could appear in the belief base for agent i.

Now suppose this agent computes its policy function on some argument to determine which action to select. During this computation, formulas such as

$$(\boldsymbol{B}_j\bullet(p\ (\wedge_2\ r\ s))\ t),$$

for some rigid term t, will need to be evaluated. This can be done by using the global assumption (3.3) to turn it into

$$\boldsymbol{B}_j\bullet((p\ (\wedge_2\ r\ s))\ t).$$

To finish, here is an extended example showing how an agent can logically reason with a mixture of certain and uncertain assumptions expressed in theories, and how conclusions obtained by reasoning with such theories can have a level of certainty directly associated with them.

Example 7.8. Consider an agent that makes recommendations of TV programs to a user. The agent has access to the TV guide through the definition of the function *tv_guide*. It also knows about the user's preferences for TV programs through the definition of the function *likes*, the uncertainty in which is modelled by densities in its codomain. Suppose now that the agent is asked to make a recommendation about a program at a particular occurrence (that is, date, time, and channel), except that there is some uncertainty in the actual occurrence intended by the user; this uncertainty, which is modelled in the definition of the density *choice*, could come about, for example, if the user asked the somewhat ambiguous question "Are there any good programs on *ABC* during dinner time?". The question the agent needs to answer is the following: given the uncertainty in *choice* and *likes*, what is the probability that the user likes the program that is on at the occurrence intended by the user.

This situation is modelled as follows. First, here are some type synonyms.

$$\begin{aligned}
&\mathit{Occurrence} = \mathit{Date} \times \mathit{Time} \times \mathit{Channel}\\
&\mathit{Date} = \mathit{Day} \times \mathit{Month} \times \mathit{Year}\\
&\mathit{Time} = \mathit{Hour} \times \mathit{Minute}\\
&\mathit{Program} = \mathit{Title} \times \mathit{Duration} \times \mathit{Genre} \times \mathit{Classification} \times \mathit{Synopsis}.
\end{aligned}$$

Here is the definition of the density *choice* that models the uncertainty in the intended occurrence, where $\boldsymbol{B}$ is the belief modality of the agent.

$$\begin{aligned}
&\mathit{choice} : \mathit{Density\ Occurrence}\\
&\boldsymbol{B}\ \forall x.((\mathit{choice}\ x) =\\
&\qquad \mathit{if}\ x = ((21,7,2007),(19,30),\mathit{ABC})\ \mathit{then}\ 0.8\\
&\qquad \mathit{else\ if}\ x = ((21,7,2007),(20,30),\mathit{ABC})\ \mathit{then}\ 0.2\\
&\qquad \mathit{else}\ 0).
\end{aligned}$$

So the uncertainty is in the time of the program; it is probably 7.30pm, but it could be 8.30pm. Note that an equivalent form of the definition for *choice* is

$$\boldsymbol{B}(\mathit{choice} = \lambda x.\mathit{if}\ x = ((21,7,2007),(19,30),\mathit{ABC})\ \mathit{then}\ 0.8\ \mathit{else}\ \ldots),$$

which is the form of the definition that is used below.

Next there is the TV guide that maps occurrences to programs.

$tv\_guide : Occurrence \to Program$

$\boldsymbol{B}\ \forall x.((tv\_guide\ x) =$

$\quad if\ ((= ((20,7,2007),(11,30),Win))\ x)$

$\qquad then\ (\text{“}NFL\ Football\text{”}, 60, Sport, G, \text{“}The\ Buffalo\ldots\text{”})$

$\quad else\ if\ ((= ((21,7,2007),(19,30),ABC))\ x)$

$\qquad then\ (\text{“}Seinfeld\text{”}, 30, Sitcom, PG, \text{“}Kramer\ldots\text{”})$

$\quad else\ if\ ((= ((21,7,2007),(20,30),ABC))\ x)$

$\qquad then\ (\text{“}The\ Bill\text{”}, 50, Drama, M, \text{“}Sun\ Hill\ldots\text{”})$

$\quad else\ if\ ((= ((21,7,2007),(21,30),ABC))\ x)$

$\qquad then\ (\text{“}Numb3rs\text{”}, 60, Crime, M, \text{“}When\ldots\text{”})$

$\quad \vdots$

$\quad else\ (\text{“ ”}, 0, NULL, NA, \text{“ ”})).$

Finally, here is the definition of the function *likes*. (A more realistic modelling of the situation would involve also using the belief modality of the user in the definition of *likes*, but this is left out for simplicity.)

$likes : Program \to Density\ \Omega$

$\boldsymbol{B}\ \forall x.((likes\ x) =$

$\quad if\ (projTitle \circ (=\ \text{“}NFL\ Football\text{”})\ x)\ then\ \lambda y.if\ y = \top\ then\ 1\ else\ if\ y = \bot\ then\ 0\ else\ 0$

$\quad else\ if\ (projGenre \circ (=\ Movie)\ x)\ then\ \lambda y.if\ y = \top\ then\ 0.75\ else\ if\ y = \bot\ then\ 0.25\ else\ 0$

$\quad else\ if\ (projGenre \circ (=\ Documentary)\ x)\ then\ \lambda y.if\ y = \top\ then\ 1\ else\ if\ y = \bot\ then\ 0\ else\ 0$

