Taming Frankenstein's Logic

Or

How I turned the Tables on Hybrid Logic

957

Taming Frankenstein's Logic

or

How I turned the Tables on Hybrid Logic

The Development of a Theorem Prover for Infinitary Hybrid Logic based on Semantic Tableaux, from Theory to Implementation

Harmen Wassenaar

s1058932 (towr@ai.rug.nl)

August 27, 2007

Thesis supervisor:

Rineke Verbrugge
(Artificial Intelligence, *University* of Groningen)

Referee:

Gerard Renardel de Lavalette
(Computer Science, *University* of Groningen)

Artificial Intelligence
University **of Groningen**

Abstract

In recent years the research into hybrid logic has taken flight; a number of advances have been made in axiomatizing, proof systems, completeness results and other areas. In this thesis we develop a theorem prover for infinitary hybrid logic and treat the necessary theory to achieve this.

Hybrid logic

Hybrid logic combines elements from first-order logic and modal logic. From the latter it inherits the possible world semantics, accessibility relations and the local perspective of evaluation; from the former it takes direct reference to worlds and quantification over worlds.

There are a number of possible realizations of this hybridization; in our case we consider the hybrid language formed by extending modal logic with nominals, the @-operator and the ↓-operator. Nominals are a special class of propositional variables that are true in exactly one world; in this way they name worlds and can be used to refer to them. The @-operator allows to change the place of evaluation to a specific world; e.g. $@_i\varphi$ is true if φ evaluates to true in world i. Finally, the ↓-operator can be used to make a nominal refer to the current world; in this way it acts as a quantifier, binding all future (free) occurrences of the same nominal. For example, the formula $\downarrow_i \varphi[i]$ (where $[i]$ denotes that the formula φ may contain a free i) holds if $\varphi[i]$ holds after i has been made to refer to the current world.

Using this extension of the modal language we can, among other things, characterize frame properties that were unavailable in classical modal logic; examples include irreflexivity, intransitivity and antisymmetry. If we look at irreflexivity, then the problem for modal logic becomes apparent when we consider two worlds satisfying the same atomic formulas and which link only to each other; this situation is modally indistinguishable from a single world that links to itself (because there exists a bisimulation). In hybrid logic, however, this problem is solved by our ability to distinguish distinct worlds by use of their nominals.

With the hybrid language sorted out, the next step is to consider a proof system. To attain strong completeness ($\Gamma \models \Delta \Rightarrow \Gamma \vdash \Delta$) without limiting ourselves to compact hybrid logic we need an infinitary proof system. The proof system used in this thesis is based on sequent calculus and has an infinitary derivation relation, i.e. it can require infinitely many premises. If the system is extended with only pure sequents (up to countably infinitely many) it remains strongly complete. This will, for example, also allow us to reason with frame properties such as reachability or the bounded chain condition.

Semantic tableaux

To automate theorem proving we first need a method of constructing proofs; the infinitary proof system as it is presented in [11] provides too few handholds to find a proof in it directly. Semantic tableaux provide an avenue of attack by applying a systematic search for a counter-model: if there is no possible counter-model to a potential theorem then it must be true.

We consider the negation of the candidate theorem, and try to satisfy this; if this is to be possible then the premises of the (original) sequent need to be true while the consequent is false. At each step in the tableau we can similarly consider a set of formulas that need to be true and a set of formulas that need to be false in a possible counter-model. These sets can be manipulated by tableaux rules that break down or combine the formulas, and in some cases by branching the tableau at the prospect of two possible counter-models. If at any time the positive and negative sets of formulas in a branch overlap, then a counter-model there would be inconsistent and therefore impossible. If this is the case on all branches it proves that the sequent we started with must have been a valid theorem. If no more tableau rules can be applied and there is no overlap, then there is a valid counter-model and therefore the sequent is not a theorem.

Automated theorem proving

Creating a theorem prover based on tableaux still requires some concessions to practicality. The largest obstacle is time: hybrid logic is undecidable and so a theorem prover might be unable to find either proof or disproof, and even when there is a proof it may take arbitrarily long. An additional problem with tableaux is that the criterion for identifying a counter-model, that no more rules apply, may sometimes never be reached even when there is a counter-model. By combining the prover with a small model checker this latter problem can be solved in a number of common cases. The general problem of running-time can be kept under control by limiting the search depth.

Another obstacle on our path is the sheer breadth of the logic we deal with; infinitary terms allow a wide range of induction structures to be used. For our prover we have limited ourselves to the simplest and most common: weak induction with a single base case.

The theorem prover has been implemented in SWI prolog. The most important parts are the rulebase and the proving 'engine' which uses the rules from the rulebase to search for a proof. Keeping this rulebase separate from the rest of the prover facilitates making changes to the behaviour of the prover, by allowing easy modification of the tableau rules.

The prover also has a small integrated model checker, but it is very limited in scope. It only works on pure formulas (i.e. formulas that do not contain propositional variables); and any infinitary terms will have to be very basic. Consequently, it may fail to recognize when a counter-model can be formed from the formulas in a tableau node; but when it does detect one, it should be correct.

The output of the prover can be one of three things: it may fail to reach a conclusion, it may find a counter-model, or it may find a proof. When it finds a proof it will interact with pdflatex to construct a pdf file showing a deductive proof of the theorem.

Contents

Introduction

> *It's a dark and stormy summer night. In his lab we find Victor Frankenstein hard at work, hacking, cutting and dissecting, sewing the parts together in novel arrangements. It is the moment of his greatest creation. But there is no lightning involved here; somewhere events had taken an awfully wrong turn in this alternate universe. This Frankenstein is no chemist obsessed with the creation of life, but a logician. His creation: a logic composed of borrowed parts – Hybrid Logic.*
>
> *And thus begins our harrowing tale.*

In reality the origin of hybrid logic is much more mundane. Its history starts in the 1960s with Arthur Prior's hybrid tense logic [15]. But only as recently as the mid 1990s has hybrid logic really taken flight. One of the developments in recent years is a sound and strongly complete infinitary proof system for hybrid logic [11], and it is in that vein that this thesis continues.

Goal of the thesis

The goal of this thesis is to develop an automated theorem prover for infinitary logic. But what is infinitary hybrid logic, and why do we want it? These are questions that will need to be answered first. And once we have established the theory, how then do we prove theorems and how do we automate this process?

In this thesis we treat the development of an infinitary hybrid logic prover from theory to implementation. Along the way we also take a small detour to have a look at Sahlqvist formulas and how they interact with our hybrid logic.

Overview of the thesis

The thesis is divided into five parts and an appendix, each consisting of two or three chapters. Every part is preceded by a small overview of the contents, and followed by the list of references particular to that part.

The first part consists of three chapters which treat the development of hybrid logic. Chapter 1 provides an introduction to modal logic, and gives a motivation for extending it to hybrid logic. The next chapter discusses the language of hybrid logic. Finally, in the third chapter, we introduce a proof system for infinitary hybrid logic based on [11].

The second part of the thesis is made up of chapters 4 and 5. In chapter 4 we apply the proof system to a number of examples: proving some useful derivative rules and theorems, demonstrating equivalence between pure and modal axioms which define the same frame properties, and finally proofs of a few infinitary theorems. Chapter 5 digresses slightly from the main line of the thesis to extend the class of frames for which we know equivalent modal and hybrid axioms. By taking a detour through first-order and second-order logic we find that the entire class of so-called Sahlqvist axioms can be rewritten in hybrid form.

Part III introduces the main method applied by our theorem prover. First, a basic introduction into semantic tableaux is provided in chapter 6, and the scope is subsequently broadened in chapter 7 to accommodate infinitary hybrid logic.

At long last, part IV brings us to the actual theorem prover. Chapter 8 deals with the overall design of the prover, followed by a discussion of the implementation in chapter 9 and an evaluation of the prover in chapter 10.

The last part of the thesis discusses what we have accomplished so far and what work may lie ahead for those that venture onwards.

Finally there are three appendices. Appendix A provides additional hand proofs, like the ones in chapter 4. Appendix B consists of additional computer proofs which did not make it into chapter 10. The last appendix, C, gives the code for the theorem prover.

Part I
Tractatus Logico-Hybridicus

Contents

1 Modal logic

"The rain in Spain falls possibly on the plain."

1.1 Introduction

Modal logic as we know it today is best seen as an extension of propositional logic. Aside from the usual propositional operators we also have the modal operators $\Box$ and $\Diamond$, which may be read in several ways depending on what sort of modal logic you have.

The standard interpretations are "necessarily" and "possibly", respectively. So this allows you to say "possibly it rains" ($\Diamond r$), and "necessarily if it rains I take my umbrella" ($\Box[r \rightarrow u]$). Intuitively, it follows from these two statements that "possibly I take my umbrella" ($\Diamond u$). Modal logic can help to formalize this sort of reasoning.

But modal logic is much broader than just this. It does not just apply to various concepts of necessity (alethic modal logic), but can also be used to deal with knowledge (epistemic logic), beliefs, what ought to be (deontic logic), what is provable (provability logic), computer programs (dynamic logic) and more.

The common characteristic binding these diverse topics is that you can make use of models composed of a collection of possible worlds (or states) that are connected in some (specific) way. This is quite obvious for a topic such as computer science, because there we are familiar with finite state machines: each statement in a program brings you from one state to the next.

For knowledge the leap away from intuition is slightly larger; you know something when it is true in all possible worlds you might conceivably be in. You might well imagine a world where the rain in Spain falls mainly outside the plains, but it cannot be the world you occupy. It is important in this respect to distinguish what is logically possible from what is epistemically possible. There is no logical reason why the rain should not fall elsewhere, but to believe otherwise would simply be contrary to evidence (assuming the old adage is true).

1.2 Semantics

The versatility of modal logic lies in the underlying semantic framework. Much of the credit for its development goes to Saul Kripke, and for this reason it is known as Kripke semantics. In this framework modal statements are interpreted in terms of (Kripke) models $M = (W, R, V)$.

The first part of the model, W, gives the set of possible worlds (or states or nodes). The precise nature of this set determines part of the interpretation; e.g. the set might consist of program states, or ways the world might be.

The second part, R ($\subseteq W \times W$), is an accessibility relation and lets you ‘move’ from one world to the next; e.g. in terms of computer science it may represent a state transition, or when dealing with knowledge it can say that a world is consistent with what you know (given the actual world).

The final part of the model, V ($P \rightarrow 2^W$), completes the model by determining in which subset of worlds a given atomic formula is true; e.g. when your set of worlds W is $\{a, b, c, d, e, f\}$, it might say that if you are wondering whether *“it rains in Spain”*, this is true in worlds a, b and c, and consequently the opposite (*“it's not raining in Spain”*) is true in the remaining worlds, d, e and f.

Using this conception of models, we can now define when a formula is true. Reading $M, w \models \varphi$ as ‘world w from model M satisfies the formula φ’ we have the following:

$M, w \models p$ iff $w \in V(p)$
$M, w \models \neg\varphi$ iff $M, w \not\models \varphi$
$M, w \models \varphi \wedge \psi$ iff $M, w \models \varphi$ and $M, w \models \varphi$ (similar for $\vee$, $\rightarrow$, $\leftrightarrow$)
$M, w \models \Diamond\varphi$ iff $\exists_{v \in W}(wRv$ and $M, v \models \varphi)$
$M, w \models \Box\varphi$ iff $M, w \models \neg\Diamond\neg\varphi$ (or $\forall_{v \in W}(wRv \Rightarrow M, v \models \varphi)$)

Furthermore, a formula is considered *true in a model*, if it is satisfied by all worlds of the model, i.e. we have $M \models \varphi \overset{def}{\equiv} \forall_{w \in W}(M, w \models \varphi)$. And something is *logically true* (valid) if it is true in all models; $\models \varphi \overset{def}{\equiv} \forall_M(M \models \varphi)$. In other words, a formula is valid when there is no counter-model (i.e. a model with a world where the opposite is true). Finally, for a set Γ we define $M, w \models \Gamma \overset{def}{\equiv} \forall_{\varphi \in \Gamma}(M, w \models \varphi)$ and $\Gamma \models \varphi \overset{def}{\equiv} \forall_{(M,w)}(M, w \models \Gamma \Rightarrow M, w \models \varphi)$

The basic modal logic corresponding to the semantics just laid out is known as the K system. It is formed by extending propositional logic with the following:

- The necessitation rule: if $\vdash \varphi$ then $\vdash \Box\varphi$
- The distribution (or K) axiom: $\vdash \Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$

This general system forms the basis for more specific modal logics by adding extra axioms to it. However, this also puts extra demands on the underlying semantics. Instead of looking at all models when determining the validity of a statement, you need to look at a subset of models.

1.3 Frame properties

At first it might seem we would need the subset consisting of all models in which the extra axioms are true. Luckily this turns out not to be the case. Instead of models we can make use of frames [1].

[1] Any model outside the frame class that makes the axioms true turns out to be bisimilar to a model inside the frame class, meaning they satisfy exactly the same

A frame $F = \langle W, R\rangle$ is like a model without the valuation function. Effectively it represents the underlying structure of a model. A formula φ is true in a frame $\langle W, R\rangle$ if it is true in all models on the frame, i.e. $\forall_V(\langle W, R, V\rangle \models \varphi)$.

When considering modal axioms in light of these frames, axioms turn out to correspond precisely with properties of the accessibility relation. So, for example, when considering epistemic logic, we have amongst others the axiom $\Box A \rightarrow A$ (or its dual $A \rightarrow \Diamond A$). The accessibility relation of the frame will be reflexive (as we would expect, in each world we must consider that world itself possible).

Together, all the frames that have the same property form a frame class, on which we can define the semantics of our modal logic. To be precise our frame class is characterized by an axiom from the modal logic in the following way

i) the modal axiom is true in every frame of the class, and

ii) every frame in which the axiom is true belongs to the class,

When multiple axioms are involved the frame class consists of the intersection of the frame classes characterized by the axioms separately.

A few examples of frame properties are shown in table 1.1. Each of these properties is a first order definable property. Not all modal axioms characterize first order properties, however; e.g. $\Box(\Box A \rightarrow A) \rightarrow \Box A$ (the Löb formula) is inherently second order in nature.

property	axiom
reflexivity	$\Box A \rightarrow A$
symmetry	$A \rightarrow \Box\Diamond A$
transitivity	$\Box A \rightarrow \Box\Box A$
density	$\Box\Box A \rightarrow \Box A$
determinism	$\Diamond A \rightarrow \Box A$

Table 1.1, examples of frame properties that can be characterized with modal axioms

1.4 Frame properties which cannot be characterized by modal axioms

As it turns out a lot of frame classes cannot be characterized by modal axioms. Part of the problem is that modal logic follows the duck hypothesis; if it walks like a duck and talks like a duck, then it is a duck. Moreover it is the same duck each and every time.

That is to say, distinct worlds may be indistinguishable to modal logic. And this poses a problem for those properties where distinguishing distinct worlds is exactly what you need to do, e.g. in irreflexive frames where a world may only be connected to a world other than itself. This also means

formulae

entire models may be indistinguishable, as you can e.g. replace two worlds by one, yielding a new model that is modally equivalent (in the sense you cannot distinguish the two models by modal formulas). To demonstrate this the concept of bisimulation [7] will be instrumental.

Definition (Bisimulation). A bisimulation between models $M = (W, R, V)$ and $M' = (W', R', V')$ is a non-empty relation E between worlds of M and M' (i.e. $E \subseteq W \times W'$), such that whenever wEw' ($w \in W$ and $w' \in W'$) we have the following:

atomic harmony: the same atomic formulas are true in w and w'
zig: if Rwv, then a world exists $v' \in W'$ such that vEv' and $R'w'v'$
zag: if $R'w'v'$, then a world exists $v \in W$ such that vEv' and Rwv

If two models are bisimilar this implies that they cannot be distinguished by a modal formula; for every world in one model there is a world in the other model that makes the same modal formulas true and vice versa.

