Fachrichtung 6.2 – Informatik
Naturwissenschaftlich-Technische Fakultät I
Mathematik und Informatik
Universität des Saarlandes

Hybrid Logic Revisited

Bachelor's Thesis

Moritz Hardt

Angefertigt unter der Leitung von
Prof. Dr. Gert Smolka

Saarbrücken, April 2006

Verfasser:	Moritz Hardt
Erstgutachter:	Prof. Dr. Gert Smolka
Zweitgutachter:	Prof. Dr. Bernd Finkbeiner

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Moritz Hardt
Saarbrücken, den 21. April 2006

Abstract

We present modal logic on the basis of the simply typed lambda calculus with a system of equational deduction. Combining first-order quantification and higher-order syntax, we can maintain modal reasoning on the basis of classical logic by remarkably simple means. Such an approach has been broadly uninvestigated, even though it has notable advantages, especially in the case of Hybrid Logic.

Hybrid logics extend modal languages with the ability to name and identify states in a model, features that have received much attention in many branches of modal logic over the past years. We give natural characterizations of $\mathcal{HL}(@, \downarrow)$ and its decidable subset $\mathcal{HL}(@)$.

We develop a tableau-like decision procedure for our equivalent of $\mathcal{HL}(@)$. Our algorithm guarantees termination by local criteria in contrast to previous decision procedures. With regards to deduction, we simplify in particular the treatment of identities. Traditionally, modal tableau calculi either rely on external labeling mechanisms or have to represent access information by equivalent formulas. In our system labeling and access information are both internal and explicit.

Acknowledgments

Once an uninitiated student not only in the field of logics, I was fortunate to find an advisor and a teacher like Gert Smolka. His great investment of time and rigor substantially improved my understanding of the subject matter. While his confidence was encouraging, his honest and outspoken remarks reminded me of the work ahead. I am very grateful to him.

I would also like to thank Bernd Finkbeiner for spontaneously refereeing this thesis.

Contents

Chapter 1

Introduction

Overview

Modal logic emerged early as a philosophical discipline to reason about concepts such as possibility, necessity and temporal phenomena. It appeared in symbolic logic with the 1918 work of Lewis where subsequent research in the 1930's was driven by syntactic analysis. Formal semantics were established by Kripke and others in the mid 1960's [37, 38], allowing modal operators to be interpreted over arbitrary relational structures. From the 1970's on, this opened the doors for computer science to discover the strength of modal logic in a wide range of applications such as the temporal verification of transition systems as proposed by Pnueli [41, 42]. As of today, modal logic is actively employed in the field of software verification, databases, computational linguistics and information retrieval.

When explaining the features of a modal logic, modal logicians stress the point that these languages support an *internal* view on a given relational structure, while on the contrary classical logic employs *external* mechanisms such as quantification and variable-binding.[1] Consequently, modal logics deserve special-purpose syntax and semantics which capture this essential idea. Nevertheless, the connections between modal and classical logic are quite intimate. In fact, "first-order correspondence" theory has been an integral part of research in modal logics leading to some of its best known results such as the discovery of the "guarded fragment" [2].

Moreover, in the past years, modern formulations of *Hybrid Logic* arose in modal, temporal and description logic. Hybrid languages extend modal logics with the ability to *name* and *identify* states in a model. The hybrid formula $\downarrow x.\Diamond x$, for example, introduces a name x for the current point of evaluation and demands x to be reachable from itself. Apparently, this formula involves a variable and a variable binder, concepts habitual in classical logic.

[1]See for example p.xii f. in [16].

We develop a formulation of modal logic based on the simply typed lambda calculus [20] with a system of equational deduction [47]. In its untyped formulation, the lambda calculus was introduced by Church in 1936. For our purposes, the lambda calculus provides coherent syntax and standard semantics on top of which we challenge the question:

How do modal logics *concur* with classical logic?

It turns out, combining first-order quantification and higher-order syntax, Kripke semantics can be fully internalized in our system, resulting in a powerful treatment of modal logics in terms of classical logic. With a few key definitions, syntax, semantics and deduction is set up for modal logics. While we make a commitment to classical logic, we maintain the "local perspective" of modal logic in our native syntax. Even more so, traditional modal syntax is preserved on a notational level. This way, standard translations which recursively eliminate modal syntax in favor of less suggestive quantified formulas are obsolete.

Along the line, our approach remedies the problems with β-conversion as they appear in the work of Fitting [26, 25] who gives examples of β/η-equal modal terms denoting differently – clearly in conflict with the lambda calculus. In order to avoid phenomena like these, we make *names* for points of evaluation explicit, even in the case of our minimal modal logic $\mathcal{MF}$. In fact, the lack of names is widely regarded as a major drawback of $\mathcal{K}$, the traditional minimal modal logic [4]. On that account, Hybrid Logic is considered the inevitable remedy. In a standard version, often referred to as $\mathcal{HL}$ or $\mathcal{HL}(@,\downarrow)$, Hybrid Logic enriches $\mathcal{K}$ with naming, binding and referencing constructions, as well as an implicit identity judgement. [6, 16, 5, 9]. We offer an equivalent characterization of $\mathcal{HL}$ in our system called $\mathcal{MFI}$ which solely introduces the identity predicate to $\mathcal{MF}$.

As $\mathcal{HL}$ is undecidable [49], we are interested in its decidable fragment $\mathcal{HL}(@)$ [7]. Characterizing $\mathcal{HL}(@)$ as the *monadic* fragment of $\mathcal{MFI}$, we develop a tableau-like decision procedure for this logic. Tzakova [50] was the first to state a tableau-based decision procedure. Recently, Bolander and Bräuner [17] extended Tzakova's system by a treatment of universal modalities and a strengthened proof of termination by means of loop-checking techniques. Not a decision procedure, but a tableau calculus is given by Blackburn [15] who discusses the broad advantages of internalizing labeled deduction in $\mathcal{HL}$. By carefully analyzing the role of identities in monadic $\mathcal{MFI}$, we obtain simple local termination criteria for our algorithm. Also deduction is simple, especially with respect to the identity judgement. Moreover, deduction is fully internal, while we still profit from an explicit access relation available on the object level.

Contributions

This thesis makes the following contributions:

1. In Chapter 3, we give an extensive discussion of modal logic based on the simply typed lambda calculus.
   (a) Higher-order logic allows to internalize Kripke semantics providing syntax, semantics and deduction for modal logics by means of a few key definitions.
   (b) Recursive standard translations are obsolete. Terms of modal logic *are* terms of classical logic in our system.
   (c) We solve the previous problems with β-conversion as in [26]. As such, our work proposes a foundation for formulations of modal logics based on the lambda calculus, e.g., a "Higher-order Modal Logic" as in [25].
   (d) Operators of $\mathcal{HL}$ are redundant in our system. We give a native characterization of $\mathcal{HL}$ called $\mathcal{MFI}$ and prove these fragments equivalent.
   (e) We characterize $\mathcal{HL}(@)$ as the *monadic* fragment of $\mathcal{MFI}$ and prove equivalence.
   (f) An interesting weaker characterization, we call quasi-monadicity and give proof that we can compute an equivalent monadic formula for each quasi-monadic one.
2. In Chapter 4, we investigate the computational aspects of monadic $\mathcal{MFI}$.
   (a) We develop a tableau-like decision procedure for the satisfiability problem of monadic $\mathcal{MFI}$.
   (b) We can simplify especially the treatment of identities as compared to the preceding calculi in [50, 15, 17].
   (c) Our calculus does not depend on any kind of external labeling mechanisms which accompany most modal tableaux. In this regard, we carry forward the arguments of [15]. However, we simplify [15] with regards to how the access relation is represented and maintained.
   (d) Termination is proven by strictly local arguments in contrast to [17] who maintains loop-checking techniques in order to strengthen the termination arguments of [50].

Chapter 2

The Logical Base

2.1 Simply Typed Lambda Calculus

Throughout the course of our investigations, we consider a simply typed lambda calculus where every term has a unique type. Interpretation for terms and types is provided by standard semantics. Informally, "standard models" are structures denoting base types by non-empty sets and functional types by the full function space over the respective types. Constants are mapped to members of the appropriate type.

In our system, logical constants are axiomatized by equations and deduction itself is purely equational. For a full discussion of this topic see [47].

2.2 First-order Predicate Logic

Figure 2.1 fixes the signature for a system of first-order predicate logic which includes derived modal operators. ML will serve as the logical foundation for our inquiries into modal logic (see Chapter 3) and the subsequent design of a decision procedure in Chapter 4.

For a discussion of the axioms and their interpretation, we refer the reader to [47, 36].

Note Functional types $\mathsf{T}\mathsf{T}'$ may be written more suggestively as $\mathsf{T} \to \mathsf{T}'$.

Definition (Variable, Closed Term) For each type there is a countably infinite set of variables Var where we omit type subscripts if possible.

- For a term t, $\mathsf{FV}\, t$ denotes the set of free variables of type V occurring in t. FV defined on sets of terms is understood.
- A term t is called *closed*, if $\mathsf{FV}\, t = \emptyset$.

Figure 2.1: First-order Predicate Logic with Modal Operators

Theory	ML	
Base Types	B, V	
Constants	$0, 1 : \mathsf{B}$	
	$\rightarrow : \mathsf{BBB}$	
	$\forall : (\mathsf{VB})\mathsf{B}$	
	$\doteq : \mathsf{VVB}$	
	$R : \mathsf{VVB}$	
Axioms	$0 \rightarrow p = 1$	
	$1 \rightarrow p = p$	
	$p \vee q = q \vee p$	Commutativity
	$f0 \rightarrow f1 \rightarrow fq = 1$	Boolean Case Analysis
	$\forall(\lambda x.1) = 1$	
	$\forall f \rightarrow fx = 1$	Instantiation
	$x \doteq x = 1$	Reflexivity
	$x \doteq y \rightarrow fx \rightarrow fy = 1$	Replacement
Derived Constants	$\neg x = x \rightarrow 0$	
	$x \vee y = (x \rightarrow y) \rightarrow y$	
	$x \wedge y = \neg(\neg x \vee \neg y)$	
	$\exists f = \neg(\forall(\lambda x.\neg(fx)))$	
	$\Box xf = \forall y.Rxy \rightarrow fy$	
	$\Diamond xf = \exists y.Rxy \wedge fy$	
Notation	$\forall x.t = \forall(\lambda x.t)$	
	$\exists x.t = \exists(\lambda x.t)$	

Definition (Parameter) We assume a countably infinite set Par of constants of type V. A member $a \in \mathsf{Par}$ is called a *parameter*. We denote the set of parameters occurring in a term t by $\mathsf{Par}\, t$. Again, Par is also defined on sets of terms.

Note As a convention,

- we will use the letters p, q for boolean variables.
- we use x, y to refer to variables of type V. Parameters are written as a, b. If left open which is referred to, we use u, v.
- Variables of type VB are written as f, g.

Definition 2.1 (Substitution) We denote by $t[u := v]$ the term obtained from t by replacing all occurrences of u by v. This is defined as usual by recursion on t. Note that u may also be a parameter.