$\quad else\ (\bullet likes\ x))$

$\bullet\boldsymbol{B}\ \forall x.((likes\ x) =$

$\quad if\ (projTitle \circ (=\ \text{“}NFL\ Football\text{”})\ x)\ then\ \lambda y.if\ y = \top\ then\ 0.9\ else\ if\ y = \bot\ then\ 0.1\ else\ 0$

$\quad else\ if\ (projGenre \circ (=\ Drama)\ x)\ then\ \lambda y.if\ y = \top\ then\ 0.2\ else\ if\ y = \bot\ then\ 0.8\ else\ 0$

$\quad else\ (\bullet likes\ x))$

$\bullet^2\boldsymbol{B}\ \forall x.((likes\ x) =$

$\quad if\ (projGenre \circ (=\ Sitcom)\ x)\ then\ \lambda y.if\ y = \top\ then\ 0.9\ else\ if\ y = \bot\ then\ 0.1\ else\ 0$

$\quad else\ (\bullet likes\ x))$

$\bullet^3\boldsymbol{B}\ \forall x.((likes\ x) =$

$\quad \lambda y.if\ y = \top\ then\ 0\ else\ if\ y = \bot\ then\ 1\ else\ 0).$

Recall that the agent needs to compute the probability that the user likes the program which is on at the occurrence intended by the user. This amounts to computing (using $\boldsymbol{B}$) the value of the term

$((choice\ \$\ tv\_guide)\ \S\ likes),$

which is a term of type $Density\ \Omega$. The answer for this computation is

$\lambda z.if\ z = \top\ then\ 0.76\ else\ if\ z = \bot\ then\ 0.24\ else\ if\ z = \top\ then\ 0.72\ else\ if\ z = \bot\ then\ 0.08\ else\ 0,$

which can be simplified to

$$\lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else}\ \mathit{if}\ z = \bot\ \mathit{then}\ 0.24\ \mathit{else}\ 0.$$

Thus

$$\boldsymbol{B}(((\mathit{choice}\ \$\ \mathit{tv\_guide})\ \S\ \mathit{likes}) = \lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else}\ \mathit{if}\ z = \bot\ \mathit{then}\ 0.24\ \mathit{else}\ 0)$$

is a consequence of the belief base of the agent. On this basis, at least in part, the agent can now decide whether to recommend the program or not.

The computation is somewhat complicated. To make it easier to understand, the computation is broken up into two steps. Figure 54 gives the computation of $(\mathit{choice}\ \$\ \mathit{tv\_guide})$ which gives the answer

$$\begin{array}{l}\lambda y.\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, 30, \mathit{Sitcom}, \mathit{PG}, \text{“}\mathit{Kramer}\ldots\text{”})\ \mathit{then}\ 0.8\ \mathit{else}\\ \quad \mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, 50, \mathit{Drama}, \mathit{M}, \text{“}\mathit{Sun\ Hill}\ldots\text{”})\ \mathit{then}\ 0.2\ \mathit{else}\ 0,\end{array}$$

which is a term of type *Density Program*. Then the second part of the computation in Figure 55 is that of

$$((\lambda y.\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \ \mathit{else}\ \mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0)\ \S\ \mathit{likes})$$

that has the answer

$$\lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else}\ \mathit{if}\ z = \bot\ \mathit{then}\ 0.24\ \mathit{else}\ \mathit{if}\ z = \top\ \mathit{then}\ 0.72\ \mathit{else}\ \mathit{if}\ z = \bot\ \mathit{then}\ 0.08\ \mathit{else}\ 0.$$

The preceding example illustrates that higher-order logic is an excellent setting for integrating probability and logic. Indeed, in this setting, no new conceptual machinery is needed at all to achieve the integration. Probabilistic constructs, such as densities, expectations, and so on, can be naturally included in higher-order theories. Furthermore, the connectives, quantifiers, modalities, and probabilistic constructs all fit harmoniously together.

8 Related Work

This section contains a discussion of related work.

In the past, the motivations for employing modal higher-order logic have mostly been of a philosophical or linguistic nature, and outside these areas there have been very few works. For a brief historical account of these motivations, the reader is referred to the excellent handbook chapter of Muskens [Musar] which develops a logic that repairs some deficiencies in Montague's Intensional Logic. A recent account of modal higher-order logic, motivated by philosophical considerations (Gödel's ontological argument), is given in [Fit02]. An earlier work motivated by mainly linguistic considerations is [Gal75]. All these treatments differ quite markedly from the one in this paper as none has a type system that is as rich as the one presented here, they are not concerned with computation (only, at most, deduction), and their intended applications are quite different (therefore, they emphasise different technical machinery).

This paper has benefitted greatly from the large literature on modal logic. In particular, the books [GKWZ03], [FM98] and [Fit02] proved to be very useful. Closely related are works