Using this we can show why certain frame classes cannot be characterized by a modal axiom. Let's first recall what characterization by an axiom means:

i) the modal axiom is true in every frame (and thus model) of the class and

ii) every frame in which the axiom is true belongs to the class.

This provides us with a straightforward approach to show certain properties cannot be characterized by a modal axiom: Find a frame that does not have the property for which each model X has a bisimulation to a model Y on a frame that does have the property.

Since every Y is a model with the frame property we are interested in, the axiom that would characterize it must be true there. And because of the bisimulation this axiom also has to be true in every X. But that means the axiom would be true in the frame we chose particularly for not having this property, which gives us a contradiction.

Now let us apply this to a few examples. The first case is irreflexivity, illustrated by figure 1.1.

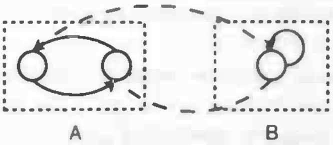


Figure 1.1, A bisimulation between an irreflexive model A and a reflexive model B. (All worlds make the same atomic formulae true)

Given any particular valuation B is a reflexive model. Through the bisimulation, shown with dashed lines, it is equivalent with a model A (with both worlds satisfying the same formulas). However, as model A is clearly from an irreflexive frame, neither of the worlds is accessible to itself. This must mean that if there were a modal axiom characterizing irreflexivity, it is true in A. And if it is true in A, then it is also true in B, because of the bisimulation. And since this holds for every valuation of B it holds in the frame underlying B, which in turn must be irreflexive if a modal axiom characterizing it were true there. However, a frame cannot be reflexive and irreflexive at the same time. So the only conclusion open to us is that there can be no such modal axiom.

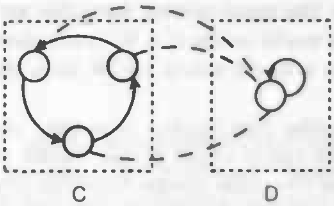


Figure 1.2, A bisimulation between an asymmetrical model C and a symmetrical model D.

A similar story holds for asymmetry, as illustrated in figure 1.2. When none of the worlds in model C are distinguishable, then the model as a whole is indistinguishable from one where there is just a single reflexive world. But such a model is symmetrical, so no modal axiom characterizing asymmetry may hold there for all valuations of the model. And since any valuation for D leads to one for C with all worlds equally valued, it cannot exist.

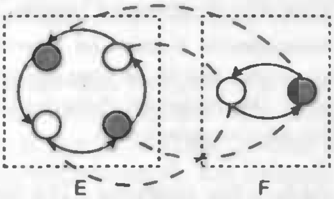


Figure 1.3, A bisimulation between an antisymmetrical model E, and a symmetrical, irreflexive model F (Worlds with the same color make the same atomic formulae true)

There are always many possible counterexamples, but it usually pays to keep it simple. A good heuristic is to start with the smallest model that does not have the property you are examining. For example, a single reflexive world in the case of irreflexivity and asymmetry. And then proceed to find a model it can map to which does have this property.

To disprove the existence of an antisymmetry axiom this means we start with two worlds, which are connected to each other (F in figure 1.3). The obvious choice for an antisymmetric model would be a loop with 3 or more worlds; however, it only works out for an even number of worlds, because otherwise we cannot have a bisimulation. The two worlds in our symmetric, irreflexive model may be different, illustrated by the two different colors in figure 1.3. So with every step along the accessibility relation we have to alternate between the two kinds of worlds, where one kind is bisimilar to the first world (white), and the other kind to the second (gray). This means 4 worlds is the least we can do (E from figure 1.3).

It turns out this counterexample works for the previous frame property as well. So we might have shown there cannot be axioms for asymmetry and antisymmetry at the same time. Likewise, it turns out intransitivity shares the counterexample for irreflexivity (figure 1.4).

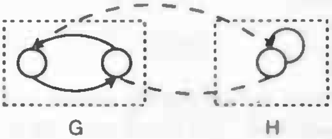


Figure 1.4, A bisimulation between an intransitive model G and transitive model H

The next frame property is a bit of an oddball. So far we have used bisimulation between models in a sense such that all worlds in one model are bisimilar to a world in the other. For trichotomy this will not work, because it says every two distinct worlds are connected in one direction or the other. So if we try to reduce a trichotomic model by identifying worlds with each other it will always keep this property, or conversely building up a model by splitting/doubling worlds only stands to lose it.

Fortunately bisimulation does not require such a strong correspondence, something weaker will also do. For this we need to define the concept of a *generated submodel* [7]. Given a set of worlds, every world reachable in zero or more steps from those worlds belongs to the generated submodel (generated by the given set of starting worlds). All that a bisimulation requires is that both models have a certain generated submodel, such that each world from one generated submodel is bisimilar to a world from the other and vice versa. This is exactly what we need for trichotomy. If we have a model with two disconnected worlds, then the model is not trichotomic (K in figure 1.5). But since both worlds are disconnected each on its own is a generated submodel, and bisimilar to the world of a single world model, which is trichotomic (I and J in figure 1.5).

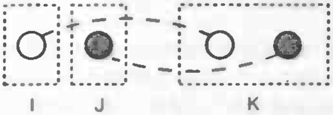


Figure 1.5, A bisimulation between trichotomic models I and J and their non-trichotomic union K

2 Hybrid logic

There is an oddity at the heart of modal logic: although possible worlds take the center stage in the semantics (as states in the model) they are left out of the picture in the syntax. Now, there is not necessarily anything wrong with such an oddity, except that in this case it also poses certain problems. Properties that are trivial to define on the semantic level become impossible to characterize in the modal language. Take, for example, irreflexivity, $\forall_s \neg sRs$, which is very easy to state in terms of first order logic, but cannot be expressed modally.

2.1 Nominals

Hybrid logic remedies this by extending modal logic with a new class of special propositional variables, the nominals. These nominals are true in exactly one world, and thus can be used to uniquely name that world. This makes it possible to say e.g. $i \rightarrow \neg\Diamond i$ (where i is a nominal), 'if we are in world i then this world (i) is not accessible'. When this is true for each nominal, i.e. one reads $i \rightarrow \neg\Diamond i$ as a schema in which i may be instantiated by any other nominal, then it characterizes irreflexivity. In this way a host of new frame properties becomes available which our logic can characterize and reason about.

To accommodate nominals the semantics has to be slightly adjusted. The model $M = (W, R, V)$ is extended to $M = (W, R, V, A)$ with an assignment function $A : I \rightarrow W$ (where I is a countable set of nominals). $A(i)$ then gives the unique world named by nominal i, therefore:

$M, w \models i$ iff $A(i) = w$

Instead of adding an assignment function, it is also possible to extend the valuation function V, and treat nominals more like propositional variables. However, conceptually it has advantages to keep nominals and (regular) propositional variables well separated. For example, if we were to extend the valuation function, we would have to constrain it such that it can only assign singleton sets to nominals, using the assignment function this is a given. Nevertheless, the equivalence is important, as it means we are still, in essence, dealing with regular Kripke models.

2.2 The satisfaction operator @

To make reasoning with nominals more comfortable the satisfaction operator $@_i$ is added for each nominal i. The formula $@_i\varphi$ should be read as 'at the world named i formula φ holds'. The truth definition for $@_i$ follows this reading precisely:

$M, w \models @_i\varphi$ iff $M, A(i) \models \varphi$

In essence, the operator $@_i$ can be seen as the modal counterpart to an accessibility relation R_i which connects every world to the world named i. This also means $@_i$ is a normal modal operator, and consequently the K axiom and (Strong[2]) necessitation hold for it.

The hybrid logic developed so far is known as $\mathcal{H}(@)$. In it we can, as a start, characterize all the frame properties listed in table 2.1. As shown, there are also often alternative hybrid axioms that correspond more closely to the characterization on the semantic level (specifically in the cases where there are only universal quantifiers). This is a result of the correspondence between nominals in the syntax and worlds in the model. The commonly used axioms are often further removed from the semantics, but more concise.

property	pure axiom	alternative	model
reflexivity	$@_i\Diamond i$		$\forall_s(sRs)$
symmetry	$@_i\Box\Diamond i$	$@_i\Diamond j \rightarrow @_j\Diamond i$	$\forall_{s,t}(sRt \rightarrow tRs)$
transitivity	$\Diamond\Diamond i \rightarrow \Diamond i$	$\Diamond i \wedge @_i\Diamond j \rightarrow \Diamond j$	$\forall_{s,t,u}(sRt \wedge tRu \rightarrow sRu)$
density	$\Diamond i \rightarrow \Diamond\Diamond i$		$\forall_{s,u}(sRu \rightarrow \exists_t sRt \wedge tRu)$
determinism	$\Diamond i \rightarrow \Box i$	$\Diamond i \wedge \Diamond j \rightarrow @_i j$	$\forall_{s,t,u}(sRt \wedge sRu \rightarrow t = u)$
irreflexivity	$@_i\neg\Diamond i$		$\forall_s(\neg sRs)$
asymmetry	$@_i\neg\Diamond\Diamond i$	$@_i\Diamond j \rightarrow @_j\neg\Diamond i$	$\forall_{s,t}(sRt \rightarrow \neg tRs)$
antisymmetry	$@_i\Box(\Diamond i \rightarrow i)$	$@_i\Diamond j \wedge @_j\Diamond i \rightarrow @_i j$	$\forall_{s,t}(sRt \wedge tRs \rightarrow s = t)$
intransitivity	$\Diamond\Diamond i \rightarrow \neg\Diamond i$	$\Diamond i \wedge @_i\Diamond j \rightarrow \neg\Diamond j$	$\forall_{s,t,u}(sRt \wedge tRu \rightarrow \neg sRu)$
trichotomy	$@_j\Diamond i \vee @_j i \vee @i\Diamond j$		$\forall_{s,t}(sRt \vee s = t \vee tRs)$

Table 2.1, examples of frame properties characterized by pure (hybrid) axioms (axioms containing no propositional variables). The top half can also be characterized by modal axioms, but the bottom half cannot.

The axioms which are used in hybrid logic are preferably pure; this means that no propositional variables (other than nominals) occur in them. When the base hybrid system is extended with only pure axioms, it is automatically complete for the class of corresponding frames [11]. But the regular modal axioms, if they exist, are of course also still valid.

The logic $\mathcal{H}(@)$ also has another attractive property: it is decidable. This means that for any well-formed formula it is possible to determine whether it is valid in a finite number of steps.

2.3 The nominal binder ↓

Hybrid logic may be further extended to add even more expressivity. The obvious choice is a binder, like ∀ or ∃. In fact, with these choices the resulting logic would be as expressive as first order logic. However, this does not do justice to the modal character of our logic, because if variables could be bound to arbitrary points we would lose the intuitive locality present in Kripke semantics. So for this reason often the ↓ operator is chosen, which

[2] Strong necessitation is embodied by the rule $\frac{\Gamma\vdash\varphi}{\Box\Gamma\vdash\Box\varphi}$

binds a nominal to the current world. (The combination of both @ and $\downarrow$ will be discussed in the next subsection; first we will examine the $\downarrow$ itself.)

One way to look at it is to say that $\downarrow_i$ gives the name i to the current world. And so you could say e.g. $\downarrow_i \Box\Diamond i$, 'If I name this world i, then in every world accessible from here, I can get back to i'. The observant reader will notice $\downarrow_i \Box\Diamond i$ characterizes symmetry and seems to follow the same structure as the corresponding pure axiom $@_i\Box\Diamond i$. It is important to note however that $\downarrow_i \varphi[i]$ binds the free occurrences of the nominal i in $\varphi[i]$, whereas $@_i\varphi[i]$ does not. The formula $@_i\varphi[i]$ merely forces the evaluation of $\varphi[i]$ to be done in the world named by i, whereas $\downarrow_i \varphi[i]$ changes what i means for future reference. This is reflected in the semantics by a local change in the model whenever $\downarrow_i$ is encountered:

$M, w \models \downarrow_i \varphi$ iff $M[i := w], w \models \varphi$

The model $M = (W, R, V, A)$ is modified to $M[i := w] = (W, R, V, A')$, where $A' = A[i := w]$ behaves like A on $I - \{i\}$ and $A'(i) = w$.

There are many other pure axioms like $@_i\Box\Diamond i$, which can be similarly rewritten using $\downarrow$. In some cases free nominals can be eliminated altogether (e.g. see table 2.2). Using such bound axioms means that you no longer need all instances of a parametrized pure axiom. This makes things at least conceptually simpler.

frame property	bound pure axiom
reflexivity	$\downarrow_i \Diamond i$
symmetry	$\downarrow_i \Box\Diamond i$
irreflexivity	$\downarrow_i \neg\Diamond i$
asymmetry	$\downarrow_i \neg\Diamond\Diamond i$
antisymmetry	$\downarrow_i \Box(\Diamond i \rightarrow i)$
transitivity	$\downarrow_i \Box\Box \downarrow_j @_i\Diamond j$
density	$\downarrow_i \Box \downarrow_j @_i\Diamond\Diamond j$

Table 2.2, several examples of axioms where using $\downarrow$ results in pure axioms without free nominals.

2.4 The hybrid language $\mathcal{H}(@, \downarrow)$

When both operators @ and $\downarrow$ are added to the hybrid language together, it yields a logic that characterizes locality [1]. In other words it characterizes the fraction of first order logic that is invariant for generated sub-models. The validity of a formula depends solely on the current world, worlds named by nominals, and successors of these worlds.

Using $\mathcal{H}(@, \downarrow)$ we can precisely explore and describe any part of a model or frame accessible via the successors of a (set of) starting point(s) (see figure 2.1 for a small example).

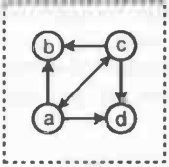


Figure 2.1, The minimal frame described by the formula

$$\begin{aligned}&{\downarrow_a} \Diamond {\downarrow_b} @_a \Diamond {\downarrow_c} @_a \Diamond {\downarrow_d} (\\ &\qquad \neg @_b a \wedge \neg @_c (a \vee b) \wedge \neg @_d (a \vee b \vee c)\\ &\qquad \wedge @_a \Box [b \vee c \vee d]\\ &\qquad \wedge @_b \Box \bot \wedge @_c [\Diamond a \wedge \Diamond b \wedge \Diamond d \wedge \Box (a \vee b \vee d)] \wedge @_d \Box \bot)\\ &\quad \vee \Box \bot.\end{aligned}$$

The union of any number of generated sub-frames also satisfies the formula.
In a similar way any generated sub-frame of any given frame can be systematically described using $\mathcal{H}(@, \downarrow)$.

Because of the way @ and ↓ operator complement each other we can always find our way back when probing our way through a model. At any time we may choose to store the world we are in by naming it with ↓ and later on we can return to it using @, and, for example, explore a different path.

Unlike $\mathcal{H}(@)$ the logic $\mathcal{H}(@, \downarrow)$ does not have a decidable satisfiability problem. In fact even if we leave out the @ operator and free nominals, the remaining fraction of the logic is still undecidable [1]. A decision algorithm may therefore never terminate, and this is a problem a practical theorem prover has to deal with.

3 Hybrid logic with an infinitary proof system

In the previous chapter we discussed the semantic underpinnings of hybrid logic; in this chapter we will provide an accompanying proof system, based on [11].

A proof system allows us to deduce theorems by purely syntactic means. New theorems are formed from old theorems and axioms by applying the rules of the system. This removes the need to translate everything to the underlying semantics and determine the truth on the underlying level.

There are two important factors to consider. First we do not want to be able to deduce 'theorems' in the system that are not valid with respect to the underlying semantics; and secondly, we want to be able to deduce every theorem that *is* valid. These problems are known as soundness and completeness, respectively. For the system laid out in this chapter proofs of soundness and completeness are discussed in [11].