Definition 2.2 (Term Size) The *size* of a term t, denoted by $|t|$ is defined recursively as usual:

$$|c| = 1 \qquad \text{(constant)}$$
$$|x| = 1 \qquad \text{(variable)}$$
$$|tt'| = 1 + |t| + |t'| \qquad \text{(application)}$$
$$|\lambda x.t| = 1 + |t| \qquad \text{(abstraction)}$$

Definition 2.3 (Formula, Formula Convention) A *formula* is a term of type B. An equation $t = 1$ may be written as t for convenience.

Chapter 3

Modal Logic Revisited

In this chapter, we will assume familiarity with the basics of modal logics and their semantical interpretation over relational structures. For a thorough introduction, we refer the reader to [16, 35]. Although focused on the base cases, our discussion applies in a straightforward manner to modal logics in general.

3.1 A Minimal Modal Logic $\mathcal{MF}$

Roughly speaking, a *basic modal language* consists of three components:

- Propositional letters
- Boolean connectives
- A modal operator

Such a language is interpreted relative to a non-empty set of entities, usually called "worlds", "states" or "vertices" and a binary relation on these vertices, e.g., $(\mathbb{N}, \leq)$. In the context of relational semantics, this structure is often referred to as a "frame". In our system, it is available by means of two constants.

- Type constant V
- Relational constant $R : \mathsf{V} \to \mathsf{V} \to \mathsf{B}$

Interestingly, these are the only two components which are free for interpretation in ML. The interpretation of the remaining (logical) constants is determined by the axioms. So, effectively, ML is a system to talk about basic relational structure.

Propositional letters, we model as higher-order variables of type $\mathsf{V} \to \mathsf{B}$ which we will call *propositional variables.*

- Propositional variables $f, g : \mathsf{V} \to \mathsf{B}$

Precisely as in modal logic, propositional variables will be assigned to *properties of vertices.* In this way, they carry contingent information which are not part of the underlying model itself.

For the modal operators, we derive higher-order constants.

- Modal operators $\Diamond, \Box : \mathsf{V} \to (\mathsf{V} \to \mathsf{B}) \to \mathsf{B}$.

By a first look at the type of these constants, we see that they enable judgement about a *vertex* and a *property of vertices.* Thus, it is straightforward to pin down their original meaning in Kripke semantics with the help of two axioms.

$$\Box x f = \forall y. Rxy \to fy$$
$$\Diamond x f = \exists y. Rxy \wedge fy$$

Written in a more suggestive way, these terms enjoy the intuitive reading:

$\Box u(\lambda x.t)$ "At u, all direct successors x satisfy t."

$\Diamond u(\lambda x.t)$ "At u, some direct successor x satisfies t."

Surprisingly, a "traditional" modal syntax is still available on a notational level. For this purpose, we reserve a single fixed variable $\pi : \mathsf{V}$ and think of π as the current point of evaluation in our model. This is the standard technique to mimic the external positional argument in Kripke semantics. In a next step, we *situate* propositions and modal operators at π hiding the free variable by some notation of choice.

Definition 3.1 (Modal Notation, $\mathcal{K}$)

$$\mathring{f} \overset{\text{def}}{=} f\pi \qquad \mathring{\Box} t \overset{\text{def}}{=} \Box\pi(\lambda\pi.t)$$

$$t, t' \in \mathcal{K} \overset{\text{def}}{=} \mathring{f} \mid \neg t \mid t \wedge t' \mid \mathring{\Box} t$$

Note Until stated otherwise, we use the symbols $\Diamond, \vee, \to, 0, 1$ as the usual abbreviations to shorten our discussion.

Example Using equational deduction and well-known quantifier laws, we can deduce validities in $\mathcal{K}$.

$$\begin{aligned}
\mathring{\Box}(\mathring{f} \wedge \mathring{g}) &= \Box\pi(\lambda\pi.f\pi \wedge g\pi) \\
&= \forall x. R\pi x \to (fx \wedge gx) && (\beta) \\
&= \forall x. (R\pi x \to fx) \wedge (R\pi x \to gx) \\
&= (\forall x. R\pi x \to fx) \wedge (\forall x. R\pi x \to gx) \\
&= \Box\pi(\lambda\pi.f\pi) \wedge \Box\pi(\lambda\pi.g\pi) && (\beta) \\
&= \mathring{\Box}\mathring{f} \wedge \mathring{\Box}\mathring{g}
\end{aligned}$$

The role of a single special-purpose variable π is a central concept to understand modal syntax. However, commitment to a single name seems artificial. We have already in the previous example worked with a second variable. So, to the components of a basic modal language, we add a further attribute:

- Vertices $u, v : \mathsf{V}$

In doing so, we arrive at what is the minimal modal fragment of our choice.

Definition 3.2 ($\mathcal{MF}$)

$$t, t' \in \mathcal{MF} = fu \mid \neg t \mid t \wedge t' \mid \Box u(\lambda x.t)$$

$\mathcal{MF}$ directly captures modal notation. On the other hand, there are certain formulas which lack a single-variable equivalent, $fu \wedge \neg fv$ is such an example.

Proposition *$\mathcal{K} \subset \mathcal{MF}$.*

The point is that $\mathcal{MF}$ already delivers certain *naming and binding* capabilities which are absent in $\mathcal{K}$. This leads us very quickly to the consideration of a modal logic which was designed to provide these mechanisms.

3.2 Hybrid Logic, Identity and $\mathcal{MFI}$

The essential advantage of $\mathcal{HL}$ over $\mathcal{K}$ is that it allows to *identify* vertices. Fur this purpose, $\mathcal{HL}$ introduces three new constructions all of which are in our system available in terms of the identity predicate $\doteq : \mathsf{V} \to \mathsf{V} \to \mathsf{B}$.

Definition 3.3 (Hybrid Notation, $\mathcal{HL}$)

$$\mathring{u} \overset{\text{def}}{=} \pi \doteq u \qquad @u.t \overset{\text{def}}{=} (\lambda\pi.t)u \qquad \downarrow x.t \overset{\text{def}}{=} (\lambda x.t)\pi$$

$$t, t' \in \mathcal{HL} \overset{\text{def}}{=} \mathring{f} \mid \mathring{u} \mid \neg t \mid t \wedge t' \mid \mathring{\Box} t \mid @u.t \mid \downarrow x.t$$

- A term $\mathring{u}$ is standardly called a *nominal.*
- @ is called the *satisfaction*-operator, $\downarrow$ has the name *down*-operator.
- $\mathcal{HL}$ is often referred to as $\mathcal{HL}(@, \downarrow)$ to distinguish the fragment $\mathcal{HL}(@)$ which excludes the down-operator. In this case, $u \in \mathsf{Par}$, by convention. For convenience, we also assume terms in $\mathcal{HL}(@)$ to be closed. This can be achieved by prefixing a formula, e.g., t corresponds to $@a.t$ where a does not occur in t.

Let us see how the newly introduced constructions work. Nominals are self-explaining. The @-operator captures all occurrences of π and replaces them by some other vertex. This way, a formula $@u.t$ deserves the reading: "At u, evaluate t." The $\downarrow$-operator simply introduces a name for the current point of evaluation.

Example (Valid Inferences in $\boldsymbol{\mathcal{HL}}$)

$$\begin{aligned} @u.\mathring{v} \wedge @v.\mathring{f} &= u \doteq v \wedge fv && (\beta) \\ &\vdash fu && \text{(Replacement, Weakening)} \\ &= @u.\mathring{f} && (\beta) \end{aligned}$$

Via β-reduction, $\mathcal{HL}$ fluently maps into the following extension of $\mathcal{MF}$.

Definition 3.4 ($\boldsymbol{\mathcal{MFI}}$)

$$t, t' \in \mathcal{MFI} \overset{\text{def}}{=} fu \mid u \doteq v \mid \neg t \mid t \wedge t' \mid \Box u(\lambda x.t)$$

Interestingly, this time an inverse mapping exists, too. While $\mathcal{K}$ and $\mathcal{MF}$ did not match up, we find essentially for every term in $\mathcal{MFI}$ an equivalent one in modal notation.

Proposition 3.1 ($\boldsymbol{\mathcal{MFI} \longrightarrow \mathcal{HL}}$) *Consider the following mapping φ defined by recursion on $t \in \mathcal{MFI}$.*

$$\begin{aligned} \varphi(fu) &= @u.\mathring{f} \\ \varphi(u \doteq v) &= @u.\mathring{v} \\ \varphi(\neg t) &= \neg(\varphi t) \\ \varphi(t \wedge t') &= (\varphi t) \wedge (\varphi t') \\ \varphi(\Box u(\lambda x.t)) &= @u.\mathring{\Box}(\downarrow x.\varphi t) \end{aligned}$$

Then,

1. *$\varphi \in \mathcal{MFI} \to \mathcal{HL}$*

2. *$\forall t \in \mathcal{MFI} : \mathsf{ML} \vdash t = \varphi t$ given that π does not occur free in t.*

Proof. 1. Follows immediately by case analysis.

2. Proof is by induction on t. The base cases work analogously as follows:

$$\varphi(fu) = @u.\mathring{f} = (\lambda\pi.f\pi)u = fu$$

We demonstrate the non-trivial induction step.

$$\begin{aligned}
\varphi(\Box u(\lambda x.t)) &= @u.\mathring{\Box}(\downarrow x.\varphi t) \\
&= @u.\mathring{\Box}(\downarrow x.t) && \text{(Induction Hypothesis)} \\
&= (\lambda\pi.\mathring{\Box}(\downarrow x.t))u \\
&= (\lambda\pi.\Box\pi(\lambda\pi.\downarrow x.t))u \\
&= \Box u(\lambda\pi.\downarrow x.t) && (\beta) \\
&= \Box u(\lambda\pi.(\lambda x.t)\pi) \\
&= \Box u(\lambda x.t) && (\eta,\ \pi \notin \mathsf{FV}t)
\end{aligned}$$

$\dashv$

As we see, $\mathcal{MFI}$ and $\mathcal{HL}$ coincide in a very natural way.

Example (Until) We can express an Until-statement ("g until f") in $\mathcal{MFI}$.

$$t = \Diamond a(\lambda x.fx \wedge \Box a(\lambda y.\Diamond y(\lambda y.y \dot{=} x) \to gy))$$

The intended reading is: "From u, we reach a point where f holds and at all intermediate points, g holds." Utilizing φ and a few steps of simplification, we find:

$$\mathsf{ML} \vdash t = @a.\mathring{\Diamond}(\downarrow x.\mathring{f} \wedge @a.\mathring{\Box}((\mathring{\Diamond}\mathring{x}) \to \mathring{g}))$$

The last term is the often demonstrated equivalent formula in $\mathcal{HL}$.

Figure 3.1: Summary of Hybrid Notation

Reserved variable $\pi : \mathsf{V}$

$$\mathring{f} = f\pi \qquad \mathring{\Box}t = \Box\pi(\lambda\pi.t)$$

$$\mathring{u} = \pi \dot{=} u \qquad @u.t = (\lambda\pi.t)u \qquad \downarrow x.t = (\lambda x.t)\pi$$

$$t, t' \in \mathcal{HL} = \mathring{f} \mid \mathring{u} \mid \neg t \mid t \wedge t' \mid \mathring{\Box}t \mid @u.t \mid \downarrow x.t$$

Characterizing the Decidable Fragment

While Figure 3.1 summarizes the modal notation we have introduced, there is still an important question unanswered. It is well known that $\mathcal{HL}(@, \downarrow)$ is *undecidable*. Of this fact, there is a very comprehensive proof in [49] based on the observation that $\mathcal{HL}$ is a "conservative reduction class" with respect to first-order predicate logic.