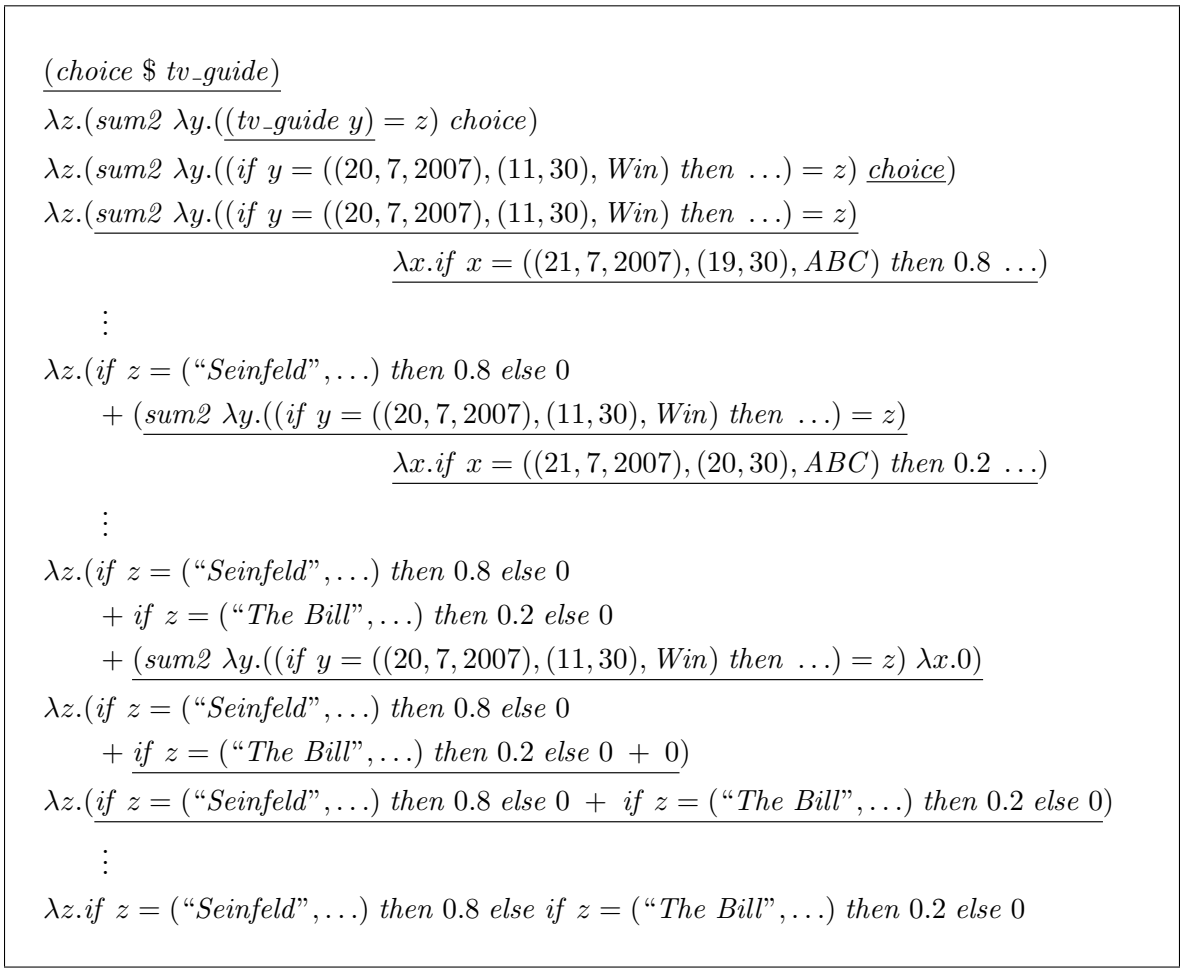

Figure 54: Computation of (*choice* $ *tv_guide*)

on the application of modal logic to artificial intelligence of which [FHMV95] inspired parts of this paper.

Modal computation has been studied for 20 years or so, mostly in the logic programming community in the context of epistemic or temporal logic programming languages. Useful surveys of this work are in [OM94] and [Ger01]. A recent paper showing the current state of the art of modal logic programming is [Ngu06]. What is common between these works and this paper is the emphasis on epistemic and temporal modalities. What is different is that almost all are based on Prolog and are, therefore, first order, and it seems they either provide epistemic modalities or temporal modalities, but not both. The reasoning system presented in this paper extends typical modal theorem proving systems, such as those in [FM98] and [Fit02], largely in that it also has a computational component. The idea of having a specialised equational reasoning component in a theorem-proving system is, of course, an old one. What is interesting here is that for agent applications it is the computational (equational reasoning) component that is the most important; the most common reasoning task in an agent is to evaluate a function call. Generally, the theorem-proving tasks are subsidiary to this.

I turn now to related work in higher-order logic. The advantages of using a higher-order approach to computational logic have been advocated for at least the last 30 years. First,