3.1 The proof system

In the first two chapters we have only implicitly given the language of our logic; it follows from the definition of the semantics as only formulas with a meaning are part of the language. Explicitly the formulas of our logic are defined, in BNF notation, by:

$\varphi ::= \bot \mid \top \mid p \mid i \mid \neg\varphi \mid \varphi_1 \oplus \varphi_2 \mid \circledast\varphi$

Where $\oplus \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ and $\circledast \in \{\Box, \Diamond, @_i, \downarrow_i\}$.

To form statements about these formulas we will use sequent notation. Sequents have the form $\Gamma \vdash \varphi$ where Γ is a countable (possibly infinite) set of formulas and φ is a formula. Allowing Γ to be countably infinite has the consequence that our proof system will be infinitary as well; proof sequents may require infinitely many premises.

Given that the system has the properties of soundness and completeness, we have that $\Gamma \vdash \varphi$ iff $\Gamma \models \varphi$. Further we define $\Gamma \vdash \Delta$ iff [3] $\Gamma \vdash \varphi$ for all $\varphi \in \Delta$.

We can now look at the actual proof system. The axiom sequents and sequent rules for the system are given in table 3.1.

One of the first things one might notice is the conspicuous absence of the K axioms, which one would expect in any proof system for a normal modal logic such as this one. The reason they are not explicitly included is because they follow SNec by applying it to MP and then applying Ded[4].

[3] In the second part of the paper, within the context of tableaux, $\Gamma \vdash \Delta$ has a different interpretation.

[4] Their omittance is a slight deviation from [11], where they were still unnecessarily included.

Taut	$\vdash \varphi$ if φ is an instance of a tautology	
MP	$\varphi, \varphi \to \psi \vdash \psi$	(modus ponens)
SD$_@$	$\vdash @_i\varphi \to \neg @_i \neg \varphi$	(self-dual)
Intr	$\vdash i \wedge \varphi \to @_i\varphi$	(introduction)
T$_@$	$\vdash @_i i$	(reflexivity)
Agree	$\vdash @_i @_j \varphi \leftrightarrow @_j \varphi$	(agree)
Back	$\vdash \Diamond @_i \varphi \to @_i \varphi$	(back)
DA	$i \vdash \downarrow_j \varphi \leftrightarrow \varphi[j := i]$	(downarrow)
Name	$\vdash \downarrow_i @_i \varphi \to \varphi$ provided $i \notin \mathtt{fnom}(\varphi)$	(name)
BG	$i \vdash \Box \downarrow_j @_i \Diamond j$ provided $i \neq j$	(bounded generalization)
SNec$_\Box$	if $\Gamma \vdash \varphi$ then $\Box\Gamma \vdash \Box\varphi$	(strong necessitation)
SNec$_@$	if $\Gamma \vdash \varphi$ then $@_i\Gamma \vdash @_i\varphi$	(strong necessitation)
SNec$_\downarrow$	if $\Gamma \vdash \varphi$ then $\downarrow_i \Gamma \vdash \downarrow_i \varphi$	(strong necessitation)
InfCut	if $\Gamma \vdash \Delta$ and $\Gamma', \Delta \vdash \varphi$ then $\Gamma, \Gamma' \vdash \varphi$	(infinitary cut)
Ded	if $\Gamma, \varphi \vdash \psi$ then $\Gamma \vdash \varphi \to \psi$	(deduction)

Table 3.1, The axiom sequents and sequent rules for $\mathcal{H}_\omega(@, \downarrow)$

The axiom **Name** relies on the mapping $\mathtt{fnom}(\varphi)$, which gives the set of free nominals that occur in a formula φ (or set of formulas). The inductive definition of fnom is fairly straightforward, so we will only comment on the two most important clauses:

$\mathtt{fnom}(@_i\varphi) = \mathtt{fnom}(\varphi) \cup \{i\}$ The @-operator does *not* bind nominals
$\mathtt{fnom}(\downarrow_i \varphi) = \mathtt{fnom}(\varphi) - \{i\}$ The $\downarrow$-operator *does* bind nominals

For practical purposes the proof system may be a bit too minimalistic. Undoubtedly some effort went into reducing redundancy and choosing the most basic axioms that came to mind, but in practice it pays to use extended axioms or derivative theorems. Two axioms that can be easily and usefully extended are **SD$_@$** and **Name**: by combining them with their contraposition you can get equalities instead of implications, which then allows their use for substitution. A few simple derivable rules are T$_\downarrow$, SD$_\downarrow$ and DedRev; possibly the two most important ones are **bridge** and **Paste**. Derivations for numerous theorems are given in the next chapter and the appendix.

The proof system can be extended with further axioms (or rather, axiom sequents) to form proof systems for logics on specific frames. As long as it is extended with only pure axioms (or axioms provably equivalent to a pure axiom) we will retain soundness and completeness between the proof system and the semantics [11]. So for example, we can take $\mathcal{H}_\omega(@, \downarrow)$ together with $\vdash @_i \neg \Diamond i$ to form a proof system for irreflexive hybrid logic.

3.2 Logics with infinitary axioms

The reason we need an infinitary proof system is because hybrid logic is not compact. Compactness means that given $\Gamma \models \varphi$ we can always find a finite $\Gamma' \subseteq \Gamma$ such that $\Gamma' \models \varphi$. Basic modal logic is an example of a logic that is compact, and because of that it can be finitely axiomized.

A finite axiomatization for hybrid logic, however, would be incomplete because we would not be able to construct proofs of non-compact theorems. At most it might be complete with respect to the compact fragment of hybrid logic. But then we would miss out on the range infinitary logics; for example, the ones shown in table 3.2.

ancestral logic	$\{@_i\Box^n\neg j \mid n \in \mathbb{N}\} \vdash @_i\Box^*\neg j$
reachability logic	$\{\neg @_i\Diamond^n j \mid n \in \mathbb{N}\} \vdash \bot$
cycle logic	$\{\neg @_i\Diamond^n i \mid n \in \mathbb{N} \wedge n \geq 1\} \vdash \bot$
BCC logic	$\{\Diamond^n \top \mid n \in \mathbb{N}\} \vdash \bot$

Table 3.2, several hybrid logics characterized by infinitary axiom sequents.

In hybrid ancestral logic we have an extra modal operator $\Box^*$, besides the usual $\Box$. $\Box^*$ corresponds with the reflexive transitive closure of the accessibility relation of $\Box$, and so addresses all descendants of a given state at once.

Hybrid reachability logic operates on the class of frames where each world is reachable (through a finite number of steps) from any other world.

As the name suggests, hybrid cycle logic is a logic where all worlds are connected to themselves (in one or more steps).

The axiom for hybrid BCC literally states that 'falsum' follows from the assumption that there are paths of every length, which means that there must be some maximum path length; unsurprisingly BCC stands for bounded chain condition. One might imagine BCC is useful for guaranteeing termination when combined with dynamic logic.

A number of example proofs involving infinitary axioms will be handled in chapter 4.

References

[1] Carlos Areces, Patrick Blackburn, and Maarten Marx. Hybrid logic: Characterization, interpolation and complexity. *Journal of Symbolic Logic*, 66(3):977–1010, 2001.
website: `http://www.loria.fr/~blackbur/pub.html`
doc: `http://www.loria.fr/~blackbur/papers/charac.pdf`

[3] Patrick Blackburn. Modal logic as dialogical logic. *Synthese*, 127:57–93, 2001.
website: `http://www.loria.fr/~blackbur/pub.html`
doc: `http://www.loria.fr/~blackbur/papers/synthese.pdf`

[4] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*, chapter Hybrid logic, pages 434–445. Cambridge University Press, 2002.
website: `http://www.mlbook.org/`

[7] Patrick Blackburn and Johan van Benthem. Modal logic: A semantic perspective. In *Handbook of Modal Logic*. Elsevier Science, 2006.
website: `http://staff.science.uva.nl/~johan/seminar.html`
doc: `http://staff.science.uva.nl/~johan/hml-blackburnvanbenthem.pdf`

[11] Barteld Kooi, Gerard Renardel de Lavalette, and Rineke Verbrugge. Hybrid logics with infinitary proof systems. *Journal of Logic Computation*, 16(2):161–175, 2006.
website: `http://logcom.oxfordjournals.org/cgi/content/abstract/16/2/161-a`
doc: `http://logcom.oxfordjournals.org/cgi/reprint/16/2/161-a.pdf`

[13] Edward John Lemmon and Dana S. Scott. *The 'Lemmon notes' : An Introduction to Modal Logic*. Oxford : Blackwell, 1977.

[17] Balder ten Cate. *Model theory for extended modal languages*, chapter Introduction to hybrid languages, pages 37–44. ILLC Dissertation Series. Institute for Logic, Language and Computation, 2005.
website: `http://www.illc.uva.nl/Publications/reportlist.php?Series=DS`
doc: `http://www.illc.uva.nl/Publications/Dissertations/DS-2005-01.text.pdf`

[19] Edward N. Zalta. *Basic Concepts in Modal Logic*. web publication, 1995.
website: `http://mally.stanford.edu/publications.html`
doc: `http://mally.stanford.edu/notes.pdf`

Part II

Show and Tell

Contents

4 Proof examples

"mmm... pudding" [5]

In this chapter a number of examples will be given of various sorts of proofs using the hybrid logic laid out in chapter 3. First, a number of simple derivable rules and sequents will be treated, followed by two generalized sequents and a generalized rule. Then we shall take a look at modal axioms defining frame conditions and their hybrid equivalents. And finally three infinitary hybrid theorems will be discussed.

It should be noted that various of the following proofs may contain sequents and rules that have not been discussed before they are used. These will be dealt with in the appendix, along with other proofs for which there is no space here.

4.1 Derivable rules and sequents

4.1.1 W (Weakening): $\Gamma \vdash \varphi \;\Rightarrow\; \Gamma, \Delta \vdash \varphi$

Possibly the most trivial, but still useful, rule that can be deduced in our logic system is the weakening rule. It follows directly from the infinitary cut rule and allows the introduction of arbitrary formulas to the antecedent of a sequent.

$$\frac{\Gamma \vdash \varphi \qquad \Delta \vdash \emptyset}{\Gamma, \Delta \vdash \varphi}\;\text{InfCut}$$

Note that $\Delta \vdash \emptyset$ must be the case, because we have $\Delta \vdash \varphi$ for all $\varphi \in \emptyset$.

4.1.2 Namerev: $\vdash \varphi \rightarrow \downarrow_i @_i\varphi$ provided $i \notin \mathtt{fnom}(\varphi)$

Namerev, as the name suggests, is a sequent which serves the opposite purpose of Name. Using the self-duality of @[6] and ↓[7] we can derive it from the contraposition of Name.

In future use, Name will be used to refer to its combination with **Namerev**, i.e. $\vdash \varphi \leftrightarrow \downarrow_i @_i\varphi$.

$$\begin{array}{cl}
\text{Name } \{i \notin \mathtt{fnom}(\varphi)\} & \\ \hline
\vdash \downarrow_i @_i\neg\varphi \rightarrow \neg\varphi & \\ \hline
\vdash \varphi \rightarrow \neg \downarrow_i @_i\neg\varphi & \text{cp} \\ \hline
\vdash \varphi \rightarrow \downarrow_i \neg @_i\neg\varphi & \text{SD}_{\downarrow} \\ \hline
\vdash \varphi \rightarrow \downarrow_i @_i\varphi & \text{SD}_{@}
\end{array}$$

[5] In all fairness, the proof is not actually in the pudding, but in the eating thereof. But a google count shows over 50% of people get this wrong.

[6] We use $\mathbf{SD_@}$: $\vdash @_i\varphi \leftrightarrow \neg @_i\neg\varphi$, which can be derived from combining the original, $\mathbf{SD_@}$: $\vdash @_i\varphi \rightarrow \neg @_i\neg\varphi$, and its contraposition.

[7] $\mathbf{SD}_{\downarrow}$: $\vdash \neg \downarrow_i \varphi \leftrightarrow \downarrow_i \neg\varphi$

4.1.3 distr$^{@}_{\vee}$ (distribution of @ over $\vee$): $\vdash @_i(\varphi \vee \psi) \leftrightarrow (@_i\varphi \vee @_i\psi)$

Because @ is a normal operator we ought to already know that distr$^{@}_{\wedge}$ ($\vdash @_i(\varphi \wedge \psi) \leftrightarrow (@_i\varphi \wedge @_i\psi)$) is a theorem; using this together with the fact that @ is also a self-dual we can also find that it distributes over $\vee$[8].

$$\cfrac{\cfrac{\cfrac{\cfrac{\text{distr}^{@}_{\wedge}}{\vdash @_i(\neg\varphi \wedge \neg\psi) \leftrightarrow (@_i\neg\varphi \wedge @_i\neg\psi)}}{\vdash \neg @_i(\neg\varphi \wedge \neg\psi) \leftrightarrow \neg(@_i\neg\varphi \wedge @_i\neg\psi)}}{\vdash \neg @_i\neg(\varphi \vee \psi) \leftrightarrow (\neg @_i\neg\varphi \vee \neg @_i\neg\psi)}}{\vdash @_i(\varphi \vee \psi) \leftrightarrow (@_i\varphi \vee @_i\psi)}\;\text{SD}_@$$

4.1.4 $\text{T}_\downarrow$: $\vdash \downarrow_i i$

Despite the differences there are also a few striking similarities between $\downarrow$ and @; for example, they are both self-duals. Another trait they share is that they are both reflexive. For @ this is given as an axiom of the logic, for $\downarrow$ we can deduce that $\vdash \downarrow_i i$ holds.

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\text{Intr}}{j \vdash \downarrow_i i \leftrightarrow j}}{j \vdash \downarrow_i i}\;\text{prop}}{@_j j \vdash @_j \downarrow_i i}\;\text{SNec}_@ \qquad \cfrac{\text{T}_@}{\vdash @_j j}}{\vdash @_j \downarrow_i i}\;\text{InfCut}}{\vdash \downarrow_j @_j \downarrow_i i}\;\text{SNec}_\downarrow}{\vdash \downarrow_i i}\;\text{Name}$$

4.1.5 $\text{B}_@$: $\vdash @_i j \leftrightarrow @_j i$

It is very intuitive that if a world named i has name j as well, then world j also has name i, because they are in fact the same world. But intuition is no substitution for proof, so if we want to use it we have to demonstrate its correctness.

The proof consist of first proving one direction of the equality ($\text{B}_@$-half), and then using a new instance of that partial result for the other direction. The use of DedRev[9], which does the reverse of Ded, makes for a slightly shorter proof (by avoiding extra distribution steps).

$$\cfrac{\cfrac{\cfrac{\cfrac{\text{T}_@}{\vdash @_j j} \qquad \cfrac{\cfrac{\cfrac{\cfrac{\text{Intr}}{\vdash i \wedge j \to @_i j}}{i, j \vdash @_i j}\;\text{DedRev}}{@_j i, @_j j \vdash @_j @_i j}\;\text{SNec}_@}{@_j j \vdash @_j i \to @_j @_i j}\;\text{Ded}}{\vdash @_j i \to @_j @_i j}\;\text{InfCut}}{\text{B}_@\text{-half:} \quad \vdash @_j i \to @_i j}\;\text{Agree} \qquad \cfrac{\text{B}_@\text{-half}}{\vdash @_i j \to @_j i}}{\vdash @_i j \leftrightarrow @_j i}$$

[8] As will be shown in the appendix, @ also distributes over $\to$ and $\leftrightarrow$. Similarly, the operator $\downarrow$ distributes over $\vee$, $\wedge$, $\to$ and $\leftrightarrow$.

[9] DedRev: $\Gamma \vdash \varphi \to \psi \;\Rightarrow\; \Gamma, \varphi \vdash \psi$

4.1.6 PR (Paste Rule): $\Gamma, @_i\Diamond j \vdash @_j\varphi \;\Rightarrow\; \Gamma \vdash @_i\Box\varphi$ provided $j \notin \mathtt{fnom}(\Gamma,\varphi) \cup i$

Compared to the previous examples, the **Paste Rule** is much more complex to prove, but it is an important rule because it is often used when proving other theorems. In $\mathcal{H}(@)$ it has to be included in the rules of the logic system, but in $\mathcal{H}(@,\downarrow)$ we can derive it from more basic rules and sequents.