On the other hand, $\mathcal{HL}(@)$ is decidable. In fact, the satisfiability problem for $\mathcal{HL}(@)$ is PSPACE-complete as shown in [7]. Conclusively, there must be an important fragment of $\mathcal{MFI}$ corresponding to this computationally mild logic. But, which is it?

The operators @ and $\downarrow$ were defined analogously in terms of λ. They are eliminated by β-reduction. Even more interesting is that we could have already extended $\mathcal{K}$ with @ and $\downarrow$. The corresponding fragment would still be $\mathcal{MF}$. On that account, we must find a different characterization of $\mathcal{HL}(@)$.

This leads us to the concept of *monadicity*.

3.3 The Monadic Fragment $\mathcal{MFI}_1$

The language $\mathcal{HL}(@)$ maintains an obvious pattern known from $\mathcal{K}$. Terms can be represented with only a *single* bound variable. Furthermore, nominals always contain a parameter. This gives rise to the following definition.

Definition 3.5 (Monadic Formula) A formula $t \in \mathcal{MFI}$ is called *monadic*, if every subterm of the form $u \doteq v$ contains a parameter and every subterm of the form $\lambda x.t'$ is closed.

Occurrences of the identity predicate are effectively unary predicates denoting singleton sets.

Definition 3.6 ($\mathcal{MFI}_1$)

$$\mathcal{MFI}_1 \overset{\text{def}}{=} \{t \in \mathcal{MFI} \mid t \text{ monadic and closed}\}$$

We can think of $\mathcal{MFI}_1$ as the one-variable fragment of $\mathcal{MFI}$.

Example The formula $\Box a(\lambda x.\Diamond x(\lambda y.a \doteq x))$ is not monadic. The problematic variable is x. Still, in this case, we can find an equivalent monadic formula $\Box a(\lambda x.a \doteq x \wedge \Diamond x(\lambda y.1))$.

As expected, $\mathcal{MFI}_1$ and $\mathcal{HL}(@)$ coincide. The mapping from $\mathcal{HL}(@)$ to $\mathcal{MFI}_1$ is simply beta-reduction as before. For the opposite direction, we have to distinguish cases with regards to the single variable possibly appearing.

Proposition 3.2 ($\boldsymbol{\mathcal{MFI}_1 \longrightarrow \mathcal{HL}(@)}$) *Consider the following mapping defined recursively on $t \in \mathcal{MFI}_1$.*

$$\begin{aligned}
\varphi(f\pi) &= \mathring{f} \\
\varphi(fa) &= @a.\mathring{f} \\
\varphi(a\doteq b) &= @a.\mathring{b} \\
\varphi(a\doteq\pi) &= \mathring{a} \\
\varphi(\pi\doteq a) &= \mathring{a} \\
\varphi(\neg t) &= \neg(\varphi t) \\
\varphi(t \wedge t') &= (\varphi t) \wedge (\varphi t') \\
\varphi(\Box a(\lambda\pi.t)) &= @a.\mathring{\Box}(\varphi t) \\
\varphi(\Box\pi(\lambda\pi.t)) &= \mathring{\Box}(\varphi t)
\end{aligned}$$

It holds,

1. $\varphi \in \mathcal{MFI}_1 \to \mathcal{HL}(@)$
2. $\forall t \in \mathcal{MFI}_1 : \mathsf{ML} \vdash t = \varphi t$

Proof. 1. We can represent the single bound variable occurring in a formula $t \in \mathcal{MFI}_1$ by π. Thus, φ is defined on all formulas in $\mathcal{MFI}_1$. Since t is closed, so is φt and by a straightforward case analysis, $\varphi t \in \mathcal{HL}(@)$.

2. Structural induction on t as in 3.1. $\dashv$

Figure 3.2: Relationship between $\mathcal{MFI}$ and $\mathcal{HL}$

$$\begin{tikzcd}
\mathcal{K} \arrow[r, phantom, "\subset"] \arrow[d, phantom, "\cap"] & \mathcal{MF} \arrow[d, phantom, "\cap"] \\
\mathcal{HL}(@) \arrow[r, leftrightarrow, "\varphi\ \beta"] \arrow[d, phantom, "\cap"] & \mathcal{MFI}_1 \arrow[d, phantom, "\cap"] \\
\mathcal{HL}(@, \downarrow) \arrow[r, leftrightarrow, "\varphi\ \beta"] & \mathcal{MFI}
\end{tikzcd}$$

3.3.1 Quasi-Monadicity

The definition of monadicity is quite restrictive. In fact, there are closed terms already in $\mathcal{MF}$ which are not monadic:

$$\Box a(\lambda x.\Box x(\lambda y.fy \vee fx))$$

However, $\mathcal{MF}$ generally admits monadic terms. The previous formula, for example, is equivalent to:

$$\Box a(\lambda x.\Box x(\lambda y.fy) \vee fx)$$

We cannot hope for the same result in the case of $\mathcal{MFI}$.

Example Consider, $t = \Diamond a(\lambda x.\Diamond x(\lambda y.y \doteq x))$. Intuitively, the problem is the two-place predicate $\doteq$ which can fix variables below a foreign binder.

The formula t demands a reflexive successor at a. Also, see [48] for a discussion why t is not expressible in $\mathcal{HL}(@)$.

Definition 3.7 (Quasi-Monadicity) We call a formula $t \in \mathcal{MFI}$ *quasi-monadic*, if every subterm of the form $u \doteq v$ contains a parameter.

As it turns out, we can transform every quasi-monadic formula into an equivalent monadic one. In the remainder of this section, we develop a proof of this fact. The idea of this proof is to recursively narrow down the scope of a modal operator until within this scope, there are only bound terms containing no other variable.

In the case of predicate logic, a similar technique is sometimes called "anti-prenexing" – quantifiers are moved "downwards".

Note From here on, we will consider $\vee$ a constant.

Definition (Literal) A *literal* is a formula of the form fu, $\neg(fu)$, $u \doteq v$, $\neg(u \doteq v)$, $\Box ut$ or $\neg(\Box ut)$.

Definition (Conjunctive Normal Form) We say a formula t is in *conjunctive normal form*, if t is a conjunction of a disjunction of literals.

Proposition 3.3 *Given a monadic term, we can compute an equivalent monadic term in conjunctive normal form.*

Proof. Let t be monadic term. We can employ the standard equivalence transformations. That is, we first move all negations towards the literals by repeatedly using the laws of de Morgan and double negation.

$$\neg(\neg p) = p$$
$$\neg(p \wedge q) = \neg p \vee \neg q$$
$$\neg(p \vee q) = \neg p \wedge \neg q$$

In a next step, we use distributivity to move disjunctions below conjunctions.

$$p \vee (q \wedge q') = (p \vee q) \wedge (p \vee q')$$
$$(q \wedge q') \vee p = (q \vee p) \wedge (q' \vee p)$$

We do not repeat this process for subterms of literals. It is easy to see that each of these laws preserves monadicity. $\dashv$

By means of the following equations, we can narrow the scope of a modal operator over a formula in conjunctive normal form.

Proposition 3.4 *The following equations are deducible in* ML*.*

$$\Box x(\lambda y.fy \wedge gy) = \Box xf \wedge \Box xg \qquad (\Box\wedge)$$
$$\Box x(\lambda y.fy \vee q) = \Box xf \vee q \qquad (\Box\vee)$$

Proof. These equations follow from well-known quantifier and boolean laws which are discussed in [47]. We demonstrate the proof of $(\Box\vee)$.

$$\begin{aligned}\Box x(\lambda y.fy \vee q) &= \forall y.Rxy \to (fy \vee q)\\ &= \forall y.(Rxy \to fy) \vee q\\ &= (\forall y.Rxy \to fy) \vee q\\ &= \Box xf \vee q\end{aligned} \qquad \dashv$$

Proposition 3.5 (Separating Literals) *Given a formula*

$$\Box u(\lambda x.l_1 \vee \cdots \vee l_k)$$

where each l_i *is a monadic literal, we can compute an equivalent monadic formula.*

Proof. Let $t = \Box u(\lambda x.l_1 \vee \cdots \vee l_k)$ and $I = \{i \mid (\lambda x.l_i) \text{ is closed}\}$. Now, consider the following formula:

$$t' = \Box u(\lambda x.\bigvee_{i\in I} l_i) \vee \bigvee_{i\notin I} l_i$$

Since each literal is monadic and thus contains at most one variable, we can repeatedly apply $(\Box\vee)$ to obtain $\mathsf{ML} \vdash t = t'$. Moreover, the term $(\lambda x.\bigvee_{i\in I} l_i)$ is closed. As a consequence, t' is monadic. Note that an empty disjunction equals 0 by convention. $\dashv$

Proposition 3.6 (Monadification) *For every quasi-monadic formula, we can compute an equivalent monadic formula.*

Proof. Let t be a monadic formula. We prove by induction on the term structure of t that we can compute a monadic term t' such that $\mathsf{ML} \vdash t = t'$.

In the base cases and the boolean cases, there is nothing to do. So, consider the induction step for $t = \Box u(\lambda x.s)$. By induction hypothesis, we can compute a monadic term s' such that $\mathsf{ML} \vdash s = s'$. Now, we proceed as follows.

1. We compute a monadic conjunctive normal form of s' (Proposition 3.3). This way, $\mathsf{ML} \vdash s' = t_1 \wedge \ldots \wedge t_k$ where each t_i is a disjunction of monadic literals.

2. By repeated application of $(\Box\wedge)$,

$$\mathsf{ML} \vdash t = \Box u(\lambda x.t_1) \wedge \ldots \wedge \Box u(\lambda x.t_k)$$

3. Finally, for each $\Box u(\lambda x.t_i)$, we separate literals (Proposition 3.5) to obtain equivalent monadic terms s'_i.

Now, $s'_1 \wedge \ldots \wedge s'_k$ is a monadic term and it holds,

$$\mathsf{ML} \vdash t = s'_1 \wedge \ldots \wedge s'_k \qquad \dashv$$

As a result, the quasi-monadic fragment is no more expressive than the monadic fragment of $\mathcal{MFI}$. This fact yields a concise characterization of $\mathcal{MFI}_1$. Interestingly, the concept of quasi-monadicity is unknown in modal logics. Traditional modal terms as in $\mathcal{K}$ or $\mathcal{HL}(@)$ are always monadic.

3.4 Alternative Approaches

Ultimately, a strong justification for our previous treatment of modal logics is an analysis of the alternatives.

3.4.1 The Case of Modal Base Syntax

We can try to be more conservative about modal syntax. That is, by looking at a traditional syntax as in $\mathcal{K}$, the modal operator indeed appears as a constant of type $\mathsf{B} \to \mathsf{B}$. Propositional letters have the character of constants (or variables) at type B. So, instead of deriving higher-order constants as we did, we now think of $\mathring{\Box} : \mathsf{B} \to \mathsf{B}$ as a primitive constant. Additionally, we install propositional constants $\mathring{f}, \mathring{g} : \mathsf{B}$. In terms of modal logic this reconstruction seems fairly plausible and resembles closest the traditional treatment of modal syntax. In this manner, it remained to characterize the newly introduced constants by the appropriate axioms or to extend the standard semantics of the lambda calculus with modal frame semantics. In fact, Fitting proceeds this way in [25].