$$\underline{((\lambda y.\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \ \mathit{else\ if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0) \S\ \mathit{likes})}$$
$$\lambda z.(\mathit{sum1}\ \lambda y.(\underline{((\lambda y.\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \mathit{else}}$$
$$\underline{\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0)\ y)} \times ((\mathit{likes}\ y)\ z)))$$
$$\lambda z.(\mathit{sum1}\ \lambda y.(\underline{(\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \mathit{else}}$$
$$\underline{\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0) \times} ((\mathit{likes}\ y)\ z)))$$
$$\vdots$$
$$\lambda z.(\mathit{sum1}\ \lambda y.(\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8 \times (\underline{(\mathit{likes}\ (\text{“}\mathit{Seinfeld}\text{”}, \ldots))}\ z)$$
$$\mathit{else}\ ((\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0) \times ((\mathit{likes}\ y)\ z))))$$
$$\vdots$$
$$\lambda z.(\mathit{sum1}\ \lambda y.(\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8 \times (\underline{(\bullet \mathit{likes}\ (\text{“}\mathit{Seinfeld}\text{”}, \ldots))}\ z)$$
$$\mathit{else}\ ((\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0) \times ((\mathit{likes}\ y)\ z))))$$
$$\vdots$$
$$\lambda z.(\mathit{sum1}\ \lambda y.(\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8 \times (\underline{(\bullet^2 \mathit{likes}\ (\text{“}\mathit{Seinfeld}\text{”}, \ldots))}\ z)$$
$$\mathit{else}\ ((\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0) \times ((\mathit{likes}\ y)\ z))))$$
$$\vdots$$
$$\lambda z.(\mathit{sum1}\ \lambda y.(\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8 \times \underline{(\lambda y.\mathit{if}\ y = \top\ \mathit{then}\ 0.9\ \mathit{else\ if}\ y = \bot\ \mathit{then}\ 0.1\ \mathit{else}\ 0\ \ z)}$$
$$\mathit{else}\ ((\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0) \times ((\mathit{likes}\ y)\ z))))$$
$$\lambda z.(\mathit{sum1}\ \lambda y.(\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ \underline{0.8 \times (\mathit{if}\ z = \top\ \mathit{then}\ 0.9\ \mathit{else\ if}\ z = \bot\ \mathit{then}\ 0.1\ \mathit{else}\ 0)}$$
$$\mathit{else}\ ((\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0) \times ((\mathit{likes}\ y)\ z))))$$
$$\vdots$$
$$\lambda z.(\mathit{sum1}\ \lambda y.(\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ (\mathit{if}\ z = \top\ \mathit{then}\ 0.72\ \mathit{else\ if}\ z = \bot\ \mathit{then}\ 0.08\ \mathit{else}\ 0)\ \mathit{else}$$
$$\underline{((\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ 0.2\ \mathit{else}\ 0) \times ((\mathit{likes}\ y)\ z))}))$$
$$\vdots$$
$$\lambda z.\underline{(\mathit{sum1}\ \lambda y.(\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ (\mathit{if}\ z = \top\ \mathit{then}\ 0.72\ \mathit{else\ if}\ z = \bot\ \mathit{then}\ 0.08\ \mathit{else}\ 0)\ \mathit{else}}$$
$$\underline{\mathit{if}\ y = (\text{“}\mathit{The\ Bill}\text{”}, \ldots)\ \mathit{then}\ (\mathit{if}\ z = \top\ \mathit{then}\ 0.04\ \mathit{else\ if}\ z = \bot\ \mathit{then}\ 0.16\ \mathit{else}\ 0)\ \mathit{else}\ 0))}$$
$$\vdots$$
$$\lambda z.(\underline{(\mathit{if}\ z = \top\ \mathit{then}\ 0.72\ \mathit{else\ if}\ z = \bot\ \mathit{then}\ 0.08\ \mathit{else}\ 0) +}$$
$$\underline{(\mathit{if}\ z = \top\ \mathit{then}\ 0.04\ \mathit{else\ if}\ z = \bot\ \mathit{then}\ 0.16\ \mathit{else}\ 0)})$$
$$\vdots$$
$$\lambda z.\mathit{if}\ z = \top\ \mathit{then}\ 0.76\ \mathit{else\ if}\ z = \bot\ \mathit{then}\ 0.24\ \mathit{else\ if}\ z = \top\ \mathit{then}\ 0.72\ \mathit{else\ if}\ z = \bot\ \mathit{then}\ 0.08\ \mathit{else}\ 0$$

Figure 55: Computation of $((\lambda y.\mathit{if}\ y = (\text{“}\mathit{Seinfeld}\text{”}, \ldots)\ \mathit{then}\ 0.8\ \mathit{else}\ \ldots) \S\ \mathit{likes})$

the functional programming community has used higher-order functions from the very beginning. The latest versions of functional languages, such as Haskell [Jon03], show the power and elegance of higher-order functions, as well as related features such as strong type systems. Of course, the traditional foundation for functional programming languages has been the λ-calculus, rather than a higher-order *logic*. However, it is possible to regard functional programs as equational theories in a logic such as the one introduced here and this also provides a useful semantics. The reasoning system presented here extends the core computational mechanisms of existing functional programming languages in that it also contains a theorem-proving system, it is modal, and it admits logic programming idioms through programming with abstractions.

In the 1980s, higher-order programming in the logic programming community was introduced through the language λProlog [NM98]. The logical foundations of λProlog are provided by almost exactly the logic studied here (with the modal facilities removed). However, a different sublogic is used for λProlog programs than the equational theories proposed here. In λProlog, program statements are higher-order hereditary Harrop formulas, a generalisation of the definite clauses used by Prolog. The language provides an elegant use of λ-terms as data structures, meta-programming facilities, universal quantification and implications in goals, amongst other features.

A long-term interest amongst researchers in declarative programming has been the goal of building integrated functional logic programming languages. Probably the best developed of these functional logic languages is the Curry language [He], which is the result of an international collaboration over the last decade. A survey of functional logic programming up to 1994 is in [Han94].

There are many other outstanding examples of systems that exploit the power of higher-order logic. For example, the HOL system [GM93] is an environment for interactive theorem proving in higher-order logic. Its most outstanding feature is its high degree of programmability through the meta-language ML. The system has a wide variety of uses from formalising pure mathematics to verification of industrial hardware. In addition, there are at least a dozen other systems related to HOL. On the theoretical side, much of the research in theoretical computer science, especially semantics, is based on the λ-calculus and hence is intrinsically higher order in nature.