A lot of the steps in the proof simply involve adding or removing @s and ↓s, manipulating sequents into the right form. Even though we already have **SNec**, **Agree** and **Name** to help us in this department, we additionally use **vacuous binding**[10] and **vacuous satisfaction**[11].

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{\mathrm{T}_@}{\vdash @_i i}
\quad
\dfrac{\dfrac{\mathrm{BG}}{i \vdash \Box \downarrow_j @_i\Diamond j}}{@_i i \vdash @_i\Box \downarrow_j @_i\Diamond j}\,\mathrm{SNec}_@
}{\vdash @_i\Box \downarrow_j @_i\Diamond j}\,\mathrm{InfCut}
\quad
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{\text{given } \{j \notin \mathtt{fnom}(\Gamma,\varphi) \cup i\}}{\Gamma, @_i\Diamond j \vdash @_j\varphi}
}{@_k\Gamma, @_k@_i\Diamond j \vdash @_k@_j\varphi}\,\mathrm{SNec}_@,\ \text{fresh } k
}{@_k\Gamma, @_i\Diamond j \vdash @_j\varphi}\,\mathrm{Agree}
}{\downarrow_j @_k\Gamma, \downarrow_j @_i\Diamond j \vdash \downarrow_j @_j\varphi}\,\mathrm{SNec}_\downarrow
}{@_k\Gamma, \downarrow_j @_i\Diamond j \vdash \downarrow_j @_j\varphi}\,\mathrm{VB}
}{@_k\Gamma, \downarrow_j @_i\Diamond j \vdash \varphi}\,\mathrm{Name}
}{\Box @_k\Gamma, \Box \downarrow_j @_i\Diamond j \vdash \Box\varphi}\,\mathrm{SNec}_\Box
\quad
\dfrac{
\dfrac{
\dfrac{\mathrm{Back}}{\vdash \Diamond @_k\neg\Gamma \to @_k\neg\Gamma}
}{\Diamond @_k\neg\Gamma \vdash @_k\neg\Gamma}\,\mathrm{DedRev}
}{@_k\Gamma \vdash \Box @_k\Gamma}\,\mathrm{cp}
}{@_k\Gamma, \Box \downarrow_j @_i\Diamond j \vdash \Box\varphi}\,\mathrm{InfCut}
}{@_i@_k\Gamma, @_i\Box \downarrow_j @_i\Diamond j \vdash @_i\Box\varphi}\,\mathrm{SNec}_@
}{@_k\Gamma, @_i\Box \downarrow_j @_i\Diamond j \vdash @_i\Box\varphi}\,\mathrm{Agree}
}{@_k\Gamma \vdash @_i\Box\varphi}\,\mathrm{InfCut}
}{\Gamma \vdash @_i\Box\varphi}\,\mathrm{VS}
$$

4.1.7 AID (At-Implication-Down equivalence) : $\vdash i \to \varphi \Leftrightarrow \vdash \downarrow_i \varphi \Leftrightarrow \vdash @_i\varphi$

Many theorems come in the form of $\vdash i \to \varphi$, $\vdash \downarrow_i \varphi$ or $\vdash @_i\varphi$. Depending on the situation one variant might be more convenient than another, and so it is very convenient to have a rule that helps transform one kind into another.

The proof for **AID** goes in three parts, from $\vdash @_i\varphi$ to $\vdash i \to \varphi$, to $\vdash \downarrow_i \varphi$, and then back to $\vdash @_i\varphi$. All steps are pretty straightforward, the only thing of note is the use of **vacuous binding after @**[12] in the third case.

$$
\dfrac{
\dfrac{
\dfrac{\text{given}}{\vdash @_i\varphi}
\quad
\dfrac{\mathrm{Intr}}{i, @_i\varphi \vdash \varphi}
}{i \vdash \varphi}\,\mathrm{InfCut}
}{\vdash i \to \varphi}\,\mathrm{Ded}
\qquad
\dfrac{
\dfrac{\mathrm{T}_\downarrow}{\vdash \downarrow_i i}
\quad
\dfrac{
\dfrac{
\dfrac{\text{given}}{\vdash i \to \varphi}
}{i \vdash \varphi}\,\mathrm{DedRev}
}{\downarrow_i i \vdash \downarrow_i \varphi}\,\mathrm{SNec}_\downarrow
}{\vdash \downarrow_i \varphi}\,\mathrm{InfCut}
\qquad
\dfrac{
\dfrac{
\dfrac{\text{given}}{\vdash \downarrow_i \varphi}
}{\vdash @_i \downarrow_i \varphi}\,\mathrm{SNec}_@
}{\vdash @_i\varphi}\,\mathrm{VB}^@
$$

4.2 Generalized rules and sequents

There are some cases where a whole range of sequents or rules can be proven at once by generalizing over certain patterns (usually of modal operators).

[10] **VB:** $\vdash \downarrow_i \varphi \leftrightarrow \varphi$ provided $i \notin \mathtt{fnom}(\varphi)$

[11] **VS:** $@_i\Gamma \vdash \varphi \;\Rightarrow\; \Gamma \vdash \varphi$ and $\Gamma \vdash @_i\varphi \;\Rightarrow\; \Gamma \vdash \varphi$ provided $i \notin \mathtt{fnom}(\Gamma,\varphi)$

[12] **VB$^@$:** $\vdash @_i \downarrow_i \varphi \leftrightarrow @_i\varphi$

Such proofs will either depend on induction, or involve other generalized sequents and rules.

4.2.1 Back$^+$ (generalized back) : $@_i\varphi \vdash \Box^n @_i\varphi$ for $n \geq 0$

One of the simplest examples of a generalized sequent is **Back$^+$**. It involves little more than a repeated application of **Back**, which means we will be making use of induction. For completeness we start with a base case for $n = 0$, even though in practice we would never use that instance.

base case:

$$\dfrac{\text{Taut}}{@_i\varphi \vdash @_i\varphi}$$

With the base case affirmed, we move on to the induction step, which builds on the induction hypothesis: $\forall_{0 \leq m < n}(@_i\varphi \vdash \Box^m @_i\varphi)$

induction step

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{induction hypothesis}}{@_i\varphi \vdash \Box^{n-1}@_i\varphi}}{\Box @_i\varphi \vdash \Box\Box^{n-1}@_i\varphi}\,\text{SNec}_\Box}{\Box @_i\varphi \vdash \Box^n @_i\varphi} \qquad \dfrac{\dfrac{\dfrac{\text{Back}}{\vdash \Diamond @_i\neg\varphi \to @_i\neg\varphi}}{\Diamond @_i\neg\varphi \vdash @_i\neg\varphi}\,\text{DedRev}}{@_i\varphi \vdash \Box @_i\varphi}\,\text{cp}}{@_i\varphi \vdash \Box^n @_i\varphi}\,\text{InfCut}$$

4.2.2 Bridge$^+$ (generalized bridge)

Before stating what sort of sequent **Bridge$^+$** stands for, it is necessary to introduce some notation first.

Let $\{n\}$ denote a specific sequence of $\Box$s and $\Diamond$s. In particular it may be defined as follows

definition $\{n\}$:

$\{1\}\varphi \overset{def}{=} \varphi$

$\{2n\}\varphi \overset{def}{=} \Box\{n\}\varphi$

$\{2n+1\}\varphi \overset{def}{=} \Diamond\{n\}\varphi$

So e.g. $\{314\}\varphi \overset{def}{=} \Box\Diamond\Box\Diamond\Diamond\Diamond\Box\Box\varphi$

One can convert a number to its corresponding sequence by going through the following four steps:

i) convert the number to binary, 314 = 100111010
ii) remove the first digit, 00111010
iii) reverse the string, 01011100
iv) replace the 0s with $\Box$s and 1s with $\Diamond$s, $\Box\Diamond\Box\Diamond\Diamond\Diamond\Box\Box$.

To convert it back, one can do these steps in reverse.

With this notation in place we can now properly state what sequent we wish to prove:

Bridge$^+$: $\{n\}i, @_i\varphi \vdash \{n\}\varphi$ (where $n \geq 1$)

As before this will be an inductive proof, so we start with the base case.

base case, $n = 1$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Intr}}{\vdash i \wedge \neg\varphi \to @_i\neg\varphi}}{i, \neg\varphi \vdash @_i\neg\varphi}\,\text{DedRev}}{i, @_i\varphi \vdash \varphi}\,\text{cp}}{\{1\}i, @_i\varphi \vdash \{1\}\varphi}\,\text{def}\ \{1\}$$

The induction step is split into two distinct cases, one for odd numbers and one for even numbers, respectively. It is important here to use a strong induction hypothesis, because the instance k which is a prerequisite for proving a case n will fall somewhere in the middle of the range in which we hypothesize the theorem to hold. Therefore our induction hypothesis will be: $\forall_{1 \leq m < n}(\{m\}i, @_i\varphi \vdash \{m\}\varphi)$

even n, $n = 2k$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{induction hypothesis}}{\{k\}i, @_i\varphi \vdash \{k\}\varphi}}{\Box\{k\}i, \Box @_i\varphi \vdash \Box\{k\}\varphi}\,\text{SNec}_\Box \qquad \dfrac{\text{Back}^+}{@_i\varphi \vdash \Box @_i\varphi}}{\Box\{k\}i, @_i\varphi \vdash \Box\{k\}\varphi}\,\text{InfCut}}{\{2k\}i, @_i\varphi \vdash \{2k\}\varphi}\,\text{def}\ \{2k\}}{\{n\}i, @_i\varphi \vdash \{n\}\varphi}$$

odd n, $n = 2k + 1$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{induction hypothesis}}{\{k\}i, @_i\varphi \vdash \{k\}\varphi}}{@_i\varphi \vdash \{k\}i \to \{k\}\varphi}\,\text{Ded}}{@_i\varphi \vdash \neg\{k\}\varphi \to \neg\{k\}i}\,\text{cp}}{\Box @_i\varphi \vdash \Box\neg\{k\}\varphi \to \Box\neg\{k\}i}\,\text{SNec}_\Box}{\Box @_i\varphi \vdash \Diamond\{k\}i \to \Diamond\{k\}\varphi}\,\text{cp} \qquad \dfrac{\text{Back}^+}{@_i\varphi \vdash \Box @_i\varphi}}{@_i\varphi \vdash \Diamond\{k\}i \to \Diamond\{k\}\varphi}\,\text{InfCut}}{\Diamond\{k\}i, @_i\varphi \vdash \Diamond\{k\}\varphi}\,\text{DedRev}}{\{2k+1\}i, @_i\varphi \vdash \{2k+1\}\varphi}\,\text{def}\ \{2k+1\}}{\{n\}i, @_i\varphi \vdash \{n\}\varphi}$$

Because every number is either odd or even, this demonstrates that $\{n\}i, @_i\varphi \vdash \{n\}\varphi$ for all $n \geq 1$. In particular this means that given $@_i\varphi$ and a specific sequence of □s and ◇s followed by i, you can deduce the same sequence of □s and ◇s followed by φ.

4.2.3 DW (Distant worlds equivalence): $\Gamma \vdash @_k(\Diamond^n i \to \varphi) \Leftrightarrow \Gamma \vdash @_k\Box^n \downarrow_i @_k\varphi$

The **DW** rule makes a claim of equivalence between two different ways of addressing worlds/states at a certain distance of the current world. Unlike the previous two generalized theorems, this one does not use induction in its proof, instead it relies on the **generalized Paste Rule**[13] and **generalized bridge**.

The proof consists of two parts, first in one direction, and then the other. However, the latter is little more than the reverse of the first.

$$\Rightarrow \qquad \begin{array}{cl} \text{given} & \\ \hline \Gamma \vdash @_k(\Diamond^n i \to \varphi) & \\ \hline \Gamma \vdash @_k\Diamond^n i \to @_k\varphi & \text{Distr}^{@}_{\to} \\ \hline \Gamma, @_k\Diamond^n i \vdash @_k\varphi & \text{DedRev} \\ \hline \Gamma, @_k\Diamond^n i \vdash @_i @_k\varphi & \text{Agree} \\ \hline \Gamma, @_k\Diamond^n i \vdash @_j \downarrow_i @_k\varphi & \text{VB}^{@} \\ \hline \Gamma \vdash @_k\Box^n \downarrow_i @_k\varphi & \text{PR}^{+} \end{array} \qquad\qquad \Leftarrow \qquad \begin{array}{cl} \text{given} & \\ \hline \Gamma \vdash @_k\Box^n \downarrow_i @_k\varphi & \\ \hline \Gamma, @_k\Diamond^n i \vdash @_i \downarrow_i @_k\varphi & \text{PR}^{+} \\ \hline \Gamma, @_k\Diamond^n i \vdash @_i @_k\varphi & \text{VB}^{@} \\ \hline \Gamma, @_k\Diamond^n i \vdash @_k\varphi & \text{Agree} \\ \hline \Gamma \vdash @_k\Diamond^n i \to @_k\varphi & \text{Ded} \\ \hline \Gamma \vdash @_k(\Diamond^n i \to \varphi) & \text{Distr}^{@}_{\to} \end{array}$$

4.3 Frames

One of the motivations behind developing hybrid logic is the greater number of frame properties one can define in it; at the same time it is desirable not to lose any important results attained in modal logic. In [18] it is shown that in $\mathcal{H}(@)$ the completeness result for extensions with pure axioms cannot be combined with completeness result for Sahlqvist axioms [16, 12, 10].

For $\mathcal{H}(@, \downarrow)$ the story is different. In this section the correspondence between a number of pure and modal axioms will be shown (the third is most notable, because it includes the counterexample for $\mathcal{H}(@)$ used in [18]). For Sahlqvist axioms in general the correspondence with hybrid formulas is shown in the next chapter. However, the proof method used there deviates quite a bit from the beaten path.

4.3.1 Reflexivity

The first frame class we will look at is the reflexive class (Ch. 1.3). In a reflexive frame each world has a connection back to itself. In modal logic we have the axiom $\Box\varphi \to \varphi$ to characterize this class. There are a number of simple axioms in hybrid logic that serve the same purpose, the usual two are $i \to \Diamond i$ or $@_i\Diamond i$ (which are easily shown equivalent with the **AIM** rule).

Demonstrating the equivalence is split into two parts, first from modal to hybrid and then back. The first case often (but by no means always) comes down to a proper instantiation of the modal axiom and possibly using contraposition or a bit of propositional magic. This is the case here; to get from $\Box\varphi \to \varphi$ to $i \to \Diamond i$, we simply instantiate φ with $\neg i$ and contrapose the result.