In a rather principal fashion, this approach neglects the repertoire of classical logic available in our system. But more importantly, it rapidly collides with the lambda calculus.

Example (The β/η-Anomalies) Consider the following two terms where q is a variable of type B.

$$t_1 = \mathring{\Box}\mathring{f} \qquad t_2 = (\lambda q.\mathring{\Box} q)\mathring{f}$$

With respect to modal semantics, these terms must denote differently, if $\mathring{f}$ is a "non-rigid" constant. So, we should certainly be able to distinguish these formulas. However, $\lambda q.\mathring{\Box} q$ η-reduces in one step to $\mathring{\Box}$. Even if we restricted or disallowed η-reduction, we would still face the problem that t_2 simply β-equals t_1. So, with respect to the underlying lambda calculus, there is absolutely no justification to discriminate t_1 and t_2. In fact, even the most rudimentary models of the calculus should *not* be able to distinguish these terms. Restricting β-conversion somehow, literally disconnects modal syntax from standard semantics.

It seems quite difficult to solve this problem on the basis of a modal base syntax. Switching back to our notation as in Definition 3.1, it is pleasant to see the complications resolve.

Example (Problems with β/η resolved)

$$t_1 = \mathring{\Box}\mathring{f} \qquad t_2 = (\lambda q.\mathring{\Box} q)\mathring{f}$$

The term t_2 now is a short hand for:

$$(\lambda q.\Box\pi(\lambda\pi.q))(f\pi)$$

But as substitution does *not* capture, this term β-reduces to

$$t_1' = \Box\pi(\lambda\pi'.f\pi)$$

in contrast to

$$t_1 = \Box\pi(\lambda\pi.f\pi)$$

as one might have expected at first glance. We can go further and analyze t_1' deductively in ML.

$$\begin{aligned}\Box\pi(\lambda\pi'.f\pi) &= \Box\pi(\lambda\pi'.0 \vee f\pi) \\ &= \Box\pi(\lambda\pi'.0) \vee f\pi && (\Box\vee) \\ &= \Box\pi(\lambda\pi.0) \vee f\pi \\ &= \mathring{\Box}0 \vee \mathring{f}\end{aligned}$$

The remarkable fact is that *classical reasoning* – as immediately available in our system – clarifies a problem related to modal syntax. But also names played an important role, here.

There are other examples of modal phenomena now enjoying a syntactic explanation. The reader may try to comprehend why in modal logic the "deduction theorem" does not hold, i.e. from $\mathring{f} \vdash \mathring{\Box}\mathring{f}$ we cannot infer $\models \mathring{f} \rightarrow \mathring{\Box}\mathring{f}$. We cannot go into detail at this point.

3.4.2 Standard Translations

The traditional “first-order” approach is to internalize Kripke semantics to *some* degree by means of a “standard translation”. Terms of modal logic are recursively mapped to first-order formulas involving quantifiers. The quantifiers can express modal operators in the obvious way taking care of their semantical interpretation.

Figure 3.3 depicts such a standard translation on an *informal* basis. Here, for the purpose of demonstration, it translates terms of $\mathcal{K}$ to corresponding first-order terms.

Figure 3.3: A First-order Standard Translation

$$\begin{aligned} ST_\pi \mathring{f} &= F\pi \\ ST_\pi(\neg t) &= \neg ST_\pi t \\ ST_\pi(t \wedge t') &= ST_\pi t \wedge ST_\pi t' \\ ST_\pi(\mathring{\Box} t) &= \forall \pi'. R\pi\pi' \rightarrow ST_{\pi'} t \end{aligned}$$

This standard translation involves for each propositional letter $\mathring{f}$ a *constant* $F : \mathsf{V} \rightarrow \mathsf{B}$ and variables π, π' of type V where π' is understood not to have occurred previously in translation.

The first problem of this well-known standard translation are propositional letters. Since higher-order variables are apparently not available in traditional first-order predicate logic, the propositional letters must be translated to unary predicates, i.e., constants. This causes complications with regards to which “frames” these translated formulas define.

The second problem is not of a technical kind, but nonetheless significant. The standard translation eliminates modal syntax and thus the way of reasoning established not only in the applications of modal logic. Still, modal logicians have always relied on the standard translation to seek insight on the basis of classical logic.

In our approach, a standard translation is obsolete. Syntax, semantics and deduction for modal logics on a classical basis are immediately available. But then, as we have seen, reasoning takes place equally on the level of quantifiers, the modal operators and the modal notation.

Chapter 4

Decidability of $\mathcal{MFI}_1$

In this chapter, we devise an algorithm to decide the satisfaction problem of monadic $\mathcal{MFI}$. The central concept of our algorithm is a binary relation $\rightarrow$ on sets of formulas called clauses.

This relation is defined by a set of rules, divided into "don't care" and "don't know" rules. Each application of a rule extends the clause with meaningful information in order to reveal possible contradictions. We call this process *saturation*. While the order in which we apply "don't care" rules is uncritical, the application of a "don't know" rule yields two distinct alternatives. In such a case, we simply do not know which of the two alternatives to proceed with and must try out both.

Essentially, three key properties of this relation will be established.

Soundness If $C \rightarrow D$ don't care, then C is satisfiable if and only if D is satisfiable.

If $C \rightarrow D_1, D_2$ don't know, then C is satisfiable if and only if D_1 is satisfiable or D_2 is satisfiable.

Completeness If C cannot be extended by a saturation rule, then C is satisfiable if and only if C does not contain obvious contradictions.

Termination The relation $\rightarrow$ terminates.

Although we employ some terminology which is not standard, our techniques are known from tableau calculi and tableau-based decision procedures. For introductory material, we refer the reader to [23, 27]. Tableau methods are established in various branches of logics. In particular, they are widely used in modal and temporal logics. In this context, see also [23, 31, 27].

4.1 Preliminaries

In a way that will be convenient for the later analysis, we restate slightly modified versions of $\mathcal{MFI}$ and its syntactic characterizations.

Definition 4.1 ($\mathcal{MFI}$)

$$t, t' \in \mathcal{MFI} \overset{\text{def}}{=} fu \mid u \doteq v \mid Ruv \mid \neg t \mid t \wedge t' \mid t \vee t' \mid \Diamond u(\lambda x.t) \mid \Box u(\lambda x.t)$$

In the following a formula is always assumed to be a member of $\mathcal{MFI}$ in negation normal form. That is, all negations occur in front of formulas of the form fu, $u \doteq v$, Ruv.

Just as before, a monadic formula can be represented with a single variable and contains a parameter in each identity.

Definition 4.2 (Proper, Monadic Term) A formula t is called

- *proper*, if it does not contain any subterm of the form Ruv.
- *quasi-monadic*, if it is proper and every sub-term of the form $u \doteq v$ contains a parameter.
- *monadic*, if it is quasi-monadic, $|\mathsf{FV}t| \leq 1$ and every subterm of the form $\lambda x.t'$ is closed.

Definition 4.3 ($\mathcal{MFI}_1$) We repeat the definition of $\mathcal{MFI}_1$ as in 3.6.

$$\mathcal{MFI}_1 \overset{\text{def}}{=} \{t \in \mathcal{MFI} \mid t \text{ monadic and closed}\}$$

Definition 4.4 (Clause) A *clause* is a finite set of formulas.

Definition 4.5 ((Purely) Monadic Clause) A clause C is called

- *purely monadic*, if $C \subseteq \mathcal{MFI}_1$.
- *monadic*, if all members are either monadic or of the form Ruv.

It is helpful to regard terms of the form Ruv as "edges". In this way, we may think of a monadic clause as the disjoint union of a set of monadic terms and a "graph" component. Later, this graph will turn out to be an indispensable data structure. Now, a purely monadic clause is a monadic clause which comes with closed terms and an empty graph component.

Definition 4.6 (Satisfiability) A formula t (a clause C) is *satisfiable*, if there exists a model $\mathcal{D}$ and an assignment σ such that $\mathcal{D} \models \mathsf{ML}$ and $\mathcal{D}, \sigma \models t$ (for every $t \in C$).

Definition 4.7 (Trivial Clause) A clause C is *trivial*, if one of the following holds:

- $t, \neg t \in C$ for some term t
- $\neg(t \doteq t) \in C$ for some term t

Proposition 4.1 *Trivial clauses are not satisfiable.*

Definition 4.8 (Degree) The degree of a clause C is defined to be the maximum term size in C.

$$\deg C \overset{\text{def}}{=} \begin{cases} 0 & \text{if } C = \emptyset \\ \max_{t \in C} |t| & \text{o.w.} \end{cases}$$

Definition 4.9 We will make use of the following notation.

$$(\lambda x.t) \downarrow u \overset{\text{def}}{=} t[x := u]$$

4.2 Saturatedness and Model Existence

The design space for our later saturation algorithm will be given in this section in terms of saturatedness conditions. The idea is that if a monadic clause respects these closure conditions and we cannot observe an obvious contradiction, then, in fact, this clause must be satisfiable. We formulate this result as a model existence theorem. In doing so, we are using standard techniques to connect syntax and semantics of a logical system. For a full discussion see for example [24, 3].

Definition 4.10 (Saturated Clause) A clause is *saturated*, if it satisfies all of the conditions in Figure 4.1.

Saturatedness is *weak* with respect to identity. ($\mathcal{S}_{\doteq}$) enforces replacement only where parameters are involved. Thus, it is simple to think of a clause which is saturated, but not satisfiable.

Example Consider the clause $\{x \doteq y, fx, \neg fy\}$. We easily verify that this clause is saturated, but not satisfiable. On the other hand, it is also not monadic.

In the remainder of this section, we develop the machinery to finally prove that saturated monadic clauses are satisfiable. Most of this work is concerned with identities.

Figure 4.1: Saturation Conditions

($\mathcal{S}_c$) C is not trivial

($\mathcal{S}_\wedge$) $s \wedge t \in C \implies \{s, t\} \subseteq C$

($\mathcal{S}_\vee$) $s \vee t \in C \implies s \in C$ or $t \in C$

($\mathcal{S}_\Diamond$) $\Diamond ut \in C \implies \{Rux, t{\downarrow}x\} \subseteq C$ for some x

($\mathcal{S}_\Box$) $\Box ut \in C, Ruv \in C \implies t{\downarrow}v \in C$

($\mathcal{S}^s_{\doteq}$) $u \doteq v \in C \implies v \doteq u \in C$

($\mathcal{S}_{\doteq}$) $u \doteq a \in C, t \in C \implies t[u := a] \in C$

Definition 4.11 (Identity Closure) Let C be a clause.

$$\overset{0}{\sim}_C \overset{\text{def}}{=} \{(u,u) \mid u \in \mathsf{FV}C \cup \mathsf{Par}C\}$$
$$\overset{1}{\sim}_C \overset{\text{def}}{=} \{(u,v) \mid u \doteq v \in C \text{ or } v \doteq u \in C\}$$
$$\overset{i+1}{\sim}_C \overset{\text{def}}{=} \overset{i}{\sim}_C \circ \overset{1}{\sim}_C$$
$$\sim_C \overset{\text{def}}{=} \bigcup_{i \geq 0} \overset{i}{\sim}_C$$

We may write $\sim_C$ without subscripts, if C is determined by the context.