The material on probabilistic reasoning in this paper is set in the context of the general problem of integrating logic and probability, a long-standing problem of major importance that is currently attracting substantial interest. There are three main threads to the study of the integration of logic and probability. The oldest by far is the philosophical thread that can be traced via Boole [Boo54, Boo52] back to Jacob Bernouilli in 1713. An extensive historical account of this thread can be found in [Hai96] and overviews of more recent work in [Háj01, Wil02]. The second thread is that of the knowledge representation and reasoning community in artificial intelligence, of which [Nil86, Hal89, FH94, Hal03, SA07] are typical works. The third thread is that of the machine learning community in artificial intelligence, of which [Mug96, DK03, MMR$^+$05, RD06, MR07, KD07] are typical works. The approach of this paper is mostly distinguished from all the works mentioned above by its setting in a more expressive logic and the exploitation of the higher-orderness of the logic for representing densities in theories.

9 Conclusion

This paper studied knowledge representation and reasoning in a multi-modal, polymorphically-typed, higher-order logic. A detailed account of the syntax and semantics of the logic, and a reasoning system that combines computation and deduction were given. Applications, especially to agents, were discussed.

There is plenty of scope for future work. For example, on the theoretical side, it would be reassuring to have a completeness theorem for the theorem-proving component of the reasoning system. Another theoretical issue concerns the correctness of agent systems. What does it mean for an agent, which may be adaptive and therefore change its behaviour according to changed conditions, to be proved correct? Presumably, what has to be established are certain invariants of the behaviour of the agent that guarantee its behaviour stays within acceptable bounds even under adaptation. On the practical side, more sophisticated agent applications and other kinds of applications need to be developed to more convincingly demonstrate the effectiveness and practicality of the logic. A prototype implementation of the reasoning system described in the paper intended just for testing out ideas already exists, but it would benefit from further work.

Acknowledgments

I greatly appreciate innumerable discussions with Kee Siong Ng, with whom I have collaborated over a number of years and who has greatly influenced the development of the entire body of work related to this paper. Many thanks are due to Joshua Cole and Rajeev Goré for helpful discussions. Kee Siong Ng was chiefly responsible for bringing to life the learning system Alkemy and the reasoning systems Escher and Bach, whose theoretical foundations are provided by this paper. Also Joshua Cole, Matt Gray, and Joel Veness contributed to several systems.

References

[And86] P.B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Academic Press, 1986.

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

[Bil86] P. Billingsley. *Probability and Measure.* Wiley, second edition, 1986.

[Boo54] G. Boole. *An Investigation of the Laws of Thought on which are founded the Mathematical Theories of Logic and Probabilities.* Walton and Maberly, 1854.

[Boo52] G. Boole. *Studies in Logic and Probability.* Watts & Co, 1952.

[CGLN05] J.J. Cole, M.J. Gray, J.W. Lloyd, and K. S. Ng. Personalisation for user agents. In F. Dignum and *et al*, editors, *Fourth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 05)*, pages 603–610, 2005.

[DK03] L. De Raedt and K. Kersting. Probabilistic logic learning. *SIGKDD Explorations*, 5(1):31–48, 2003.

[DM79] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[Dud02] R.M. Dudley. *Real Analysis and Probability*. Cambridge University Press, 2002.

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

[EIM02] Thomas Eiter, Toshihide Ibaraki, and Kazuhisa Makino. Decision lists and related boolean functions. *Theoretical Computer Science*, 270(1-2):493–524, 2002.

[FH94] R. Fagin and J.Y. Halpern. Reasoning about knowledge and probability. *Journal of the ACM*, 41(2):340–367, 1994.

[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.

[Ger01] M. Gergatsoulis. Temporal and modal logic programming languages. In A. Kent and J.G. Williams, editors, *Encyclopedia of Microcomputers*, volume 27, supplement 6, pages 393–408. Marcel Dekker, 2001.

[GKWZ03] D.M. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyaschev. *Many-Dimensional Modal Logics: Theory and Applications*. Studies in Logic and The Foundations of Mathematics, Volume 148. Elsevier, 2003.

[GM93] M.J.C. Gordon and T.F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[Hai96] T. Hailperin. *Sentential Probability Logic*. Lehigh University Press, 1996.

[Háj01] A. Hájek. Probability, logic and probability logic. In L. Goble, editor, *The Blackwell Guide to Philosophical Logic*, chapter 16, pages 362–384. Blackwell, 2001.

[Hal89] J.Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1989.

[Hal03] J.Y. Halpern. *Reasoning about Uncertainty*. MIT Press, 2003.

[Han94] M. Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 19&20:583–628, 1994.

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

[Jon03] S. Peyton Jones. *Haskell 98 Language and Libraries*. Cambridge University Press, 2003.

[KD07] K. Kersting and L. De Raedt. Bayesian logic programming: Theory and tool. In L. Getoor and B. Taskar, editors, *Introduction to Statistical Relational Learning*. MIT Press, 2007.

[Llo99] J.W. Lloyd. Programming in an integrated functional and logic language. *Journal of Functional and Logic Programming*, 1999(3), March 1999.

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

[LN07a] J.W. Lloyd and K. S. Ng. Learning modal theories. In S. Muggleton, R. Otero, and A. Tamaddoni-Nezhad, editors, *Proceedings of the 16th International Conference on Inductive Logic Programming (ILP 2006)*, pages 320–334. Springer, LNAI 4455, 2007.