[13] **PR**$^{+}$: $\Gamma, @_i\Diamond^n j \vdash @_j\varphi \;\; \Leftrightarrow \;\; \Gamma \vdash @_i\Box^n\varphi$ **provided** $j \notin fnom(\Gamma, \varphi) \cup \{i\}$

The other direction, $\vdash @_i\Diamond i \Rightarrow \vdash \Box\varphi \to \varphi$ is more involved:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Bridge } \{ \text{ with } i \notin nom(\varphi) \ \}}{\Diamond i, \Box\neg\varphi \vdash @_i\neg\varphi}}{\Diamond i, \neg @_i\neg\varphi \vdash \neg\Box\neg\varphi}}{\Diamond i, @_i\varphi \vdash \Diamond\varphi}}{@_i\Diamond i, @_i\varphi \vdash @_i\Diamond\varphi}\,\text{SNec}_{@}\text{, Agree} \qquad \dfrac{\text{given}}{\vdash @_i\Diamond i}}{@_i\varphi \vdash @_i\Diamond\varphi}\,\text{InfCut}}{\vdash @_i\varphi \to @_i\Diamond\varphi}\,\text{Ded}}{\vdash @_i(\varphi \to \Diamond\varphi)}\,\text{Distr}^{@}_{\to}}{\vdash \varphi \to \Diamond\varphi}\,\text{VS}$$

4.3.2 Determinism

Our next example deals with deterministic (or partly functional) frames (Ch. 1.3). The usual modal axiom for characterizing determinism is $\Diamond\varphi \to \Box\varphi$. Instantiating φ with i gives a hybrid axiom that seems to convey what we want; but we still need to demonstrate that we can deduce the modal axiom from the hybrid one.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Bridge}^{+}\ \{i \notin \mathtt{fnom}(\varphi)\}}{\Box i, @_i\neg\varphi \vdash \Box\neg\varphi}}{\Box i, \Diamond\varphi \vdash @_i\varphi}\,\text{cp, SD}_{@} \qquad \dfrac{\dfrac{\text{given}}{\vdash \Diamond i \to \Box i}}{\Diamond i \vdash \Box i}\,\text{DedRev}}{\Diamond i, \Diamond\varphi \vdash @_i\varphi}\,\text{InfCut}}{@_k\Diamond i, @_k\Diamond\varphi \vdash @_k @_i\varphi}\,\text{SNec}_{@}\text{, fresh k}}{@_k\Diamond i, @_k\Diamond\varphi \vdash @_i\varphi}\,\text{Agree}}{@_k\Diamond\varphi \vdash @_k\Box\varphi}\,\text{PR}}{\vdash @_k\Diamond\varphi \to @_k\Box\varphi}\,\text{Ded}}{\vdash @_k(\Diamond\varphi \to \Box\varphi)}\,\text{Distr}^{@}_{\to}}{\vdash \Diamond\varphi \to \Box\varphi}\,\text{VS}$$

However, aside from the obvious choice of hybrid axiom, there is also one more reminiscent of the underlying first-order logic frame condition $(\forall_x\forall_y\forall_z(R_{xy} \wedge R_{xz}) \to y = z)$, namely the hybrid axiom $\Diamond i \wedge \Diamond j \to @_i j$. Proving that it is equivalent to $\Diamond\varphi \to \Box\varphi$ and $\Diamond i \to \Box i$ will require a bit more work, because a simple instantiation will not suffice for either direction.

To keep things simple, we will prove equivalence with the earlier hybrid axiom, $\Diamond i \to \Box i$. This requires less effort than using its modal counterpart $\Diamond\varphi \to \Box\varphi$; and, seeing as we already know those two are equivalent, either is fine.

The proof is split in two parts. First we demonstrate that it holds that: $\vdash \Diamond i \wedge \Diamond j \to @_i j \Rightarrow \vdash \Diamond i \to \Box i$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{given}}{\vdash \Diamond i \wedge \Diamond j \to @_j i}}{\Diamond i, \Diamond j \vdash @_j i}\,\text{DedRev}}{@_k\Diamond i, @_k\Diamond j \vdash @_k @_j i}\,\text{SNec}_{@}}{@_k\Diamond i, @_k\Diamond j \vdash @_j i}\,\text{Agree}}{@_k\Diamond i, \vdash @_k\Box i}\,\text{PR}}{\vdash @_k\Diamond i \to @_k\Box i}\,\text{Ded}}{\vdash @_k(\Diamond i \to \Box i)}\,\text{Distr}^{@}_{\to}}{\vdash \Diamond i \to \Box i}\,\text{VS}$$

Next we show that the reverse also holds: $\vdash \Diamond i \rightarrow \Box i \Rightarrow \vdash \Diamond i \wedge \Diamond j \rightarrow @_i j$

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{given}}{\vdash \Diamond j \rightarrow \Box j}}{\Diamond j \vdash \Box j}\,\text{DedRev} \qquad \dfrac{\text{Bridge}}{\Diamond i, \Box j \vdash @_i j}\,\text{InfCut}}{\Diamond i, \Diamond j \vdash @_i j}}{\vdash \Diamond i \wedge \Diamond j \rightarrow @_i j}\,\text{Ded}$$

4.3.3 The General Axiom

The General Axiom (G) is a generalization of frame axioms, and was found by Lemmon and Scott[13]. It encompasses many of the common frame axioms; amongst which the previous two examples. In its modal form the axiom reads: **G**: $\Diamond^h \Box^i \varphi \rightarrow \Box^j \Diamond^k \varphi$. The corresponding frame condition is $R^h_{wv} \wedge R^j_{wu} \rightarrow \exists_x (R^i_{vx} \wedge R^k_{ux})$.

To dispel the possible illusion that we can always find a hybrid equivalent by just instantiating φ in **G** appropriately, we will first look at the case $h = i = j = k = 1$, which gives weak directionality. If we replace φ with i or $\neg i$ (with contraposition) we get $\Diamond\Box i \rightarrow \Box\Diamond i$ in both cases. While at first this might seem reasonable, it does not actually correspond with weak directionality.

The easiest way to demonstrate this is to give a frame where one is true, but the other is not. In figure 4.1 we see just this, $\Diamond\Box i \rightarrow \Box\Diamond i$ is true in the frame underlying the model ($\Diamond\Box i$ is true nowhere), but the frame is clearly not weakly directional.

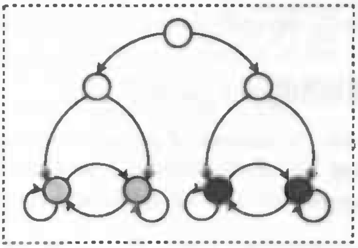


Figure 4.1, A counterexample to $\Diamond\Box i \rightarrow \Box\Diamond i$ as hybrid axiom for the property weakly directional.

Fortunately, the other approach, looking to the first-order frame condition, does help us find a hybrid formula. It leads us to the hybrid axiom $\Diamond^h v \wedge \Diamond^j u \rightarrow @_u \Diamond^k \downarrow_x @_v \Diamond^i x$. In general the $\downarrow$ cannot be eliminated from this formula (it can if i or k are 0, and for certain h, i, j, k in some frames that have additional frame properties). This means that in general there is not a hybrid correspondent in $\mathcal{H}(@)$, but there is in $\mathcal{H}(@, \downarrow)$.

As before, the proof falls into two parts. Our first step we will be to demonstrate that we can deduce the modal axiom from the hybrid one, i.e.

$\vdash \Diamond^j u \wedge \Diamond^h v \to @_u\Diamond^k \downarrow_x @_v\Diamond^i x \;\Rightarrow \vdash \Diamond\Box\varphi \to \Box\Diamond\varphi$. For brevity's sake a number of steps have been joined together.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{given}}{\vdash \Diamond^j u \wedge \Diamond^h v \to @_u\Diamond^k \downarrow_x @_v\Diamond^i x}}{\Diamond^j u, \Diamond^h v \vdash @_u\Diamond^k \downarrow_x @_v\Diamond^i x}\,\text{DedRev} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Bridge } \{x \notin nom(\varphi)\}}{\Diamond^i x, \Box^i\varphi \vdash @_x\varphi}}{@_v\Diamond^i x, @_v\Box^i\varphi \vdash @_x\varphi}\,\text{SNec}_@\text{, Agree}}{\downarrow_x @_v\Diamond^i x, \downarrow_x @_v\Box^i\varphi \vdash \downarrow_x @_x\varphi}\,\text{SNec}_\downarrow}{\downarrow_x @_v\Diamond^i x, @_v\Box^i\varphi \vdash \varphi}\,\text{VB, Name}}{\Diamond^k \downarrow_x @_v\Diamond^i x, \Box^k @_v\Box^i\varphi \vdash \Diamond^k\varphi}\,\text{SNec}_\Box^{(k)}}{\Diamond^k \downarrow_x @_v\Diamond^i x, @_v\Box^i\varphi \vdash \Diamond^k\varphi}\,\text{Back}^+}{@_u\Diamond^k \downarrow_x @_v\Diamond^i x, @_v\Box^i\varphi \vdash @_u\Diamond^k\varphi}\,\text{SNec}_@\text{; Agree}}{\Diamond^j u, \Diamond^h v, @_v\Box^i\varphi \vdash @_u\Diamond^k\varphi}\,\text{InfCut}}{@_w\Diamond^j u, @_w\Diamond^h v, @_v\Box^i\varphi \vdash @_u\Diamond^k\varphi}\,\text{SNec}_@\ \{\text{fresh } w\}\text{, Agree}}{@_w\Diamond^h v, @_v\Box^i\varphi \vdash @_w\Box^j\Diamond^k\varphi}\,\text{PR}^+}{@_w\Diamond^h v, @_w\neg\Box^j\Diamond^k\varphi \vdash @_v\neg\Box^i\varphi}\,\text{cp}}{@_w\neg\Box^j\Diamond^k\varphi \vdash @_w\Box^h\neg\Box^i\varphi}\,\text{PR}^+}{\neg\Box^j\Diamond^k\varphi \vdash \Box^h\neg\Box^i\varphi}\,\text{VS}^{(2)}}{\vdash \neg\Box^j\Diamond^k\varphi \to \Box^h\neg\Box^i\varphi}\,\text{Ded}}{\vdash \Diamond^h\Box^i\varphi \to \Box^j\Diamond^k\varphi}\,\text{cp}$$

Next we consider $\vdash \Diamond\Box\varphi \to \Box\Diamond\varphi \;\Rightarrow \vdash \Diamond^j u \wedge \Diamond^h v \to @_u\Diamond^k \downarrow_x @_v\Diamond^i x$, the reverse of the last step; fortunately this is a lot less involved. One detail that warrants mention is the use of **generalized Bounded Generalization** [14], which has not been introduced before now.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{BG}^+}{v \vdash \Box^i \downarrow_x @_v\Diamond^i x}}{\Diamond^h v \vdash \Diamond^h\Box^i \downarrow_x @_v\Diamond^i x}\,\text{SNec}_\Box^{(h)} \qquad \dfrac{\dfrac{\dfrac{\text{given}}{\vdash \Diamond^h\Box^i \downarrow_x @_v\Diamond^i x \to \Box^j\Diamond^h \downarrow_x @_v\Diamond^i x}}{\Diamond^h\Box^i \downarrow_x @_v\Diamond^i x \vdash \Box^j\Diamond^k \downarrow_x @_v\Diamond^i x}\,\text{DedRev} \qquad \dfrac{\dfrac{\text{Bridge}^+}{\Diamond^j u, @_u\Box^k \downarrow_x @_v\Box^i\neg x \vdash \Diamond^j\Box^k \downarrow_x @_v\Box^i\neg x}}{\Diamond^j u, \Box^j\Diamond^k \downarrow_x @_v\Diamond^i x \vdash @_u\Diamond^k \downarrow_x @_v\Diamond^i x}\,\text{cp}}{\Diamond^j u, \Diamond^h\Box^i \downarrow_x @_v\Diamond^i x \vdash @_u\Diamond^k \downarrow_x @_v\Diamond^i x}\,\text{InfCut}}{\Diamond^j u, \Diamond^h v \vdash @_u\Diamond^k \downarrow_x @_v\Diamond^i x}\,\text{InfCut}}{\vdash \Diamond^j u \wedge \Diamond^h v \to @_u\Diamond^k \downarrow_x @_v\Diamond^i x}\,\text{Ded}$$

4.4 Examples of infinitary proofs

We end this chapter with a number of proof examples that involve infinite sequents, finally making use of the infinitary part in infinitary hybrid logic. Each of the three examples involves an infinitary axiom sequent taken from chapter 5 of [11].

4.4.1

One very simple example of a proof in infinitary hybrid logic is to show that there are cycles in a reflexive frame. In formal notation, we wish to show that: $\mathcal{H}_\omega(@, \downarrow) + i \to \Diamond i \;\Rightarrow\; AS3 : \{\neg @_i\Diamond^n i | n \in \mathbb{N} \wedge n \geq 1\} \vdash \bot$

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{axiom}}{\vdash i \to \Diamond i}}{\vdash @_i\Diamond i}\,\text{AID} \qquad \dfrac{\text{prop}}{@_i\Diamond i, \neg @_i\Diamond i \vdash \bot}}{\neg @_i\Diamond i \vdash \bot}\,\text{Cut}}{\{\neg @_i\Diamond^n i | n \in \mathbb{N} \wedge n \geq 1\} \vdash \bot}\,\text{W}$$

[14] **BG**$^+$: $\vdash @_i\Box^k \downarrow_j @_i\Diamond^k j$ (or, as in this case, by implicit use of **AIM** and **DedRev**, $i \vdash \Box^k \downarrow_j @_i\Diamond^k j$)

4.4.2

A slightly more difficult infinitary proof involves demonstrating that reachability (excluding steps of size zero) implies seriality; i.e. we have that $\mathcal{H}_\omega(@,\downarrow) + AS2^+ : \{\neg @_i \Diamond^k j | k \in \mathbb{N} \wedge k \geq 1\} \vdash \bot \;\Rightarrow \vdash \Diamond\top$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Taut}}{\bot \vdash \{\Box^n \neg j | n \in \mathbb{N}\}}}{\Box\bot \vdash \Box\{\Box^n \neg j | n \in \mathbb{N}\}}\,\text{SNec}_\Box}{@_i\Box\bot \vdash @_i\Box\{\Box^n \neg j | n \in \mathbb{N}\}}\,\text{SNec}_@}{@_i\Box\bot \vdash \{@_i\Box^k \neg j | k \in \mathbb{N} \wedge k \geq 1\}}}{@_i\Box\bot \vdash \{\neg @_i \Diamond^k j | k \in \mathbb{N} \wedge k \geq 1\}} \qquad \dfrac{\text{given}}{\{\neg @_i \Diamond^k j | k \in \mathbb{N} \wedge k \geq 1\} \vdash \bot}}{@_i\Box\bot \vdash \bot}\,\text{InfCut}}{\vdash \neg @_i \Box\bot}\,\text{Ded}}{\vdash @_i\Diamond\top}}{\vdash \downarrow_i @_i\Diamond\top}\,\text{SNec}_\downarrow}{\vdash \Diamond\top}\,\text{Name}$$

4.4.3

The last example is in quite a different class than the earlier two, and requires quite a bit more creativity. The goal here is to prove that the Löb formula follows from the bounded chain condition under transitivity; $\mathcal{H}_\omega(@,\downarrow) + AS4 : \{\Diamond^n\top | n \in \mathbb{N}\} \vdash \bot + \Diamond\Diamond i \rightarrow \Diamond i \;\Rightarrow \vdash \Box(\Box\varphi \rightarrow \varphi) \rightarrow \Box\varphi$

It should be possible to prove this because it is known that the Löb formula characterizes transitive, converse well-founded frames; and BCC is a stronger condition than converse well-foundedness.

We start with a bold claim on which the rest of our proof hinges. Note that the premise of the sequent is the negation of the Löb formula; so in the end we want to show that the conclusion of the sequent leads to a contradiction with the rest of the logic (notably $AS4$).

Claim 1: $\Box(\Box\varphi \rightarrow \varphi), \Diamond\neg\varphi \vdash \Box^k(\Box\varphi \rightarrow \varphi) \wedge \Diamond^k\neg\varphi \wedge \{\Diamond\top | i \in [0..k]\}$ for all $k \in \mathbb{N}^+$

To prove this claim we will need to employ induction. The base case is quite simple; it follows from conjoining various propositional tautologies

Base case:

$$\dfrac{\dfrac{\text{Taut}}{\Box(\Box\varphi \rightarrow \varphi), \Diamond\neg\varphi \vdash \Box(\Box\varphi \rightarrow \varphi) \wedge \Diamond\neg\varphi} \qquad \dfrac{\dfrac{\dfrac{\text{Taut}}{\bot \vdash \varphi}}{\Box\bot \vdash \Box\varphi}\,\text{SNec}_\Box}{\Diamond\neg\varphi \vdash \Diamond\top}\,\text{cp} \qquad \dfrac{\text{Taut}}{\vdash \top}}{\Box(\Box\varphi \rightarrow \varphi), \Diamond\neg\varphi \vdash \Box(\Box\varphi \rightarrow \varphi) \wedge \Diamond\neg\varphi \wedge \top \wedge \Diamond\top}\,\text{con}$$

The induction step is not much different; first, we show how to make a single step from case $k-1$ to k, and then apply this to the induction hypothesis.