Proposition 4.2 *For a clause C, $\sim_C$ is an equivalence relation on* $\mathsf{FV}C \cup \mathsf{Par}C$.

Definition 4.12 ((Trivial) Equivalence Class) For a clause C and $u \in \mathsf{FV}C \cup \mathsf{Par}C$, we write

$$[u]_C \overset{\text{def}}{=} \{v \mid u \sim_C v\}$$

to denote the *equivalence class* of u with respect to $\sim_C$. An equivalence class $[u]_C$ is called *trivial*, if $[u]_C = \{u\}$.

Again, we may write $[u]_C$ without subscripts, if not ambiguous.

Proposition 4.3 (Parameter Existence) *Let C be a monadic clause. Every nontrivial equivalence class $[u]_C$ contains a parameter.*

Proof. Straightforward induction on the definition of $\sim_C$ using monadicity.⊣

The previous proposition justifies the weak replacement condition ($\mathcal{S}_{\doteq}$) in the case of monadicity. If a monadic clause is also saturated, we can prove that the identities in this clause agree with their closure whenever parameters are involved.

Proposition 4.4 (Agreement) *Let C be a saturated monadic clause. Given $u \neq a$, it holds:*

$$u \sim_C a \iff u \dot{=} a \in C$$

Proof. From right to left, this holds directly by definition of $\sim$. So, assume $u \sim a$, then $u \overset{i}{\sim} a$ for some i. We prove by induction on i that in this case $u \dot{=} a \in C$.

The case $i = 0$ is excluded by assumption. Suppose, $u \overset{i+1}{\sim} a$. Then, there is a v such that $u \overset{i}{\sim} v$ and $v \overset{1}{\sim} a$. Since C is saturated, we have $v \dot{=} a \in C$ by $(\mathcal{S}^s_{\dot{=}})$. Now, if $u = v$, we are done. Otherwise, if one of u, v is a parameter, we can apply the induction hypothesis and obtain $u \dot{=} v \in C$. Therefore, $u \dot{=} a \in C$ by using $(\mathcal{S}_{\dot{=}})$.

The problematic case is where $u \neq v$ and both are not parameters. Here, we have a v' such that $u \overset{i-1}{\sim} v'$ and $v' \overset{1}{\sim} v$. Again, $v' \dot{=} v \in C$ and thus by monadicity of C, $v' \in \mathsf{Par}$. Now, we can apply the induction hypothesis to obtain $u \dot{=} v' \in C$. Given these identities, we infer:

$$\begin{aligned} \{u \dot{=} v', v' \dot{=} v, v \dot{=} a\} \subseteq C &\implies \{u \dot{=} v', v' \dot{=} a\} \subseteq C && (\mathcal{S}_{\dot{=}}) \\ &\implies u \dot{=} a \in C && (\mathcal{S}_{\dot{=}}) \end{aligned}$$ ⊣

Theorem 1 (Model Existence) *Every saturated monadic clause is satisfiable in a finite model.*

Proof. Let C be a saturated monadic clause. The only objects of type V subject to the identity judgement are variables and parameters $V \overset{\text{def}}{=} \mathsf{FV}C \cup \mathsf{Par}C$ which are partitioned by $\sim_C$ into equivalence classes.

$$V^{\sim} \overset{\text{def}}{=} \{[u]_C \mid u \in V\}$$

By Parameter Existence (Proposition 4.3), we can assume a function $\rho \in V^{\sim} \to V$ such that:

1. $\rho\{u\} = u$
2. $\rho\nu \in (\nu \cap \mathsf{Par})$ if ν is a nontrivial equivalence class.

Now, we determine a model $\mathcal{D}$ and an assignment σ by the following equations:

$$\begin{aligned} &\mathcal{D}\mathsf{V} = V^{\sim} \\ &\mathcal{D}R = \lambda\nu \in \mathcal{D}\mathsf{V}.\lambda\nu' \in \mathcal{D}\mathsf{V}.R(\rho\nu)(\rho\nu') \in C \\ &\mathcal{D}a = [a] \\ &\sigma x = [x] \\ &\sigma f = \lambda\nu \in \mathcal{D}\mathsf{V}.f(\rho\nu) \in C \end{aligned}$$

$\mathcal{D}$ is chosen to give standard interpretation to the logical constants, i.e. $\mathcal{D} \models \mathsf{ML}$. The value of σ on variables not occurring in C is arbitrary.

While it is easy to prove that $\mathcal{D}$ is a standard model, it remains to show $\mathcal{D}, \sigma \models C$. This is shown by structural induction over the terms in C.

- The base cases follow from Proposition 4.5 with straightforward arguments.

- The induction step is clear in the boolean cases. We demonstrate the case $\Diamond ut \in C$. Since C is saturated, we have $\{Rux, t \downarrow x\} \subseteq C$ for some x. By induction hypothesis $\mathcal{D}, \sigma \models Rux, t{\downarrow}x$ from which we infer $\mathcal{D}, \sigma \models \exists x.Rux \wedge t{\downarrow}x$. Since, $t{\downarrow}x$ β-equals tx, we have $\mathcal{D}, \sigma \models \Diamond ut$. $\dashv$

Proposition 4.5 (Base Cases) *Let C be a saturated monadic clause. Then, it holds:*

1. $u \doteq a \in C$ *or* $a \doteq u \in C \implies [u] = [a]$
2. $\neg(u \doteq a) \in C$ *or* $\neg(a \doteq u) \in C \implies [u] \neq [a]$
3. $fu \in C, u \sim a \implies fa \in C$
4. $\neg(fu) \in C, u \sim a \implies fa \notin C$
5. $Ruv \in C, u \sim a, v \sim b \implies Rab \in C$

Proof. Straightforward use of the saturation conditions and Agreement. In particular, nontriviality of C is needed in the cases 2. and 4. We demonstrate the following.

2. Assume $\neg(a \doteq u) \in C$. Since C is nontrivial, we have $u \neq a$ and also $a \doteq u \notin C$. By $(\mathcal{S}^s_{\doteq})$ contrapositively, $u \doteq a \notin C$. Thus, by Agreement $u \not\sim a$. The case of $\neg(u \doteq a) \in C$ is simpler.

3. Assume $fu \in C$ and $u \sim a$. If $u = a$, we are done. Otherwise, by Agreement, we have $u \doteq a \in C$. With $(\mathcal{S}_{\doteq})$, $fa \in C$.

4. Assume $\neg(fu) \in C$. Again, either directly or by Agreement and $(\mathcal{S}_{\doteq})$, we have $\neg(fa) \in C$. Since C is nontrivial, $fa \notin C$. $\dashv$

The models obtained in Theorem 1 are usually neither "tree" nor "acyclic models". This result cannot be sharpened.

Example Consider the following clause:

$$\{\neg(a \doteq b), \Diamond a(\lambda x.x \doteq b), \Diamond b(\lambda x.x \doteq a)\}$$

Without proof, we claim that this formula cannot be satisfied by a model where R is interpreted by an acyclic graph.

4.3 Saturation

Previously, we have established the notion of a saturated monadic clause along with proof that these clauses are satisfiable. The computational counterpart is now an algorithm which performs saturation steps. We approach this goal by defining a binary relation on clauses by means of a few easily computable rules.

Figure 4.2: Saturation Rules

	If	**and**	**add**
$(\mathcal{C}_\wedge)$	$s \wedge t \in C$	–	s and t
$(\mathcal{C}_\vee)$	$s \vee t \in C$	$s \notin C$ and $t \notin C$	s or t
$(\mathcal{C}_\Diamond)$	$\Diamond ut \in C$	$\Diamond ut$ not expanded in C	$Ru(\gamma C)$ and $t{\downarrow}(\gamma C)$
$(\mathcal{C}_\Box)$	$\Box ut \in C$	$Ruv \in C$	$t{\downarrow}v$
$(\mathcal{C}^s_{\dot=})$	$u\dot=v \in C$	–	$v\dot=u$
$(\mathcal{C}_{\dot=})$	$u\dot=a \in C$	$t \in C$	$t[u := a]$

Definition 4.13 (Expansion) We say $\Diamond ut$ *is expanded in* C if and only if $\{Rux, t{\downarrow}x\} \subseteq C$ for some x.

Definition 4.14 (Saturation) We choose γ to be a fixed mapping from clauses to *variables* such that $\gamma C \notin \mathsf{FV}C$. The saturation relation over clauses is defined as follows:

- $C \to D \overset{\text{def}}{\iff} C \subset D$ and D can be obtained from C by applying one of the rules in Figure 4.2.
- The reflexive, transitive closure of $\to$ is denoted by $\to^*$.
- We say $C \to D$ *don't care*, if $C \to D$ by one of the rules in Figure 4.2 excluding $(\mathcal{C}_\vee)$.
- We say $C \to D_1, D_2$ *don't know*, if D_1 and D_2 are the two distinct alternative results of applying $(\mathcal{C}_\vee)$ to some $s \vee t \in C$.

The property we call "soundness" ensures that satisfiability propagates back and forth over the application of a saturation rule.

Proposition 4.6 (Soundness) *1. If $C \to D$ don't care, then C is satisfiable if and only if D is satisfiable.*

2. *If $C \to D_1, D_2$ don't know, then C is satisfiable if and only if D_1 is satisfiable or D_2 is satisfiable.*

Proof. 1. Assume $C \to D$ don't care and D is satisfiable. Since $C \subseteq D$, C is satisfiable. On the other hand, given that C is satisfiable, we can prove by case analysis over the saturation rules that D is satisfiable. We demonstrate the modal cases.

($\mathcal{C}_\Diamond$) Assume C is satisfiable and $\Diamond ut \in C$. There exist $\mathcal{D}, \sigma$ such that $\mathcal{D} \models \mathsf{ML}$ and $\mathcal{D}, \sigma \models t$ for every $t \in C$. In particular, $\mathcal{D}, \sigma \models \Diamond ut$. Since $\Diamond ut = \exists x.Rux \wedge tx$, there exists a $\nu \in \mathcal{D}\mathsf{V}$ such that $\mathcal{D}, \sigma' \models Ru(\gamma C), t\downarrow(\gamma C)$ where $\sigma' = \sigma[\gamma C := \nu]$. Since $\gamma C \notin \mathsf{FV}C$, we have $\mathcal{D}\sigma t = \mathcal{D}\sigma' t$ for all $t \in C$. Consequently, $\mathcal{D}, \sigma' \models t$ for every $t \in D$. Thus, D is satisfiable.

($\mathcal{C}_\Box$) This is an equivalence transformation, proven in ML as follows:

$$\begin{aligned}
\Box ut \wedge Ruv &= (\forall x.Rux \to tx) \wedge Ruv \\
&= (\forall x.Rux \to tx) \wedge (Ruv \to tv) \wedge Ruv && \text{(Instantiation)} \\
&= (\forall x.Rux \to tx) \wedge Ruv \wedge tv && \text{(Modus Ponens)} \\
&= \Box ut \wedge Ruv \wedge t\downarrow v && (\beta)
\end{aligned}$$

2. The right-to-left implication is proven as before. For the other direction, let C be a satisfiable clause and assume some $s \vee t \in C$. Then, either $C \cup \{s\}$ is satisfiable or $C \cup \{t\}$ is satisfiable. ⊣

Definition 4.15 (Terminal Clause) We call a clause C *terminal*, if there does not exist a clause D such that $C \to D$.