[LN07b] J.W. Lloyd and K. S. Ng. Probabilistic and logical beliefs. In M. Dastani, J. Leite, A. El Fallah Seghrouchni, and P. Torroni, editors, *Proceedings of the Workshop on Languages, Methodologies and Development Tools for Multi-agent Systems (LADS'007)*, 2007.

[LN07c] J.W. Lloyd and K. S. Ng. Reflections on agent beliefs. In M. Baldoni, T.C. Son, M. Birna van Riemsdijk, and M. Winikoff, editors, *Proceedings of the Workshop on Declarative Agent Languages and Technologies (DALT 2007)*, pages 99–114, 2007.

[LS06] J.W. Lloyd and T.D Sears. An architecture for rational agents. In M. Baldoni and *et al*, editors, *Declarative Agent Languages and Technologies (DALT 2005)*, pages 51–71. Springer, LNAI 3904, 2006.

[MMR+05] B. Milch, B. Marthi, S. Russell, D. Sontag, D.L. Ong, and A. Kolobov. BLOG: Probabilistic models with unknown objects. In L.P. Kaelbling and A. Saffiotti, editors, *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1352–1359, 2005.

[MR07] B. Milch and S. Russell. First-order probabilistic languages: Into the unknown. In S. Muggleton, R. Otero, and A. Tamaddoni-Nezhad, editors, *Proceedings of the 16th International Conference on Inductive Logic Programming*, 2007.

[Mug96] S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.

[Musar] R. Muskens. Higher order modal logic. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*. Elsevier, to appear.

[Ng05a] K. S. Ng. *Learning Comprehensible Theories from Structured Data*. PhD thesis, Computer Sciences Laboratory, The Australian National University, 2005.

[Ng05b] K.S. Ng. Generalization behaviour of alkemic decision trees. In *Inductive Logic Programming, 15th International Conference (ILP 2005)*, pages 246–263. Springer, LNAI 3625, 2005.

[Ng06] K.S. Ng. (Agnostic) PAC learning concepts in higher-order logic. In *European Conference on Machine Learning (ECML 2006)*, pages 711–718. Springer, LNAI 4212, 2006.

[Ngu06] L.A. Nguyen. Multimodal logic programming. *Theoretical Computer Science*, 360:247–288, 2006.

[Nil86] N.J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–88, 1986.

[NM98] G. Nadathur and D.A. Miller. Higher-order logic programming. In D.M. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *The Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 499–590. Oxford University Press, 1998.

[OM94] M.A. Orgun and W. Ma. An overview of temporal and modal logic programming. In D.M. Gabbay and H.J. Ohlbach, editors, *Proceedings of the First International Conference on Temporal Logics (ICTL'94)*, volume 827 of *Lecture Notes in Artificial Intelligence*, pages 445–479. Springer, 1994.

[Put94] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[RD06] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

[Riv87] R. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[RN02] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, second edition, 2002.

[SA07] A. Shirazi and E. Amir. Probabilistic modal logic. In R.C. Holte and A. Howe, editors, *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pages 489–495, 2007.

[TBF05] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005.

[Tho99] S. Thompson. *Haskell: The Craft of Functional Programming*. Addison-Wesley, second edition, 1999.

[Was04] L. Wasserman. *All of Statistics*. Texts in Statistics. Springer, 2004.

[Wil02] J. Williamson. Probability logic. In D. Gabbay, R. Johnson, H.J. Ohlbach, and J. Woods, editors, *Handbook of the Logic of Inference and Argument: The Turn Toward the Practical*, volume 1 of *Studies in Logic and Practical Reasoning*, pages 397–424. Elsevier, 2002.

[Woo97] M. Wooldridge. Agent-based software engineering. *IEE Proceedings on Software Engineering*, 144(1):26–37, 1997.

A Induction

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 preceding 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, $A = X$. □

B Probability

This appendix presents some standard notions of measure theory [Dud02, Bil86], particularly that of a density.

Definition B.1. Let X be a set. A collection $\mathcal{A} \subseteq 2^X$ is called a *σ-algebra* on X if $\emptyset \in \mathcal{A}$, and $\mathcal{A}$ is closed under complementation and countable unions.

Definition B.2. A *measurable space* is a pair $(X, \mathcal{A})$, where X is a set and $\mathcal{A}$ is a σ-algebra on X

Definition B.3. Let $(X, \mathcal{A})$ and $(Y, \mathcal{B})$ be measurable spaces. A function $f : X \to Y$ is *measurable* if $f^{-1}(B) \in \mathcal{A}$, for all $B \in \mathcal{B}$.

Definition B.4. Let $(X, \mathcal{A})$ be a measurable space. A function $\mu : \mathcal{A} \to [0, \infty]$ is a *measure* if it is countably additive (that is, if $\{B_i\}_{i=1}^{\infty}$ is a sequence of disjoint sets in $\mathcal{A}$, then $\mu(\cup_{i=1}^{\infty} B_i) = \sum_{i=1}^{\infty} \mu(B_i)$).

A measure μ is a *probability* if $\mu(X) = 1$.

Definition B.5. A *measure space* is a triple $(X, \mathcal{A}, \mu)$, where $(X, \mathcal{A})$ is a measurable space and μ is a measure on $(X, \mathcal{A})$.

A *probability space* is a triple $(X, \mathcal{A}, \mu)$, where $(X, \mathcal{A})$ is a measurable space and μ is a probability on $(X, \mathcal{A})$.

A measurable function defined on a measure space induces a measure on its codomain. (Note that $(f \circ g)(x)$ means $g(f(x))$.)