Induction step: $(k > 1)$

The induction hypothesis we assume in this induction is the following: $\Box(\Box\varphi \to \varphi), \Diamond\neg\varphi \vdash \Box^j(\Box\varphi \to \varphi) \wedge \Diamond^j\neg\varphi \wedge \{\Diamond^i\top | i \in [0..j]\}$ for all $0 < j < k$

$$\dfrac{\dfrac{\text{frame transitivity}}{\Box^{k-1}(\Box\varphi \to \varphi) \vdash \Box^k(\Box\varphi \to \varphi)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\text{Taut}}{\Box\varphi, (\Box\varphi \to \varphi) \vdash \varphi}}{\Box^k\varphi, \Box^{k-1}(\Box\varphi \to \varphi) \vdash \Box^{k-1}\varphi}\,\text{SNec}_\Box^{(k-1)}}{\Diamond^{k-1}\neg\varphi, \Box^{k-1}(\Box\varphi \to \varphi) \vdash \Diamond^k\neg\varphi}\,\text{cp} \qquad \dfrac{\dfrac{\dfrac{\text{Taut}}{\bot \vdash \varphi}}{\Box^k\bot \vdash \Box^k\varphi}\,\text{SNec}_\Box^{(k)}}{\Diamond^k\neg\varphi \vdash \Diamond^k\top}\,\text{cp}}{\Box^{k-1}(\Box\varphi \to \varphi), \Diamond^{k-1}\neg\varphi \vdash \Diamond^k\neg\varphi \wedge \Diamond^k\top}\,\text{con}}{\dfrac{\Box^{k-1}(\Box\varphi \to \varphi), \Diamond^{k-1}\neg\varphi \vdash \Box^k(\Box\varphi \to \varphi) \wedge \Diamond^k\neg\varphi \wedge \Diamond^k\top}{\text{OneStep:}\quad \Box^{k-1}(\Box\varphi \to \varphi), \Diamond^{k-1}\neg\varphi, \{\Diamond^i\top | i \in [0..k-1]\} \vdash \Box^k(\Box\varphi \to \varphi) \wedge \Diamond^k\neg\varphi \wedge \{\Diamond^i\top | i \in [0..k]\}}}\,\text{con}$$

$$\dfrac{\dfrac{\text{induction hypothesis}}{\Box(\Box\varphi \to \varphi), \Diamond\neg\varphi \vdash \Box^{k-1}(\Box\varphi \to \varphi) \wedge \Diamond^{k-1}\neg\varphi \wedge \{\Diamond^i\top | i \in [0..k-1]\}} \qquad \text{OneStep}}{\Box(\Box\varphi \to \varphi), \Diamond\neg\varphi \vdash \Box^k(\Box\varphi \to \varphi) \wedge \Diamond^k\neg\varphi \wedge \{\Diamond^i\top | i \in [0..k]\}}\,\text{InfCut}$$

Therefore, having proved claim 1, we have in particular (by weakening the conclusion) that $\Box(\Box\varphi \to \varphi), \Diamond\neg\varphi \vdash \{\Diamond\top | i \in [0..k]\}$ for all $k \in \mathbb{N}^+$, and thus $\Box(\Box\varphi \to \varphi), \Diamond\neg\varphi \vdash \{\Diamond^n\top | n \in \mathbb{N}\}$. With this we can finish our proof.

$$\dfrac{\dfrac{\dfrac{\text{Claim1}}{\Box(\Box\varphi \to \varphi), \Diamond\neg\varphi \vdash \{\Diamond^n\top | n \in \mathbb{N}\}} \qquad \dfrac{\text{AS4}}{\{\Diamond^n\top | n \in \mathbb{N}\} \vdash \bot}}{\Box(\Box\varphi \to \varphi), \Diamond\neg\varphi \vdash \bot}\,\text{InfCut}}{\vdash \Box(\Box\varphi \to \varphi) \to \Box\varphi}\,\text{Ded}^{(2)}$$

5 Sahlqvist formulas in $\mathcal{H}(@,\downarrow)$

An important class of formulas in modal logics is the class of the Sahlqvist formulas [5, 16]. Sahlqvist formulas follow a particular structure which translates to important correspondence results. Importantly, 1) a Sahlqvist formula is canonical (in the sense that it characterizes a frame class), 2) the class of frames corresponding to a Sahlqvist formula is first-order definable, and 3) the corresponding frame condition can be effectively computed from a given Sahlqvist formula.

Because Sahlqvist formulas are canonical, and encompass many formulas of interest, they can be useful for determining completeness results. For example, if the basic modal system K is extended with only Sahlqvist axioms, the resulting logic will always be complete.

In [18] it is shown that combining the completeness result for pure formulas in $\mathcal{H}(@)$ and the completeness result for Sahlqvist formulas is problematic: we are not guaranteed completeness if we extend a system with both types of axioms. When our hybrid language also includes the $\downarrow$-operator, however, it is a different story. It will be shown that in $\mathcal{H}(@,\downarrow)$ every Sahlqvist formula is provably equivalent to a pure formula, and therefore we retain completeness when a system is extended with both types of axioms.

This result applies to every hybrid logic based on the same semantics for @ and $\downarrow$, as long as the completeness result for pure formulas applies to it. So in particular it also applies to the infinitary hybrid logic discussed in chapter 3.

This section aims to outline how Sahlqvist formulas correspond with pure formulas in $\mathcal{H}(@,\downarrow)$, and conversely what structure we may expect from a pure formula which can be translated to a Sahlqvist formula. The latter turns out to to be closely tied to Kracht formulas [5, 12].

5.1 Sahlqvist formulas

The full breadth of the work surrounding Sahlqvist formulas is too much to cover here. I will limit myself to the structure of Sahlqvist formulas, and briefly outline how to translate one to a (local) first-order correspondent. Why the structure of the Sahlqvist fragment is the way it is, and why the translation works will be left open (but see in particular [5] and its preceding section).

The structure of Sahlqvist formulas is best given as a grammar in Backus-Naur form [15]:

[15] Unfortunately none of the accessible resources seem to agree with me on that count, and prefer to give an informal account in words only.

```
positive_formula ::= ⊤ | ⊥ | p_i |
                     ¬ negative_formula |
                     □ positive_formula |
                     ◇ positive_formula |
                     positive_formula ∧ positive_formula |
                     positive_formula ∨ positive_formula |
                     negative_formula → positive_formula

negative_formula ::= ⊤ | ⊥ |
                     ¬ positive_formula |
                     □ negative_formula |
                     ◇ negative_formula |
                     negative_formula ∧ negative_formula |
                     negative_formula ∨ negative_formula |
                     positive_formula → negative_formula


boxed_atom ::= p | □ boxed_atom

Sahlqvist_antecedent ::= boxed_atom |
                         negative_formula |
                         ◇ Sahlqvist_antecedent |
                         Sahlqvist_antecedent ∧ Sahlqvist_antecedent |
                         Sahlqvist_antecedent ∨ Sahlqvist_antecedent


Sahlqvist_implication ::= Sahlqvist_antecedent → positive_formula

Sahlqvist_formula ::= Sahlqvist_implication |
                      □ Sahlqvist_formula |
                      Sahlqvist_formula ∧ Sahlqvist_formula |
                      Sahlqvist_formula ∨ Sahlqvist_formula {when there
                        are no shared variables}
```

To simplify the notation of first-order and second-order formulas, we will be using the notion of restricted quantifiers. Restricted quantifiers have the form $\forall_{y \rhd x}$ and $\exists_{y \rhd x}$ and are defined in the following way:

$\forall_{y \rhd x}\varphi(x,y) \overset{def}{=} \forall y(R_{xy} \to \varphi(x,y))$ and

$\exists_{y \rhd x}\varphi(x,y) \overset{def}{=} \exists y(R_{xy} \wedge \varphi(x,y))$

Restricted quantifiers correspond directly to the modal operators in the modal language.

In various places $\forall^r$ and $\exists^r$ are used to denote a restricted quantifier without specifying what variable is restricting it, and Q^r is used when in addition the type of quantifier is left unspecified.

The first step in finding a local first-order correspondent for a Sahlqvist formula is to reduce the problem to finding the correspondents of a (number of) Sahlqvist implications. To do this we use three rules that split the formula along the lines it is built up:

- If φ is a Sahlqvist formula corresponding to a first-order frame condition $\alpha(x)$ then $\forall_{y \rhd x}\alpha[y/x]$ is the correspondent of the formula $\Box\varphi$
- If φ corresponds to α and ψ to β then $\varphi \wedge \psi$ has the correspondent $\alpha \wedge \beta$
- If φ corresponds to α and ψ to β, and furthermore φ and ψ have no variables in common , then $\varphi \vee \psi$ corresponds with $\alpha \vee \beta$

What we are now left with is the task of finding correspondents for a number of Sahlqvist implications.

The second step is to translate the Sahlqvist implications to second-order formulas. Recall that a Sahlqvist implication has the form $\varphi \to \psi$, where φ is a Sahlqvist antecedent and ψ is a positive formula. The main translation is defined as follow:

$$
\begin{aligned}
&ST_x(\neg\varphi) = \neg ST_x(\varphi)\\
&ST_x(\varphi \oplus \psi) = ST_x(\varphi) \oplus ST_x(\psi) \text{ where } \oplus \in \{\wedge, \vee, \to, \leftrightarrow\}\\
&ST_x(\Diamond\varphi) = \exists_{y \rhd x} ST_y(\varphi)\\
&ST_x(\Box\varphi) = \forall_{y \rhd x} ST_y(\varphi)\\
&ST_x(p_i) = P_i(x)
\end{aligned}
$$

We now get a second-order formula $\forall P_1 \ldots \forall P_n(ST_{x_0}(\varphi) \to \text{POS})$, where POS is the translation of the positive formula ψ, and there is a predicate P_i for each propositional variable p_i. (In the rest of the section quantifier sequences like $\forall P_1 \ldots \forall P_n$ will often be abbreviated as $\forall\bar{P}$.)

Next we deal with the translation of the Sahlqvist antecedent $(ST_{x_0}(\varphi))$. Recall that a Sahlqvist antecedent is built up from negative formulas and boxed atoms by applying conjunction, disjunction and diamonds. The goal now is to remove the diamonds and the disjunctions.

We can pull out the diamonds using the equivalences

$$
\begin{aligned}
(\exists^r x_i \alpha(x_i) \wedge \beta) &\leftrightarrow \exists^r x_i(\alpha(x_i) \wedge \beta) \quad \text{and}\\
(\exists^r x_i \alpha(x_i) \to \beta) &\leftrightarrow \forall^r x_i(\alpha(x_i) \to \beta)
\end{aligned}
$$

And at the same time, as we come across them, we can split up disjunctions[16] using

$$
\begin{aligned}
((\alpha \vee \beta) \to \gamma) &\leftrightarrow ((\alpha \to \gamma) \wedge (\beta \to \gamma)) \quad \text{and}\\
\forall\bar{P}\forall^r\bar{x}(\alpha \wedge \beta) &\leftrightarrow (\forall\bar{P}\forall^r\bar{x}.\alpha \wedge \forall\bar{P}\forall^r\bar{x}.\beta)
\end{aligned}
$$

We end up with a conjunction of formulas $\forall\bar{P}\forall^r\bar{x}(\text{BOX-AT} \wedge \text{NEG} \to \text{POS})$ where BOX-AT is a conjunction of boxed atoms and NEG is a conjunction of (translations of) negative formulas. We can then bring NEG to the other side of the implication where it joins POS as a positive formula. This yields $\forall\bar{P}\forall^r\bar{x}(\text{BOX-AT} \to \text{POS} \vee \neg\text{NEG})$.

[16] Disjunctions of negative formulas do not have to be split, because such a disjunction is itself a negative formula.

Here all our work comes to fruition. We get to eliminate the predicates P_i, which will leave us with the first-order formula that has been our goal. To eliminate the predicates we need to find the appropriate instances. First we can look at any predicates which only occur in the consequent (POS$\vee\neg$NEG). For these we substitute $\sigma(P) \equiv \lambda u.u \neq u$.

The rest of the instances will be read off from BOX-AT. We need the minimal instances that will make the antecedent true. We take the set of (translations of) boxed atoms from BOX-AT in which P_i occurs, which have either the form $P_i(x_j)$ or $\forall_{y \rhd^n x_j} P_i(y)$. In the first case we add $u = x_j$ and in the second $R^n_{x_j u}$, and so the complete substitution will look like $\sigma(P) = \lambda u.(u = x_j \vee \ldots \vee u = x_m \vee R^{n_k}_{x_k u} \vee \ldots \vee R^{n_l}_{x_l u})$.

After applying these substitutions we have our first-order formula and all that is left is to simplify it as desired.

5.2 Kracht formulas

In the previous section we defined the class of Sahlqvist formulas and laid out an algorithm that would yield their local first-order correspondent. In this section we will look at the other direction. We define a class of first-order formulas, the Kracht formulas, and give an algorithm that provides their modal correspondent. It should not come as a surprise that there is more to it than just this similarity; the modal correspondent of a Kracht formula is a Sahlqvist formula, and the first-order correspondent to a Sahlqvist formula is a Kracht formula.

Kracht formulas are first-order formulas that are built up from the atomic formulas $u \neq u$, $u = u$, $x = y$ and R_{xy} using the connectives $\wedge$, $\vee$ and the restricted quantifiers $\forall_{y \rhd x}$, $\exists_{y \rhd x}$. Furthermore, all atomic formulas other than $u \neq u$ and $u = u$ should contain at least one inherently universal variable, i.e. a variable that is either free, or bound by a restricted universal quantifier that does not lie within the scope of an existential quantifier.

The first step in finding a modal correspondent to our Kracht formula is rewriting it to a normal form. For this we define type 1 formulas, which have the form

$$\forall^r x_1 \ldots \forall^r x_n Q^r_1 y_1 \ldots Q^r_m y_m.\beta(x_0,\ldots,x_n,y_1,\ldots,y_m) \quad \text{(with } n,m \geq 0\text{)}$$

Each bound variable is restricted by an earlier variable; so every x_i is restricted by an x_j such that $j < i$, and every y_i is either restricted by an x_k or an y_j with $j < i$. The quantifier-free formula β is a disjunctive normal form built from $u = u$, $u \neq u$, R_{ux}, $u = x$ and R_{xu}; the only atomic formulas that are excluded are $R_{yy'}$ and $y = y'$ (typically Q^r_1 is $\exists^r$ and so none of the y_i are inherently universal, whereas all x_i are).

Type 1 formulas form a special subclass of Kracht formula, and importantly each Kracht formula can be rewritten to a type 1 formula by pulling quantifiers to the front, using the equivalences:

$$(\mathsf{Q}^r u.\beta) \vee \gamma \leftrightarrow \mathsf{Q}^r u.(\beta \vee \gamma) \quad \text{and} \quad (\mathsf{Q}^r u.\beta) \wedge \gamma \leftrightarrow \mathsf{Q}^r u.(\beta \wedge \gamma)$$

Where $u \notin \mathtt{var}(\gamma)$.