Note, by the side condition in ($\mathcal{C}_\vee$), we ensure that if ($\mathcal{C}_\vee$) applies to a clause, we can perform a don't know step. In other words, if a clause cannot be extended by the application of a don't care or don't know rule, then it is terminal.

Proposition 4.7 (Completeness) *A monadic terminal clause is satisfiable if and only if it is nontrivial.*

Proof. Let C be a monadic terminal clause. If C is trivial, it is not satisfiable (Proposition 4.1).

Assume C is not trivial. Since C is monadic and we have established a model existence theorem for saturated monadic clauses, it remains to show that C is saturated. This is proven by case analysis over the definition of saturatedness.

($\mathcal{S}_c$) By Assumption.

($\mathcal{S}_\Diamond$) Assume $\Diamond ut \in C$. Since C is terminal, $\Diamond ut$ is expanded, i.e., $\{Rux, t\!\downarrow\! x\} \subseteq C$ for some x.

The remaining conditions follow directly from the fact that the respective saturation rule cannot extend C. ⊣

Theorem 2 (Termination) *The saturation relation $\rightarrow$ terminates on purely monadic clauses.*

Proof. See Section 4.4. ⊣

Saturation terminates after finitely many steps on input of a purely monadic clause. But, to make use of our completeness result in this case, the terminal clauses we arrive at must be monadic.

Proposition 4.8 *If $C \rightarrow D$ and C is monadic, then D is monadic.*

Proof. Let C be a monadic clause and assume $C \rightarrow D$. We prove by case analysis over the saturation rules that D is monadic.

($\mathcal{C}_\Diamond$) Given $\Diamond ut \in C$, observe that every sub-term of $t\!\downarrow\!(\gamma C)$ of the form $\lambda x.t'$ is also a sub-term of t and thus closed by assumption. From the fact that t is closed, it also follows that $|\mathsf{FV}(t\!\downarrow\!(\gamma C))| \leq 1$. Since γC replaces a variable in t, all sub-terms of the form $u \doteq v$ still contain a parameter.

($\mathcal{C}_\Box$) Analogous.

($\mathcal{C}_\doteq$) Application of this rule never introduces a variable.

There is nothing to do in the remaining cases. ⊣

Corollary 4.9 (Decidability of $\mathcal{MFI}_1$) *Given a purely monadic clause, we can decide whether this clause is satisfiable or not.*

Proof. A simple procedure $\mathcal{A}$ works as follows:

$\mathcal{A}C = \mathsf{false}$ if C is trivial

$\mathcal{A}C = \mathcal{A}D$ if $C \rightarrow D$ don't care

$\mathcal{A}C = (\mathcal{A}D_1)\ \mathsf{or}\ (\mathcal{A}D_2)$ if $C \rightarrow D_1, D_2$ don't know

$\mathcal{A}C = \mathsf{true}$ if C is a nontrivial terminal clause

Given a purely monadic clause C, we know that $\mathcal{A}$ terminates on input C by Theorem 2. If $\mathcal{A}C = \mathsf{true}$, then by Proposition 4.8 and Completeness C is satisfiable. If $\mathcal{A}C = \mathsf{false}$, then from Soundness it follows that C is not satisfiable. ⊣

The algorithm we have presented is often supported by a data structure called a *tableau.* A tableau for a clause C is a binary tree where each node is a clause. The root of this tree is C itself. Whenever a node C in this tableau has two successors D_1, D_2, we have $C \rightarrow D_1, D_2$ don't know. If a node C has exactly one successor D, then $C \rightarrow D$ don't care. From the saturation relation, we can easily extract tableaux. In the terminology of tableau calculi, saturation is usually called *branch extension.*

A tableau is called *closed*, if all leafs in the tree are trivial clauses. Otherwise, it is *open.* Tableau calculi are also widely used as proof systems. A closed tableau for a clause $\{t\}$ gives proof of the fact that $\neg t$ is valid. On the other hand, if we cannot find a closed tableau for t, a model exists in which t is true, proving that $\neg t$ is not valid. As intended, this notion of a proof works fine for semi-decidable logics.

In the case of a decision procedure we are mainly interested in the saturation relation and its computational behavior.

4.4 Termination

Proof Idea

Remember that we will be given a purely monadic clause C. The degree of this clause does not change when saturation steps are performed and as we have already seen, monadicity is preserved.

$$\begin{array}{rcl} \text{monadic } C & \rightarrow & D \text{ monadic} \\ \deg C & = & \deg D \end{array}$$

Monadicity is important because it helps to address terms with respect to the *single* free variable which possibly occurs. This way, we can partition the proper terms of C into sets C_a, C_x, and so on where C_a is the set of closed proper terms occurring in C.

Now, to proof termination we have to carefully bound the number of variables introduced. As soon as $(\mathcal{C}_\Diamond)$ introduces a fresh variable, we receive a witnessing edge Rux. Interestingly, we can maintain the invariant that in this case, $\deg C_u > \deg C_x$. Since $\deg C_u$ is bounded by $\deg C$ which we mentioned to be invariant, paths in the graph $\{(u, x) \mid Rux \in C\}$ are short.

$$\begin{tikzcd} u \arrow[d, "R"'] & \deg C_u \arrow[d, phantom, "\vee"] \\ x & \deg C_x \end{tikzcd}$$

In a next step, we enforce an injective mapping φ from free variables to "appropriate" terms of the form $\Diamond ut$. This way, every clause C_u can sponsor at most $|C_u|$ free variables. We think of φ as a *justification* for the free variables.

$$\begin{tikzcd}
\Diamond ut \arrow[r, phantom, "\neq"] & \Diamond ut' \\
x \arrow[u, "\varphi"] \arrow[r, phantom, "\neq"] & x' \arrow[u, "\varphi"]
\end{tikzcd}$$

By tracing φ from a given free variable to a parameter, we can establish the notion of a *level*. Each variable is located in some level and from top to bottom, levels make it easy to bound the number of variable recursively.

$$y \xrightarrow{\varphi} \Diamond xt,\ x \xrightarrow{\varphi} \dots \xrightarrow{\varphi} \Diamond at'$$

Proof of Theorem 2

Note We will denote the set of propositional variables occurring in a clause C by $\mathsf{Prop}C$.

Definition Let $\beta \in \mathbb{N}^3 \to \mathbb{N}$ be a function s.t. $|C| \leq \beta(p, v, n)$ whenever C is a clause for which it holds:

1. $|\mathsf{Par}C| + |\mathsf{Prop}C| \leq p$
2. $|\mathsf{FV}C| \leq v$
3. $\mathrm{deg}C \leq n$

Definition Let C be a monadic clause.

- $C_a \overset{\text{def}}{=} \{t \in C \mid t \text{ proper and closed}\}$
- $C_x \overset{\text{def}}{=} \{t \in C \mid t \text{ proper and } \mathsf{FV}t = \{x\}\}$

Definition 4.16 (Admissibility) Let C be a clause, $n \in \mathbb{N}$ and $\varphi \in \mathsf{FV}C \to C$. C is called (n, φ)-admissible, if

1. $\mathrm{deg}C \leq n$ and C monadic
2. $Rux \in C \implies \mathrm{deg}C_u > \mathrm{deg}C_x$
3. φ injective and $\forall x \in \mathsf{FV}C : \exists u, t : \varphi x = \Diamond ut \wedge \{Rux, t{\downarrow}x\} \subseteq C$

A clause C is called

- *n-admissible*, if there exists a $\varphi \in FVC \to C$ such that C is (n, φ)-admissible.
- *admissible*, if there exists an $n \in \mathbb{N}$ such that C is n-admissible.

Definition (Levels) Let C be an (n, φ)-admissible clause. We define recursively:

$$\begin{aligned} L_{\varphi,0} &\overset{\text{def}}{=} \mathsf{Par}C \\ L_{\varphi,m+1} &\overset{\text{def}}{=} \bigcup_{u \in L_{\varphi,m}} S_{\varphi,u} \\ S_{\varphi,u} &\overset{\text{def}}{=} \{x \in \mathsf{FV}C \mid \exists t : \varphi x = \Diamond ut\} \end{aligned}$$

Proposition 4.10 (Levels Are Bounded) *Let C be an (n, φ)-admissible clause with at most p parameters and propositions. Then,*

1. $|C_u| \leq \beta(p, 1, n)$
2. $|S_{\varphi,u}| \leq |C_u|$
3. $|L_{\varphi,n}| \leq p \cdot \beta(p, 1, n)^n$
4. $|L_{\varphi,n+1}| = 0$

Proof.
1. By definition, C_u is a clause with $|\mathsf{FV}C_u| \leq 1$. Since $C_u \subseteq C$, the remaining follows.
2. If $x \in S_{\varphi,u}$, then $\varphi x = \Diamond ut$ for some t. By monadicity, t is closed. Thus, $\varphi x \in C_u$. Now, the claim follows from the injectivity of φ.
3.

$$\begin{aligned} |L_{\varphi,0}| &= |\mathsf{Par}C| \leq p \\ |L_{\varphi,m+1}| &\leq \sum_{u \in L_{\varphi,m}} |S_{\varphi,u}| \\ &\leq \sum_{u \in L_{\varphi,m}} |C_u| && (2) \\ &\leq \sum_{u \in L_{\varphi,m}} \beta(p, 1, n) && (1) \\ &= |L_{\varphi,m}| \cdot \beta(p, 1, n) \end{aligned}$$

   By solving this recurrence, we have $|L_{\varphi,m}| \leq p \cdot \beta(p, 1, n)^m$.

4. If $x \in L_{\varphi,m}$, then there exists a path in the graph $\{(u, x) \mid Rux \in C\}$ of length m from a parameter a to x. This is shown by induction on m. Now it follows from item (1) and (2) of Admissibility that such a path has length at most n. ⊣

Essentially, from the totality of φ, it follows that all free variables are located in some level.

Proposition 4.11 *Let C be an (n,φ)-admissible clause. Then,*

$$\mathsf{FV}C \subseteq \bigcup_{m\geq 1} L_{\varphi,m}$$

Proof. We define recursively:

$$\begin{aligned} l\,a &= 0 \\ l\,x &= 1 + l\,u \quad \text{if } \exists t : \varphi x = \Diamond ut \end{aligned}$$

If $\varphi x = \Diamond ut$, then $Rux \in C$ and therefore $\deg C_u > \deg C_x$. Since $\deg C \leq n$, the definition is admissible. Moreover, it is defined on all free variables in C. Now, proof is by induction on m that $l\,x = m$ implies $x \in L_{\varphi,m}$. ⊣

Corollary 4.12 *Let C be an n-admissible clause with at most p parameters and propositions. Then,*

1. $|\mathsf{FV}C| \leq n \cdot p \cdot \beta(p,1,n)^n$
2. $|C| \leq \beta(p, n\cdot p\cdot \beta(p,1,n)^n, n)$

Lemma 4.13 (Invariance) *Let C be an n-admissible clause. If $C \to D$, then D is n-admissible.*

Proof. Let C be an n-admissible clause and φ be such that C is (n,φ)-admissible. Assume $C \to D$. We prove that D is n-admissible according to the three cases of Admissibility by inspection of the saturation rules.