Proposition B.1. *Let $(X, \mathcal{A}, \mu)$ be a measure space, $(Y, \mathcal{B})$ a measurable space and $f : X \to Y$ a measurable function. Then $f^{-1} \circ \mu$ is a measure on $\mathcal{B}$, so that $(Y, \mathcal{B}, f^{-1} \circ \mu)$ is a measure space.*

Next comes the definition of a random variable.

Definition B.6. A *random variable* is a measurable function $f : X \to Y$, where $(X, \mathcal{A}, \mu)$ is a probability space and $(Y, \mathcal{B})$ is a measurable space. The probability measure $f^{-1} \circ \mu$ on $\mathcal{B}$ is called the *distribution* (or *law*) of f.

Usually random variables are real-valued, that is, their codomain is $\mathbb{R}$ with Lebesgue measurable sets.

The next result is the Radon-Nikodym theorem [Dud02, Theorem 5.5.4]. For that, the definition of absolute continuity is needed.

Definition B.7. Let ν and μ be two measures on the same measurable space. Then ν is *absolutely continuous* with respect to μ if $\nu(A) = 0$, whenever $\mu(A) = 0$.

Proposition B.2. *Let $(X, \mathcal{A}, \mu)$ be a measure space, where μ is σ-finite, and ν a finite measure on $\mathcal{A}$ that is absolutely continuous with respect to μ. Then there exists a non-negative, integrable function h on X such that*

$$\nu(A) = \int_A h \, d\mu,$$

for all $A \in \mathcal{A}$.

A key concept for probabilistic reasoning is that of a density.

Definition B.8. Let $(X, \mathcal{A}, \mu)$ be a measure space and $f : X \to \mathbb{R}$ a measurable function. Then f is a *density (on* $(X, \mathcal{A}, \mu)$*)* if (i) $f(x) \geq 0$, for all $x \in X$, and (ii) $\int_X f \, d\mu = 1$.

If $(X, \mathcal{A}, \mu)$ is a measure space, then *Density* X denotes the set of densities on $(X, \mathcal{A}, \mu)$.

There are two main cases of interest. The first is when μ is the counting measure on X, in which case $\int_X f \, d\mu = \sum_{x \in X} f(x)$; this is the discrete case. The second case is when X is $\mathbb{R}^n$, for some $n \geq 1$, and μ is Lebesgue measure; this is the continuous case. The mixed case also arises where a product space has some discrete and some continuous components.

A density f gives a probability ν on $\mathcal{A}$ by the definition

$$\nu(A) = \int_A f \, d\mu,$$

for $A \in \mathcal{A}$. In the common discrete case, this definition specialises to

$$\nu(A) = \sum_{x \in A} f(x).$$

Definition B.9. Let $(X, \mathcal{A}, \mu)$ be a measure space, h a density on $(X, \mathcal{A}, \mu)$, and $f : X \to \mathbb{R}$ a random variable. Then the *expectation* of f with respect to h, $\mathbb{E}_h(f)$, is defined by

$$\mathbb{E}_h(f) = \int_X f(x) \times h(x) \, d\mu(x).$$

Some (higher-order) functions that operate on densities will be needed. The following gives the natural 'composition' $\natural$ of two functions whose codomains are densities.

Definition B.10. Let $(X, \mathcal{A}, \mu)$, $(Y, \mathcal{B}, \nu)$, and $(Z, \mathcal{C}, \xi)$ be measure spaces. The function $\natural : (X \to \textit{Density } Y) \to (Y \to \textit{Density } Z) \to (X \to \textit{Density } Z)$ is defined by

$$(f \natural g)(x)(z) = \int_Y f(x)(y) \times g(y)(z) \, d\nu(y),$$

for $f : X \to \textit{Density } Y$, $g : Y \to \textit{Density } Z$, $x \in X$, and $z \in Z$.

Specialised to the discrete case, the definition is

$$(f \natural g)(x)(z) = \sum_{y \in Y} f(x)(y) \times g(y)(z).$$

Intuitively, this definition captures the intended meaning of $\natural$. The meaning of the left hand side is the probability that $(f \natural g)$ maps x to z. Each component of the sum on the right hand side is the product of the probability that f takes x to y multiplied by the probability that g takes y to z; now we sum over all possible y to get the probability that x is mapped to z.

Note that $(f \natural g)(x)$ *is* a density:

$$\begin{aligned}
&\int_Z (f \natural g)(x)(z)\ d\xi(z) \\
&= \int_Z \int_Y f(x)(y) \times g(y)(z)\ d\nu(y)\ d\xi(z) \\
&= \int_Y \int_Z f(x)(y) \times g(y)(z)\ d\xi(z)\ d\nu(y) \\
&= \int_Y f(x)(y) \times (\int_Z g(y)(z)\ d\xi(z))\ d\nu(y) \\
&= \int_Y f(x)(y)\ d\nu(y) \\
&= 1,
\end{aligned}$$

by Fubini's theorem, under weak conditions [Dud02, Theorem 4.4.5].

The composition of two functions, the first of which has a codomain of densities can be similarly defined.

Definition B.11. The function

$$\flat : (X \to \mathit{Density}\ Y) \to (Y \to Z) \to (X \to \mathit{Density}\ Z)$$

is defined by

$$(f \flat g)(x)(z) = \int_{g^{-1}(z)} f(x)(y)\ d\nu(y),$$

where $f : X \to \mathit{Density}\ Y$, $g : Y \to Z$, and $x \in X$.