It is, however, important to be careful about choosing the order in which the quantifiers are pulled out. If, for example, we take the Kracht formula $\forall_{x_1 \rhd x_0} \forall_{x_2 \rhd x_1}(x_1 = x_2) \wedge \exists_{y_1 \rhd x_0}(x_0 = y_1)$, then it can be rewritten in two ways: either to $\forall_{x_1 \rhd x_0}[\forall_{x_2 \rhd x_1}(x_1 = x_2) \wedge \exists_{y_1 \rhd x_0}(x_0 = y_1)]$, or otherwise to $\exists_{y_1 \rhd x_0}[x_0 = y_1 \wedge \forall_{x_1 \rhd x_0} \forall_{x_2 \rhd x_1}(x_1 = x_2)]$, unfortunately the latter is no longer a Kracht formula. To avoid this problem from occurring you have to pull out only universal quantifiers until you have ascertained each atomic formula (other than $u = u$ and $u \neq u$) contains an inherently universal variable; that is, either x_0 or a variable that is bound by one of the universal quantifiers you have just pulled out.

Once all restricted quantifiers have been pulled to the front, we have a formula of the form $\forall^r x_1 \ldots \forall^r x_n \mathsf{Q}^r_1 y_1 \ldots \mathsf{Q}^r_m y_m.\alpha(x_0, \ldots, x_n, y_1, \ldots, y_m)$ where α is built from atomic formulas using $\vee$ and $\wedge$. All that is now needed to turn this into a type 1 formula is rewriting α into disjunctive normal form.

After having obtained the type 1 formula we can proceed with the next step, turning our first-order formula into a second-order formula from which we can ultimately read off the the modal formula which corresponds to our type 1 formula. As before we will work toward a special class of formulas; in this case type 2 formulas, which have the form:

$$\tilde{\forall} \bar{P} \tilde{\forall} \bar{Q} \forall^r \bar{x} \left(\bigwedge_{0 \leq i \leq n} ST_{x_i}(\sigma_i) \rightarrow \beta \right)$$

Here σ_i is a conjunction of boxed atoms in p_i and q_i and β is a DNF of formulas $ST_x(\varphi)$ where φ is a modal formula positive in each p_i, q_j. Essentially it is a type we have encountered before, in the translation of Sahlqvist formulas, where σ_i is BOX-AT and β is POS. Once we have a type 2 formula, we can almost proceed opposite to what we did with the Sahlqvist formulas.

To get from type 1 to type 2 we use the following intermediary step; we have that a type 1 formula $\forall^r \bar{x} \mathsf{Q}^r \bar{y}.\beta$ is equivalent to

$$\tilde{\forall} \bar{P} \tilde{\forall} \bar{Q} \forall^r \bar{x} \left(\bigwedge_{0 \leq i \leq n} ST_{x_i}(p_i \wedge \Box q_i) \rightarrow \mathsf{Q}^r \bar{y}.\beta' \right)$$

Where β' is obtained by replacing the atomic formulas in β in the following way:

$$\begin{array}{lll} u = u \longrightarrow ST_u(\top) & u \neq u \longrightarrow ST_u(\bot) & u = x_i \longrightarrow ST_u(p_i) \\ R_{ux_i} \longrightarrow ST_u(\Diamond p_i) & R_{x_i u} \longrightarrow ST_u(q_i) & \end{array}$$

There is some choice here about which substitution to use if we have $R_{x_i x_j}$ or $x_i = x_j$, and we only need to introduce p_is and q_js if they actually occur in β'.

The only thing needed to get from here to a type 2 formula is eliminating the quantifiers from $\mathsf{Q}^r\bar{y}.\beta'$. This is done by repeatedly distributing the last quantifier in the sequence over β'; if it is an existential quantifier, we first write β' in DNF and then we can use

$$\exists_{y_i \rhd z}\left(\bigvee_{k\leq K}\bigwedge_{l\leq L_k} ST_{u_{kl}}(\varphi_{kl})\right) \leftrightarrow \bigvee_{k\leq K}\left(ST_z(\Diamond \bigwedge_{\substack{l\leq L_k\\ u_{kl}=y_i}}\varphi_{kl}) \wedge \bigwedge_{\substack{l\leq L_k\\ u_{kl}\neq y_i}} ST_{u_{kl}}(\varphi_{kl})\right)$$

In the case of universal quantifiers we write β' in CNF and then use a similar equivalence where each operator is switched with its dual.

All that is left is to turn the type 2 formula into a Sahlqvist formula. The easiest approach is to first rewrite

$$\tilde{\forall}\bar{P}\tilde{\forall}\bar{Q}\forall^r\bar{x}\left(\bigwedge_{0\leq i\leq n} ST_{x_i}(\sigma_i) \to \beta\right) \quad \text{to} \quad \tilde{\forall}\bar{P}\tilde{\forall}\bar{Q}\neg\exists^r\bar{x}\left(\bigwedge_{0\leq i\leq n} ST_{x_i}(\sigma_i) \wedge \neg\beta\right)$$

Next we rewrite $\bigwedge ST_{x_i}(\sigma_i) \wedge \neg\beta$ to a DNF, and distribute the existential quantifiers over it as we did before. We end up with $\tilde{\forall}\bar{P}\tilde{\forall}\bar{Q}\neg ST_{x_0}(\bigvee \alpha_k)$. Each α_k is a Sahlqvist antecedent (consisting of BOX-AT parts from the σ_is and NEG parts from $\neg\beta$), and thus so is the conjunction of them. Finally we remove the negation, so as to get a the second-order translation of a Sahlqvist formula, $\tilde{\forall}\bar{P}\tilde{\forall}\bar{Q}ST_{x_0}(\bigvee \alpha_k \to \bot)$, from which, at last, we read off the desired Sahlqvist formula: $\bigvee \alpha_k \to \bot$.

5.3 Pure formulas

In the previous two sections we have seen how to obtain a Kracht formula from a Sahlqvist formula, and, vice versa, how to obtain a Sahlqvist formula from a Kracht formula. In this section the spotlight will be on the corresponding fragment of pure hybrid formulas.

In light of the previous sections this problem has become somewhat trivial. We can branch off from the Sahlqvist-Kracht transformation cycle (see figure 5.1) at the point where we have type 1 formulas and define a (reversible) translation to hybrid formulas.

The translation from type 1 formulas to pure formulas can be done as follows:

$T(\forall_{y\rhd^k x}\varphi) = @_x\Box^k \downarrow_y T(\varphi)$
$T(\exists_{y\rhd^k x}\varphi) = @_x\Diamond^k \downarrow_y T(\varphi)$
$T(\varphi \oplus \psi) = T(\varphi) \oplus T(\psi)$ for $\oplus \in \{\wedge, \vee, \to, \leftrightarrow\}$
$T(x = y) = @_x y$
$T(R_{xy}) = @_x \Diamond y$
$T(u = u) = \top$
$T(u \neq u) = \bot$

It should be obvious that there is a straightforward reverse translation as well.

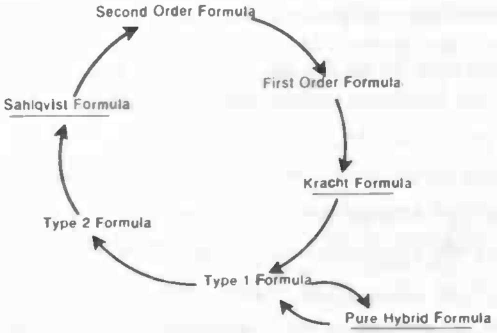


Figure 5.1, a schematic representation of the translation steps between Sahlqvist, Kracht and bound pure (hybrid) formulas.

Because of this one-to-one translation we can very easily characterize a fragment of hybrid logic that corresponds to Sahlqvist formulas. We simply take the translation of type 1 formulas; and so our hybrid Sahlqvist fragment looks like $@_{x_1^r}\Box\downarrow_{x_1}\ldots @_{x_n^r}\Box\downarrow_{x_n}\ @_{y_1^r}\Box\!\!\!\!\Diamond\downarrow_{y_1}\ldots @_{y_m^r}\Box\!\!\!\!\Diamond\downarrow_{y_m}\ \varphi$ where $\Box\!\!\!\!\Diamond$ is either $\Diamond$ or $\Box$, and each x_i^r is an x_j with $j < i$ and each y_i^r is an x_k or some y_j with $j < i$, and finally φ is a DNF of $\bot$, $\top$, $@_x y$, $@_x\Diamond y$, $@_y\Diamond x$.

Just as there were in the case of Sahlqvist and Kracht formulas, there are formulas outside this fragment that are equivalent to one in the fragment. For example, we can consider the equivalences

$@_a\Box\downarrow_b @_c\Box\downarrow_d \psi \leftrightarrow @_c\Box\downarrow_d @_a\Box\downarrow_b \psi$ (with $b \neq c$)

and

$@_a\Box\!\!\!\!\Diamond^\alpha\downarrow_b @_b\Box\!\!\!\!\Diamond^\beta\downarrow_c \psi \leftrightarrow @_a\Box\!\!\!\!\Diamond^\alpha\Box\!\!\!\!\Diamond^\beta\downarrow_c \psi$ (provided b does not occur in ψ),

where in both cases ψ is built from formulas of the form $\bot$, $\top$ and $@_u\chi$ using any connective. Aside from those equivalences, there is also a rewriting rule that bears mentioning,

$\Gamma \vdash @_a\Box\downarrow_b @_a\phi \Leftrightarrow \Gamma \vdash @_a\Diamond b \rightarrow @_a\phi$.

So, for example, if we have a formula in this hybrid Sahlqvist fragment, we can pull out every '$@_{x_0}\Box^k\downarrow_{x_i}$' to the front as a '$\Diamond^k x_i \rightarrow$', which is a useful way to eliminate '$\downarrow$'s.

5.4 Example

We will now apply the methods from the previous three sections to an example. The example we will use is that of a modal formula characterizing weakly connected frames (in which any two worlds accessible from a given world are either identical, connected or both). The common axiom we find for weak connectedness is $\Box(\varphi \wedge \Box\varphi \rightarrow \psi) \vee \Box(\psi \wedge \Box\psi \rightarrow \varphi)$; the first thing we may notice is that this is not in fact a Sahlqvist formula. Fortunately, it is equivalent to one, and a little rewriting yields the equivalent Sahlqvist formula $\Diamond(p \wedge \Box p \wedge q) \rightarrow \Box(q \vee \Diamond q \vee p)$.

Our first goal will be to derive the first-order frame condition which corresponds to the Sahlqvist axiom. We start with

$$\Diamond(p \wedge \Box p \wedge q) \rightarrow \Box(q \vee \Diamond q \vee p)$$

And translate this to the corresponding second-order formula

$$\forall P \forall Q\, (\exists_{y \rhd x}[P(y) \wedge \forall_{w \rhd y} P(w) \wedge Q(y)] \rightarrow \forall_{z \rhd x}[Q(z) \vee \exists_{v \rhd z} Q(v) \vee P(z)])$$

Then we pull out the existential quantifier(s) from the antecedent

$$\forall P \forall Q \forall_{y \rhd x}\, ([P(y) \wedge \forall_{w \rhd y} P(w) \wedge Q(y)] \rightarrow \forall_{z \rhd x}[Q(z) \vee \exists_{v \rhd z} Q(v) \vee P(z)])$$

With only boxed atoms left in the antecedent, we now first determine the minimal instantiations for the predicates, $\sigma(P) = \lambda u.(u = y \vee R_{yu})$ and $\sigma(Q) = \lambda u.(u = y)$, and then apply these substitutions.

$$\forall P \forall Q \forall_{y \rhd x} \forall_{z \rhd x}(z = y \vee \exists_{v \rhd z}(v = y) \vee z = y \vee R_{yz})$$

Which finally simplifies to

$$\forall y \rhd x \forall_{z \rhd x}(z = y \vee R_{zy} \vee R_{yz})$$

Next we translate this result to a hybrid formula.
As it happens $\forall_{y \rhd x} \forall_{z \rhd x}(z = y \vee R_{zy} \vee R_{yz})$ is already a type 1 formula so all that is needed is applying the translation. We get

$@_x \Box \downarrow_y @_x \Box \downarrow_z (@_z y \vee @_z \Diamond y \vee @_y \Diamond z)$, or $\Diamond y \wedge \Diamond z \rightarrow (@_z y \vee @_z \Diamond y \vee @_y \Diamond z)$, which is equivalent

The translation back to a Sahlqvist formula takes a bit more work, even though we are fortunate enough that $\forall_{y \rhd x} \forall_{z \rhd x}(z = y \vee R_{zy} \vee R_{yz})$, our starting formula, is already a type 1 formula. As y and z are both inherently universal we have a choice as how to translate R_{yz} and R_{zy}.
If we use $z = y \longrightarrow ST_z(p)$, $R_{yz} \longrightarrow ST_z(q)$ and $R_{zy} \longrightarrow ST_z(\Diamond p)$ we get

$$\tilde{\forall} P \tilde{\forall} Q\ \forall_{y \rhd x} \forall_{z \rhd x}(ST_y(p \wedge \Box q) \rightarrow ST_z(p) \vee ST_z(\Diamond p) \vee ST_z(q))$$

Next, we can simplify the consequent.

$$\tilde{\forall} P \tilde{\forall} Q\ \forall_{y \rhd x} \forall_{z \rhd x}(ST_y(p \wedge \Box q) \rightarrow ST_z(p \vee \Diamond p \vee q))$$

Now we bring in the quantifiers.

$$\tilde{\forall} P \tilde{\forall} Q\ \forall y \rhd x(ST_y(p \wedge \Box q) \rightarrow \forall z \rhd x ST_z(p \vee \Diamond p \vee q))$$
$$\tilde{\forall} P \tilde{\forall} Q\ \exists_{y \rhd x} ST_y(p \wedge \Box q) \rightarrow \forall_{z \rhd x} ST_z(p \vee \Diamond p \vee q)$$

And then we do the reverse translation for the quantifiers,

$$\tilde{\forall} P \tilde{\forall} Q\ ST_x \Diamond(p \wedge \Box q) \rightarrow ST_x \Box(p \vee \Diamond p \vee q)$$

Simplify,

$$\tilde{\forall} P \tilde{\forall} Q\ ST_x[\Diamond(p \wedge \Box q) \rightarrow \Box(p \vee \Diamond p \vee q)]$$

And finally, we go from second-order formula to modal, and we get

$$\Diamond(p \wedge \Box q) \rightarrow \Box(p \vee \Diamond p \vee q)$$

Note that this is not the Sahlqvist formula we started off with. When we translate a Sahlqvist formula to a Kracht formula and back, we are only guaranteed to get an equivalent Sahlqvist formula (and the same is true if we go from Kracht to Sahlqvist and back).

5.5 Concluding remarks

Especially in the case of automated theorem proving there are advantages to working with only hybrid axioms; the ability to translate a large class of modal axioms to hybrid form is therefore a valuable one. Moreover, as the process can be automated, it is possible to integrate it into a prover system. Unfortunately, it does not apply to all modal axioms, but only to those in Sahlqvist form.

References

[5] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*, chapter 3.6 Sahlqvist formulas, 3.7 More about Sahlqvist formulas, pages 156–178. Cambridge University Press, 2002.
website: `http://www.mlbook.org/`

[10] L.A. Chagrova. An undecidable problem in correspondence theory. *Journal of Symbolic Logic*, 56:1261–1272, 1991.

[11] Barteld Kooi, Gerard Renardel de Lavalette, and Rineke Verbrugge. Hybrid logics with infinitary proof systems. *Journal of Logic Computation*, 16(2):161–175, 2006.
website: `http://logcom.oxfordjournals.org/cgi/content/abstract/16/2/161-a`
doc: `http://logcom.oxfordjournals.org/cgi/reprint/16/2/161-a.pdf`

[12] Marcus Kracht. *Diamonds and Defaults*, volume 229 of *Synthese library*, chapter How completeness and correspondence got married, pages 175–214. Kluwer Academic Publishers, 1993.

[13] Edward John Lemmon and Dana S. Scott. *The 'Lemmon notes' : An Introduction to Modal Logic*. Oxford : Blackwell, 1977.

[16] Henrik Sahlqvist. Completeness and correspondence in the first and second order semantics for modal logic. In Stig Kanger, editor, *Proceedings of the Third Scandinavian Logic Symposium*, volume 82 of *Studies in Logic and the Foundations of Mathematics*, pages 110–143, Amsterdam, Oxford, 1975. North Holland Publishing Co. [etc.].