1. We easily see that $\deg D = \deg C \leq n$. Monadicity follows from Proposition 4.8.

2. $(\mathcal{C}_\wedge)$, $(\mathcal{C}_\vee)$ and $(\mathcal{C}^s_{\doteq})$ are clear. We consider the remaining cases:

   $(\mathcal{C}_\Diamond)$ In this case, there is $\Diamond ut \in C$ such that $D\backslash C = \{Ru(\gamma C), t\!\downarrow\!(\gamma C)\}$. The only terms in D which contain γC are $Ru(\gamma C)$ and $t\!\downarrow\!(\gamma C)$, but both are smaller than $\Diamond ut$. Thus, $\deg D_u > \deg D_{\gamma C}$.

   $(\mathcal{C}_\Box)$ In this case, $\Box ut \in C, Rux \in C$ and $D\backslash C = \{t\downarrow x\}$. However, $|t\!\downarrow\!x| < |\Box ut|$. Therefore, $\deg D_u > \deg D_x$.

   $(\mathcal{C}_{\doteq})$ Assume $t \in C$ and $D\backslash C = \{t[u := a]\}$ was added by $(\mathcal{C}_{\doteq})$. The term $t[u := a]$ cannot increase the degree of some C_x, since a is a parameter.

   However, $(\mathcal{C}_{\doteq})$ may have also added an edge of the form Rax. But in this case, $a \in L_{\varphi,0}$ and $x \in L_{\varphi,m}$ where $m \geq 1$. Thus, $\deg C_a > \deg C_x$ by admissibility of C. Moreover, $\deg D_a = \deg C_a$ and $\deg D_x = \deg C_x$.

3. We must check the case where $(\mathcal{C}_\Diamond)$ introduces a new variable. So, assume $\Diamond ut \in C$ and $D \backslash C = \{Ru(\gamma C), t \downarrow (\gamma C)\}$. We extend φ as follows:

$$\psi x \overset{\text{def}}{=} \begin{cases} \Diamond ut & \text{if } x = \gamma C \\ \varphi x & \text{otherwise} \end{cases}$$

   Now, $\psi \in \mathsf{FV}D \to D$ and we claim that ψ satisfies condition (3) of Admissibility.

   Since $\Diamond ut$ is not expanded in C, there cannot be a $y \in \mathsf{FV}C$ with $\psi y = \Diamond ut = \psi x$. Therefore, ψ is injective, given the injectivity of φ. The remainder follows immediately.

We have shown, D is (n, ψ)-admissible, thus n-admissible. $\dashv$

Proposition *The saturation relation $\to$ terminates on admissible clauses.*

Proof. Let C be n-admissible. Define $B = \beta(p, n \cdot p \cdot \beta(p, 1, n)^n, n)$ where $p = |\mathsf{Par}C| + |\mathsf{Prop}C|$. This is the bound from Corollary 4.12. Note that B is a constant which solely depends on C.

Now, assume $C \to^* D$. By Invariance D is n-admissible. Since saturation neither introduces parameters nor propositions, it follows with Corollary 4.12 that $|D| \leq B$. Thus, paths in $\to$ emanating from C have length at most B as saturation increases clause sizes in each step. $\dashv$

Proposition *Purely monadic clauses are admissible.*

Proof. Let C be a purely monadic clause. It is easy to see that C is $(\deg C, \emptyset)$-admissible. While (1) of Admissibility follows immediately, (2) and (3) are vacuously true. $\dashv$

4.5 Remarks About Saturation

4.5.1 Related Tableau Calculi

The saturation rules in Definition 4.14 directly translate into a tableau calculus. As monadic $\mathcal{MFI}$ is equivalent to $\mathcal{HL}(@)$, our saturation algorithm also decides the satisfiability problem for the latter. Tzakova was the first to introduce a tableau calculus for $\mathcal{HL}(@)$ in [50]. Besides the standard modal and boolean rules, her system comprises four rules concerned with @ and the nominals (identity). In order to achieve termination, Tzakova states an involved special-purpose branch extension procedure.

Shortly after her, Blackburn [15] discusses the broad advantages of internalizing labeled deduction for Hybrid Logic. He stresses the point that nominals should be considered formulas in order to establish labeling discipline at the object level. To handle identities, Blackburn introduces four rules: Reflexivity, Symmetry, Replacement and a rule called "Bridge".

Finally, Bolander and Bräuner [17] extend the calculi of Tzakova and Blackburn by a treatment of the universal modalities and give a simplified discussion. In the case of Tzakova's system, they recognize that two of her rules are subsumed by a strong replacement rule as used by Blackburn. However, such a strong replacement rule demands loop-checking as we will analyze later.

Our calculus certainly carries forward the arguments of Blackburn, as state labels are integral parts of formulas. Blackburn avoids the use of meta-level information and thus has to represent the successor relation Ruv by an equivalent formula like $\Diamond u(\lambda x.x \doteq v)$ or $@u.\mathring{\Diamond}\mathring{v}$, respectively. Moreover, to maintain this representation scheme, he must install the additional rule "Bridge".

$$\frac{@\mathring{u}.\mathring{\Diamond}\mathring{v} \quad @\mathring{v}.\mathring{a}}{@u.\mathring{\Diamond}\mathring{a}}$$

In our system, the access relation itself is a formula. But then, "Bridge" happens to be a special case of $(\mathcal{C}_{\doteq})$:

$$\{Ruv, v \doteq a\} \rightarrow \{\ldots, Rua\}$$

As opposed to Blackburn's system, many modal tableaux as those of Fitting [23] and Gabbay [27] maintain external state labels and access information. It turns out that both of these we can naturally represent on the object level.

Loop-Checking And The Role of $(\mathcal{C}_{\doteq})$

A weak saturation rule with respect to $\doteq$ causes more work in the prelude of our model existence theorem. Computationally, however, a stronger saturation rule leads to divergence very quickly. Consider, for example, a replacement rule $(\mathcal{C}_{\doteq})$ where we omit the requirement $a \in \mathsf{Par}$:

$$(\mathcal{C}_{\doteq}) \qquad \textbf{If} \quad u \doteq v \in C \quad \textbf{and} \quad t \in C \quad \textbf{add} \quad t[u := v]$$

Now, a simple formula causes information to be passed along a growing chain of successors.

$$\begin{aligned}
\{\Diamond u(\lambda x.x \doteq u)\} &\rightarrow \{\ldots, Rux_1, x_1 \doteq u\} && (\mathcal{C}_{\Diamond})\\
&\rightarrow \{\ldots, \Diamond x_1(\lambda x.x \doteq u)\} && (\mathcal{C}_{\doteq})\\
&\rightarrow \{\ldots, Rx_1x_2, x_2 \doteq u\} && (\mathcal{C}_{\Diamond})\\
&\rightarrow \ldots
\end{aligned}$$

Interestingly, this example (written as $\mathring{\Diamond}\mathring{u} \wedge \mathring{u}$) causes the same behavior in the calculus of Bolander and Bräuner [17] where divergence is blocked out by "loop-checking" techniques. While we can avoid loop-checking in the case of identity, our termination criteria cannot handle "universal modalities" without changes. To be precise, item (2) of Admissibility cannot be maintained.

4.5.2 Limitative Observations

Recall that a quasi-monadic term (Definition 4.2) may nest bound variables, but guarantees to have a parameter in each identity. Interestingly, even in the case without identity, this leads to divergence.

Proposition 4.14 (Divergence) *Saturation diverges on quasi-monadic ($\doteq$-free) clauses.*

Proof. Consider the following quasi-monadic formulas.

$$\begin{aligned} t_1 &= \Diamond a(\lambda x.1) \\ t_2 &= \Box a(\lambda x.\Diamond b(\lambda y.fx)) \\ t_3 &= \Box b(\lambda x.\Diamond a(\lambda y.fx)) \end{aligned}$$

We claim that completion diverges on $C = \{t_1, t_2, t_3\}$.

Intuitively, t_1 demands a successor at a. For any successor of a, t_2 demands a successor at b and vice versa by t_3. Now, the critical variable is x. It places whenever $(\mathcal{C}_\Box)$ is applied the fresh variable previously introduced by $(\mathcal{C}_\Diamond)$ inside the inner $\Diamond$-term such that this formula appears unexpanded in C. The sketch is as follows:

$$\begin{aligned} C &\to \{\ldots, Rax_1\} && (\mathcal{C}_\Diamond) \\ &\to \{\ldots, \Diamond b(\lambda y.fx_1)\} && (\mathcal{C}_\Box) \\ &\to \{\ldots, Rbx_2\} && (\mathcal{C}_\Diamond) \\ &\to \{\ldots, \Diamond a(\lambda y.fx_2)\} && (\mathcal{C}_\Box) \\ &\to \{\ldots, Rax_3\} && (\mathcal{C}_\Diamond) \\ &\to \ldots \end{aligned}$$

⊣

We can circumvent this problem by preprocessing, but we are also interested in a native solution.

Open Problem 1 How can the saturation rules be strengthened so that they handle quasi-monadic clauses natively?

A Note On Computational Complexity

Although the bound on the number of variables in Corollary 4.12 *roughly* reads as 2^{n^2}, we cannot hope for anything much better without modification of the saturation algorithm.

Proposition 4.15 *For any number n, there exists a monadic formula t of size polynomial in n such that saturation of $\{t\}$ introduces more than n^n disjoint variables.*

Proof. For a given n, we recursively define the formula t_n of size $c \cdot n^2$ where c is some small constant.

$$\begin{aligned} t_1 &= \Diamond x_1(\lambda y.f_1 y) \wedge \ldots \wedge \Diamond x_1(\lambda y.f_n y) \\ t_n &= \Diamond x_n(\lambda y.f_1 y) \wedge \ldots \wedge \Diamond x_n(\lambda y.f_n y) \wedge \Box x_n(\lambda x_{n-1}.t_{n-1}) \end{aligned}$$

Let $t = t_n[x_n := a]$. Then, $\{t\}$ is a purely monadic clause. When saturating $\{t\}$, n disjoint variables $y_1, \ldots y_n$ are created in order to expand the terms $\Diamond a(\lambda y.f_1 y), \ldots, \Diamond a(\lambda y.f_n y)$. But then, $(\mathcal{C}_\Box)$ will add $t_{n-1} \downarrow y_i$ for each $1 \leq i \leq n$ and so on. $\dashv$

As a result, with respect to space consumption the saturation algorithm is *not* optimal. Monadic $\mathcal{MFI}$ inherits the PSPACE result from $\mathcal{HL}(@)$, while we observe an exponential space consumption at this point.

Nevertheless, the formula stated above is essentially a $\mathcal{K}$ formula and as such it causes the same computational behavior in the calculi we have previously discussed. In fact, since $\mathcal{K}$ does not have the polysize model property [16], any tableau based procedure deciding $\mathcal{K}$ is affected by such examples if it manages resources naively.

For $\mathcal{K}$, however, it is simple to use space in an efficient way by exploring the clause a "diamond at a time". This is not possible in an equally straightforward way when identity is involved.

Open Problem 2 How do we obtain a PSPACE result for $\mathcal{MFI}_1$ based on the saturation algorithm?