Specialised to the discrete case, the definition is

$$(f \flat g)(x)(z) = \sum_{y \in g^{-1}(z)} f(x)(y).$$

Intuitively, this definition captures the intended meaning of $\flat$. The meaning of the left hand side is the probability that $(f \flat g)$ maps x to z. Each component of the sum on the right hand side is the probability that f takes x to y; now we sum over all y that map to z under g to get the probability that x is mapped to z.

Note that $(f \flat g)(x)$ *is* a density:

$$\begin{aligned}
&\int_Z (f \flat g)(x)(z)\ d\xi(z) \\
&= \int_Z \int_{g^{-1}(z)} f(x)(y)\ d\nu(y)\ d\xi(z) \\
&= \int_Y f(x)(y)\ d\nu(y) \\
&= 1.
\end{aligned}$$

There is a sense in which $\flat$ is a special case of $\natural$. To see this, consider a function $g : Y \to Z$. Associated with g is a function $g' : Y \to \mathit{Density}\ Z$ defined by

$$g'(y) = \chi_{g(y)},$$

where $\chi_{g(y)}$ is the characteristic function defined to be 1 at $g(y)$ and 0 elsewhere. The function g' is intuitively 'isomorphic' to g. Then, for $f : X \to \mathit{Density}\ Y$ and $g : Y \to Z$, it follows that

$$f \natural g' = f \flat g.$$

The proof of this follows.

$$\begin{aligned}
&(f \natural g')(x)(z) \\
&= \int_Y f(x)(y) \times g'(y)(z)\ d\nu(y) \\
&= \int_Y f(x)(y) \times \chi_{g(y)}(z)\ d\nu(y) \\
&= \int_{g^{-1}(z)} f(x)(y) d\nu(y) \\
&= (f \flat g)(x)(z).
\end{aligned}$$

Definition B.12. The function

$$\S : \mathit{Density}\ Y \to (Y \to \mathit{Density}\ Z) \to \mathit{Density}\ Z$$

is defined by

$$(f\ \S\ g)(z) = \int_Y f(y) \times g(y)(z)\ d\nu(y),$$

where $f : \mathit{Density}\ Y$, $g : Y \to \mathit{Density}\ Z$, and $z \in Z$.

Specialised to the discrete case, the definition is

$$(f\ \S\ g)(z) = \sum_{y \in Y} f(y) \times g(y)(z).$$

Definition B.13. The function

$$\$: \mathit{Density}\ Y \to (Y \to Z) \to \mathit{Density}\ Z$$

is defined by

$$(f\ \$\ g)(z) = \int_{g^{-1}(z)} f(y)\ d\nu(y),$$

where $f : \mathit{Density}\ Y$, $g : Y \to Z$, and $z \in Z$.

Specialised to the discrete case, the definition is

$$(f \$ g)(z) = \sum_{y \in g^{-1}(z)} f(y).$$

Note that $\flat$, \$, and § are all essentially special cases of $\natural$. Let $\star$ be a distinguished singleton set. Then § : *Density* $Y \to (Y \to$ *Density* $Z) \to$ *Density* Z, for example, can be identified with a function

$$\natural : (\star \to \textit{Density } Y) \to (Y \to \textit{Density } Z) \to (\star \to \textit{Density } Z)$$

in the obvious way.

Another useful way of constructing densities is as follows. Suppose that $(X, \mathcal{A}, \mu)$ is a measure space, $(Y, \mathcal{B})$ is a measurable space and $f : X \to Y$ a measurable function. Then $(Y, \mathcal{B}, f^{-1} \circ \mu)$ is a measure space. Furthermore, if h is a density on $(Y, \mathcal{B}, f^{-1} \circ \mu)$, then $f \circ h$ is a density on $(X, \mathcal{A}, \mu)$ because, by [Dud02, Theorem 4.1.11],

$$\int_X f \circ h \, d\mu = \int_Y h \, d(f^{-1} \circ \mu).$$

Consequently, one can define a function

$$\overline{f} : \textit{Density } Y \to \textit{Density } X$$

by

$$\overline{f}(h) = f \circ h,$$

for each $h \in$ *Density* Y.

One can define conditional densities. Consider a function $f :$ *Density* $X \times Y$ that defines a product density. Then the conditional density obtained by conditioning on values in X can be expressed by the function $f_1 : X \to$ *Density* Y defined by

$$f_1(x)(y) = \frac{f(x,y)}{\int_Y f(x,y) \, d\nu(y)},$$

for $x \in X$ and $y \in Y$. Clearly, $f_1(x)$ *is* a density. Conditioning on the other argument is analogous to this. Specialised to the discrete case, the definition is

$$f_1(x)(y) = \frac{f(x,y)}{\sum_{y \in Y} f(x,y)}.$$

Marginal densities can also be defined. Consider a function

$$f : \textit{Density } X \times Y \times Z$$

that defines a product density. Then the marginal density over the first argument can be expressed by the function $f_1 :$ *Density* X defined by

$$f_1(x) = \int_Z \int_Y f(x,y,z) \, d\nu(y) \, d\xi(z),$$

for $x \in X$. By Fubini's theorem, f_1 *is* a density. This is easily extended to marginalising in arbitrary products. Specialised to the discrete case, the definition is

$$f_1(x) = \sum_{z \in Z} \sum_{y \in Y} f(x,y,z).$$

Index