Chapter 5

Summary and Conclusion

Chapter 3 In this chapter, we presented modal logic on the basis of the simply typed lambda calculus. Our focus was to give modal logic a uniform and natural treatment in terms of classical logic. On the one hand, first-order quantification was strong enough to express the semantics of the modal operators, on the other hand, first-order predicate logic as such was *syntactically too weak* for our purposes. This is why *higher-order syntax* came to play such an important role. We employed higher-order variables, derived higher-order constants and finally expressed operators of Hybrid Logic by means of the λ-operator.

We eventually arrived at a concept of three different levels of reasoning:

Notational Level On a notational level, we preserved the traditional syntax of modal logics as in $\mathcal{K}$ and $\mathcal{HL}$.

Native Modal Syntax It turned out that modal notation was naturally subsumed by our native syntax and the fragments $\mathcal{MF}$ and $\mathcal{MFI}$ which we defined in terms of this syntax. However, in the case of $\mathcal{HL}$ we obtained a tight equivalence of notation and syntax.

Quantifiers At the bottom, quantifiers were employed to give modal operators their precise meaning.

We demonstrated these levels to interact freely on the basis of equational deduction. Syntax clarified notation. Quantifiers were used several times to derive validities in modal logic.

From our point of view, traditional studies in correspondence between modal and first-order predicate logic suffered from their syntactical weakness.

Chapter 4 Based on our modal syntax, we devised a tableau-like decision procedure for $\mathcal{MFI}_1$, the monadic fragment of $\mathcal{MFI}$. It became evident that the design and the analysis of such an algorithm notably seized upon the rich structure accessible from the object level in our system. Explicit names, identities and an explicit access relation turned out to simplify the style of deduction, considerably.

In a next step, we wanted to know whether termination could be achieved by simple local criteria in our rules and whether the proof of termination itself could be established by local arguments. In the literature, e.g., [17], the argumentation is often that as soon as identities are involved, one faces the same problems as in $\mathcal{K}$ over transitive frames where formulas are passed along a chain of successors. We demonstrated that, in deed, this happens too, if one uses an untamed replacement rule. However, in the case of $\mathcal{MFI}_1$ such a rule can be avoided.

Further Work

There are numerous starting points for further work.

Modal logics in our system Our discussion applies straightforwardly to modal logics in general. One line of research where this seems especially suitable is the exploration of rich modal logics, such as an unrestricted "Higher-order Modal Logic". In spite of the intensional tradition [29, 30] and a recent approach by Fitting [25], such a logic is still widely uninvestigated on the basis of classical higher-order logic, but see [19]. Having established basic modal languages such as $\mathcal{MF}$ and $\mathcal{MFI}$, one can extend these easily by higher-order quantification.

Decision procedures We stated two open problems about our decision procedure. One relates to how we can manage quasi-monadic clauses. Although quasi-monadic formulas are not more expressive than monadic formulas, they enjoy a remarkably simple characterization. We wish to find out how to handle quasi-monadicity, appropriately.

There has not yet been a saturation based decision procedure (such as a tableau calculus) for $\mathcal{MFI}_1$, i.e., $\mathcal{HL}(@)$ which uses space efficiently. As opposed to the situation with $\mathcal{K}$, this is difficult to achieve due to the identity involved. Quite recently, Horrocks et al. [34] demonstrate a goal-directed tableau based decision procedure for the related description logic $\mathcal{SHOIQ}$. We are highly interested to see an optimal PSPACE solution for $\mathcal{MFI}_1$ based on the saturation approach.

We have developed local termination criteria for our algorithm. However, there is a strong indication that one must rely on "loop-checking" as soon as universal modalities are involved. We are interested in further discussion of this topic. It turns out, in order to deal with universal modalities we do not need new constants in our system. Instead, we can generalize the type of $\Diamond, \Box$ to $(\mathsf{VVB})\mathsf{V}(\mathsf{VB})$, i.e., we install an additional relational argument. Our old constants are available as $\Diamond R, \Box R$. Now, ML allows us to build complex relations such as

$$\lambda x.\lambda y.\Diamond Rx(\lambda z.\Diamond Rz(\lambda z.z\doteq y))$$

which reads as "R composed R". Universal modalities can be derived with $\lambda x.\lambda y.1$ and $\lambda x.\lambda y.0$, respectively. As such they are only a special case of large class of modalities, e.g.,

$$\Box(\lambda x.\lambda y.t)xf$$

where t is a monadic term. Generalizations like these give rise to an extended and challenging discussion of termination criteria and decidability results.

Bibliography

[1] R. Alur and T. Henzinger. A Really Temporal Logic. In *30th Annual Symposium on Foundations of Computer Science*, pages 164–169, New York, 1989. IEEE Computer Society Press.

[2] H. Andreka, Istvan Nemeti, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 4:217–274, 1998.

[3] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof.* Kluwer Academic Publishers, 2002.

[4] C. Areces and P. Blackburn. Bringing them all together. *Journal of Logic and Computation*, 11(5):657–669, 2001. Special Issue on Hybrid Logics. Areces, C. and Blackburn, P. (eds.).

[5] C. Areces, P. Blackburn, V. Goranko, M. Marx, and J. Seligman. *The Hybrid Logic Book.* ., 2005. In preparation.

[6] C. Areces, P. Blackburn, and M. Marx. Hybrid logic is the bounded fragment of first order logic. In R. de Queiroz and W. Carnielli, editors, *Proceedings of 6th Workshop on Logic, Language , Information and Computation, WOLLIC99*, pages 33–50, Rio de Janeiro, Brazil, 1999.

[7] C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. In J. Flum and M. Rodríguez-Artalejo, editors, *Computer Science Logic*, number 1683 in LNCS, pages 307–321, Madrid, Spain, 1999. Springer. Proceedings of the 8th Annual Conference of the EACSL, Madrid, September 1999.

[8] C. Areces and M. de Rijke. From description to hybrid logics, and back. In F. Wolter, H. Wansing, M. de Rijke, and M. Zakharyaschev, editors, *Advances in Modal Logic. Volume 3.* CSLI Publications, 2001.

[9] C. Areces and B. ten Cate. Hybrid logics. In P. Blackburn, F. Wolter, and J. van Benthem, editors, *Handbook of Modal Logics.* ., 2005. (In Preparation).

[10] Carlos Areces, Patrick Blackburn, and Maarten Marx. The computational complexity of hybrid temporal logics. *Logic Journal of the IGPL*, 8(5):653–679, 2000.

[11] Franz Baader and Tobias Nipkow. *Term Rewriting And All That*. Cambridge University Press, 1998.

[12] P. Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of the IGPL*, 8(3):339–625, 2000.

[13] P. Blackburn and J. Seligman. Hybrid languages. *Journal of Logic, Language and Information*, 4:251–272, 1995.

[14] P. Blackburn and J. Seligman. What are hybrid languages? In M. Kracht, M. de Rijke, H. Wansing, and M. Zakharyaschev, editors, *Advances in Modal Logic*, volume 1, pages 41–62. CSLI Publications, Stanford University, 1998.

[15] Patrick Blackburn. Internalizing labelled deduction. *Journal of Logic and Computation*, 10(1):137 – 168, 2000.

[16] Patrick Blackburn and Maarten de Rijke. *Modal Logic*. Cambridge Tracks in Theoretical Computer Science. Cambridge University Press, 2001.

[17] Thomas Bolander and Torben Bräuner. *Two Tableau-Based Decision Procedures for Hybrid Logic*. Volume 194 of Schlingloff [46], 2005.

[18] Aldo Bressan. *A General Interpreted Modal Calculus*. Yale University Press, 1972.

[19] Chad E. Brown. Encoding hybrid logic in higher-order logic. http://gtps.math.cmu.edu/cebrown/slides.html, 2005.

[20] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[21] M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga. *Handbook of Tableau Methods*. Kluwer Academic Publishers, 1999.

[22] Brian A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2002.

[23] Melvin Fitting. *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel Publishing Co., Dordrecht, 1983.

[24] Melvin Fitting. *First-order logic and automated theorem proving (2nd ed.)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[25] Melvin Fitting. Higher-order modal logic – a sketch, 2001.

[26] Melvin Fitting and Richard L. Mendelsohn. *First-Order Modal Logic.* Kluwer Academic Publishers, 1998.

[27] Dov Gabbay. *Labelled Deductive Systems.* Oxford University Press, 1996.

[28] Dov M. Gabbay, Ian Hodkinson, and Mark Reynolds. *Temporal Logic.* Oxford, 1994.

[29] Daniel Gallin. *Intensional and Higher-Order Modal Logic*, volume 19 of *North-Holland Mathematics Studies.* North-Holland, Amsterdam, 1975.

[30] L. T. F. [pseud.] Gamut. *Logic, Language, and Meaning*, volume 2. University of Chicago Press, Chicago, Illinois, 1991.

[31] Rajeev Goré. *Tableau Methods for Modal and Temporal Logic.* Automated Reasoning Project (TR-ARP-15-95), 1995.

[32] Erich Grädel. Why are modal logics so robustly decidable?

[33] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic.* MIT Press, 2000.

[34] I. R. Horrocks and U. Sattler. A tableaux decision procedure for shoiq. In *Proceedings of Nineteenth Inter- national Joint Conference on Artificial Intelligence.* Morgan Kaufmann, January 2005.

[35] G. E. Hughes and M. J. Cresswell. *A New Introduction to Modal Logic.* Routledge, 1996.

[36] Mark Kaminski. *Studies in Higher-Order Equational Logic.* Saarland University, 2005.

[37] Saul A. Kripke. Semantical analysis of modal logic I: Normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[38] Saul A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

[39] Saul A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

[40] C. Lutz, U. Sattler, and F. Wolter. Modal logics and the two-variable fragment. In *Annual Conference of the European Association for Computer Science Logic CSL'01*, LNCS, Paris, France, 2001. Springer Verlag.

[41] Zohar Manna and Amir Pnueli. *Temporal Logic of Reactive and Concurrent Systems – Specification.* Springer, 1992.

[42] Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems – Safety.* Springer, 1995.

[43] Martin Mundhenk, Thomas Schneider, Thomas Schwentick, and Volker Weber. Complexity of hybrid logics over transitive frames. In Schlingloff [46], pages 62–78.

[44] Doron A. Peled. *Software Reliability Methods.* New York, 2001.

[45] Arthur Prior. *Past, Present and Future.* Oxford University Press, Oxford, 1967.

[46] Holger Schlingloff, editor. *4th Workshop "Methods for Modalities" (M4M), Berlin, December 1-2, 2005, Fraunhofer Institute FIRST, Proceedings*, volume 194 of *Informatik-Berichte.* Humboldt-Universität zu Berlin, 2005.

[47] Gert Smolka. *Lecture Notes: Introduction to Computational Logic.* Saarland University, http://www.ps.uni-sb.de/courses/cl-ss05/script/index.html, 2005.

[48] Balder ten Cate. *Model theory for extended modal languages.* PhD thesis, University of Amsterdam, 2005.

[49] Balder ten Cate and Massimo Franceschet. On the complexity of hybrid logics with binders, 2005.

[50] M. Tzakova. Tableau calculi for hybrid logics. In N. V. Murray, editor, *Proceedings of the International Conference TABLEAUX'99 - Analytic Tableaux and Related Methods, June 7-11, 1999, Saratoga Springs, NY USA*, volume 1617 of *LNAI.* Springer, 1999.

List of Figures

Index