[18] Balder ten Cate, Maarten Marx, and Petrúcio Viana. Hybrid logic with Sahlqvist axioms. *Logic Journal of IGPL*, 13(3):293–300, 2005.
website: `http://jigpal.oxfordjournals.org/cgi/content/abstract/13/3/293`
doc: `http://jigpal.oxfordjournals.org/cgi/reprint/13/3/293.pdf`

Part III

Etiquette: Table Arrangements for Hybrid Logic

Contents

6 A short introduction to semantic tableaux

One of the most popular proof procedures used for automated theorem proving in modal logics is the method of semantic tableaux. The method aims to prove theorems by refutation: given a formula it will engage in a systematic search to find a counter-model that would satisfy the negation of the formula. If an exhaustive search does not yield a counter-model this is taken to mean there is not one, and therefore the theorem must be true. However, this conclusion depends on the rules of the tableau forming a sound and complete proof system (i.e. *only* valid theorems can be proven, and *all* valid theorems can be proven).

A tableau calculus typically consists of a set of rules that break down formulas into their constituent parts. This means that each node in the tableau proof tree will contain a set that consists of subformulas of previous nodes. In other words tableaux usually conform to the subformula property to some extent, which greatly helps in limiting the search space.

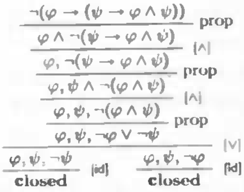
When we want to prove a theorem, we start the proof tree with the singleton set containing the negation of the theorem. So if we want to show φ holds we have $\{\neg\varphi\}$ as our root. From there on we continually apply the tableau rules until there is no longer any rule that applies (with a preference for applying the closing rule, to prevent unnecessary work). Then we examine whether all branches of the tableau are closed; if this is not the case then we have a counter-model.

6.1 An example: propositional logic

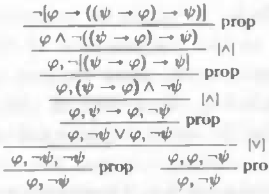
To get a feel for the tableau method, we will first have a look at how the tableau method might be applied to propositional logic. The set of tableau rules we need can be quite short. Taking into account that we can either normalize a formula at the start, or use propositional rewriting rules as we go (to eliminate $\rightarrow$ and $\leftrightarrow$, and distribute $\neg$ over connectives), the following set of three rules suffices:

$$\boxed{\frac{\Gamma, \varphi \wedge \psi}{\Gamma, \varphi, \psi}\,[\wedge] \qquad \frac{\Gamma, \varphi \vee \psi}{\Gamma, \varphi \mid \Gamma, \psi}\,[\vee] \qquad \frac{\Gamma, \varphi, \neg\varphi}{\textbf{closed}}\,[\mathrm{id}]}$$

These rules strictly satisfy the subformula property, so we can now examine any propositional formula and determine whether it is a tautology or not in a finite number of steps. Each tableau rule either reduces the number of connectives ($\vee$, $\wedge$), of which there are finitely many, or it closes a branch. So any path from the root to a leaf will have at most as many tableau rules applied as there are connectives. Nevertheless, in the worst case the total number of nodes can still be exponential in the depth of the tree, due to branching.


$\neg(\varphi \to (\psi \to \varphi \wedge \psi))$
prop
$\varphi \wedge \neg(\psi \to \varphi \wedge \psi)$
$[\wedge]$
$\varphi, \neg(\psi \to \varphi \wedge \psi)$
prop
$\varphi, \psi \wedge \neg(\varphi \wedge \psi)$
$[\wedge]$
$\varphi, \psi, \neg(\varphi \wedge \psi)$
prop
$\varphi, \psi, \neg\varphi \vee \neg\psi$
$[\vee]$
$\varphi, \psi, \neg\psi$
[id]
closed
$\varphi, \psi, \neg\varphi$
[id]
closed

Example 6.1
A closing tableau for the theorem $\varphi \to (\psi \to \varphi \wedge \psi)$. There is no counter-model, therefore the formula is a tautology.


$\neg[\varphi \to ((\psi \to \varphi) \to \psi)]$
prop
$\varphi \wedge \neg((\psi \to \varphi) \to \psi)$
$[\wedge]$
$\varphi, \neg[(\psi \to \varphi) \to \psi]$
prop
$\varphi, (\psi \to \varphi) \wedge \neg\psi$
$[\wedge]$
$\varphi, \psi \to \varphi, \neg\psi$
prop
$\varphi, \neg\psi \vee \varphi, \neg\psi$
$[\vee]$
$\varphi, \neg\psi, \neg\psi$
prop
$\varphi, \neg\psi$
$\varphi, \varphi, \neg\psi$
prop
$\varphi, \neg\psi$

Example 6.2
An open tableau for the formula $\varphi \to ((\psi \to \varphi) \to \psi)$. A counterexample is formed when φ and $\neg\psi$ both hold.

In example 6.1 we can see the tableau method at work. The use of propositional rewriting is abbreviated with prop. To prove $\varphi \to (\psi \to \varphi \wedge \psi)$ we start by taking the negation, $\neg(\varphi \to (\psi \to \varphi \wedge \psi))$, and then rewrite it to change the main connective to one our rules can deal with. This gives us $\varphi \wedge \neg(\psi \to \varphi \wedge \psi)$, which we can split using rule $[\wedge]$ into $\{\varphi, \neg(\psi \to \varphi \wedge \psi)\}$. To apply the next tableau rule, we once again need to rewrite one of the formulas in the set, we then get $\{\varphi, \psi \wedge \neg(\varphi \wedge \psi)\}$. Now we can use $[\wedge]$ again, and split $\psi \wedge \neg(\varphi \wedge \psi)$, this yields $\{\varphi, \psi, \neg(\varphi \wedge \psi)\}$. A last rewrite results in $\{\varphi, \psi, \neg\varphi \vee \neg\psi\}$; and using $[\vee]$ the tableau branches, giving us two sets, $\{\varphi, \psi, \neg\varphi\}$ and $\{\varphi, \psi, \neg\psi\}$, both of which are closed by [id].

Example 6.2, which examines the formula $\varphi \to ((\psi \to \varphi) \to \psi)$, follows much the same road, except in the end we are left with the set $\{\varphi, \neg\psi\}$ in both branches. Neither of the formulas in the set can be broken down any further, so it must constitute a counterexample. And indeed, applying the assignments $V(\varphi) = \top$ and $V(\psi) = \bot$ in our starting formula we get $\top \to ((\bot \to \top) \to \bot)$ which simplifies to $\bot$, which confirms that we have found a counter-model.

6.2 Tableaux and sequent calculus

Up to now we have had the tableau method operate on a single set of formulas, but we can also split this into two sets, a positive set (of formulas we want to satisfy) and a negative set (of formulas we try not to satisfy). When there is an overlap between these sets it means there is a formula we do and do not want satisfied at the same time, which of course cannot be realized, and therefore no counter-model can be constructed. So this will be the new closing condition for branches in the tableau.

In essence the change is quite superficial, but it has a few advantages. One of these is that we can now deal with negative formulas in a neat way; we can remove a negation by transferring the formula to the other set. But the main advantage is notational: there is now a strong connection with sequent calculus[17]. We can interpret the nodes of a tableau, and the proof

[17] A result so well-known, I cannot find a reference where it is not assumed well-known.

as a whole, in terms of sequents rather than loose sets of formulas. If we take Γ as the conjunction of the positive set and Δ as the disjunction of the negative set, then we can think of $\Gamma \vdash \Delta$ as a sequent[18]. If Γ holds, then Δ holds. And reading the tableau proof upside down, we have a valid deductive proof in sequent calculus.

The change has a number of consequences for our rule set. First, because we have two sets to work with, we generally have separate rules for both kinds of sets, 'left' rules and 'right' rules.

$$\frac{\Gamma, \neg\varphi \vdash \Delta}{\Gamma \vdash \varphi, \Delta}\,[\mathrm{L}\neg] \qquad \frac{\Gamma \vdash \neg\varphi, \Delta}{\Gamma, \varphi \vdash \Delta}\,[\mathrm{R}\neg]$$

$$\frac{\Gamma, \varphi \vee \psi \vdash \Delta}{\Gamma, \varphi \vdash \Delta \mid \Gamma, \psi \vdash \Delta}\,[\mathrm{L}\vee] \qquad \frac{\Gamma \vdash \varphi \vee \psi, \Delta}{\Gamma \vdash \varphi, \psi, \Delta}\,[\mathrm{R}\vee]$$

$$\frac{\Gamma, \varphi \wedge \psi \vdash \Delta}{\Gamma, \varphi, \psi \vdash \Delta}\,[\mathrm{L}\wedge] \qquad \frac{\Gamma \vdash \varphi \wedge \psi, \Delta}{\Gamma \vdash \varphi, \Delta \mid \Gamma \vdash \psi, \Delta}\,[\mathrm{R}\wedge]$$

$$\frac{\Gamma, \varphi \vdash \Delta, \varphi}{\textbf{closed}}\,[\mathrm{id}]$$

We have also gained rules to handle negation, again in a 'right' and 'left' variety. And we have a different rule for closing a branch, which looks at both sets.

In the next chapter we will extend the tableau method in this second representation to hybrid logic, and introduce rules for handling modal and hybrid operators.

[18] Note that this deviates from the interpretation in the first part of the paper! There $\Gamma \vdash \Delta$ was read as $\Gamma \vdash \varphi$ for all $\varphi \in \Delta$, i.e. a conjunction.

7 Tableaux for hybrid logic

After the small introduction of the previous chapter, we now go to work on the actual tableau system for our hybrid logic. It will be largely based on the tableau system in [6], but there will be some differences. Continuing the line of thought of previous chapter, we will use the sequent tableau representation. Moreover, a number of rules will be altered to make their automated use more convenient.

7.1 Propositional rules

Starting where we left off in the previous chapter, we first make two minor adaptions to the rule set that deals with propositional operators. We add two pairs of rules to break up formulas with → and ↔ as main operator, such that we do not implicitly need to use rewriting rules to handle them. And in addition each formula in the tableau will be grounded in a named world, which is achieved by prepending the starting formulas with $@_i$ (where i is a fresh nominal).

Propositional rules

$$\frac{\Gamma, \neg @_s\varphi \vdash \Delta}{\Gamma \vdash @_s\varphi, \Delta}\;[\mathrm{L}\neg] \qquad \frac{\Gamma \vdash \neg @_s\varphi, \Delta}{\Gamma, @_s\varphi \vdash \Delta}\;[\mathrm{R}\neg]$$

$$\frac{\Gamma, @_s(\varphi \vee \psi) \vdash \Delta}{\Gamma, @_s\varphi \vdash \Delta \mid \Gamma, @_s\psi \vdash \Delta}\;[\mathrm{L}\vee] \qquad \frac{\Gamma \vdash @_s(\varphi \vee \psi), \Delta}{\Gamma \vdash @_s\varphi, @_s\psi, \Delta}\;[\mathrm{R}\vee]$$

$$\frac{\Gamma, @_s(\varphi \wedge \psi) \vdash \Delta}{\Gamma, @_s\varphi, @_s\psi \vdash \Delta}\;[\mathrm{L}\wedge] \qquad \frac{\Gamma \vdash @_s(\varphi \wedge \psi), \Delta}{\Gamma \vdash @_s\varphi, \Delta \mid \Gamma \vdash @_s\psi, \Delta}\;[\mathrm{R}\wedge]$$

$$\frac{\Gamma, @_s(\varphi \rightarrow \psi) \vdash \Delta}{\Gamma \vdash @_s\varphi, \Delta \mid \Gamma, @_s\psi \vdash \Delta}\;[\mathrm{L}\rightarrow] \qquad \frac{\Gamma \vdash @_s(\varphi \rightarrow \psi), \Delta}{\Gamma, @_s\varphi \vdash @_s\psi, \Delta}\;[\mathrm{R}\rightarrow]$$

$$\frac{\Gamma, @_s(\varphi \leftrightarrow \psi) \vdash \Delta}{\Gamma, @_s\varphi, @_s\psi \vdash \Delta \mid \Gamma \vdash @_s\varphi, @_s\psi, \Delta}\;[\mathrm{L}\leftrightarrow] \qquad \frac{\Gamma \vdash @_s(\varphi \leftrightarrow \psi), \Delta}{\Gamma, @_s\varphi \vdash @_s\psi, \Delta \mid \Gamma, @_s\psi \vdash @_s\varphi, \Delta}\;[\mathrm{R}\leftrightarrow]$$

$$\frac{\Gamma, @_s\varphi \vdash \Delta, @_s\varphi}{\textbf{closed}}\;[\mathrm{id}]$$

7.2 Modal rules

Extending the tableau system to hybrid logic requires the introduction of new kinds of rules to deal with its various aspects. We start with the modal operators, ◇ and □.

The basic method here is to break down formulas to ones of the form $@_ij$, $@_ip$ and $@_i\Diamond j$ by giving (individually) accessible worlds names. If we have a ◇ on the left side (or equivalently a □ on the right side) this is quite simple: give the world it leads to a (new) name, splitting $@_i\Diamond\varphi$ into

$@_i\Diamond j$ and $@_j\varphi$. Because $@_i\Diamond\varphi$ can be reconstructed from $@_i\Diamond j$ and $@_j\varphi$ we can replace it and avoid redundancy. We will see later on how these rules correspond with the theory from the first part of the paper.

If we have a $\Box$ on the left side (or equivalently a $\Diamond$ on the right side) it is more difficult because we do not know how many worlds there are (if any). So in this case we take the set of all worlds we do know are accessible, and if we have $@_i\Box\varphi$ then for each j such that $@_i\Diamond j$ we have $@_j\varphi$. However, it is always possible that we may later find a new $@_i\Diamond k$, so we cannot throw away $@_i\Box\varphi$ without (possibly) losing information.

Modal rules

$$\frac{\Gamma, @_s\Diamond\varphi \vdash \Delta}{\Gamma, @_s\Diamond t, @_t\varphi \vdash \Delta}\,[\mathrm{L}\Diamond] \qquad \frac{\Gamma, @_s\Diamond t \vdash @_s\Diamond\varphi, \Delta}{\Gamma \vdash @_t\varphi, \Delta}\,[\mathrm{R}\Diamond]$$

$$\frac{\Gamma, @_s\Box\varphi, @_s\Diamond t \vdash \Delta}{\Gamma, @_t\varphi \vdash \Delta}\,[\mathrm{L}\Box] \qquad \frac{\Gamma \vdash @_s\Box\varphi, \Delta}{\Gamma, @_s\Diamond t \vdash @_t\varphi, \Delta}\,[\mathrm{R}\Box]$$

If we read [R□] and [L◇] from bottom to top, we see these proof steps are equivalent to an application of the **Paste Rule** (note that the t introduced by the tableau rule is a fresh nominal, so the conditions for applying **PR** are met). And reading [L□] and [R◇] upside down, they turn out to be an application of **bridge**.

7.3 @ rules

Next we consider the @ operator. The general strategy here is to eliminate them as much as possible, by reducing sequences of @s to a single one, and by removing formulas of the form $@_s t$ that occur on the right side.

@ rules

$$\frac{\Gamma, @_s@_t\varphi \vdash \Delta}{\Gamma, @_t\varphi \vdash \Delta}\,[\mathrm{L\ Agree}] \qquad \frac{\Gamma \vdash @_s@_t\varphi, \Delta}{\Gamma \vdash @_t\varphi, \Delta}\,[\mathrm{R\ Agree}]$$

$$\frac{\Gamma, @_s s \vdash \Delta}{\Gamma \vdash \Delta}\,[\mathrm{L\ ref}] \qquad \frac{\Gamma \vdash \Delta, @_s s}{\textbf{closed}}\,[\mathrm{T}] \qquad \frac{\Gamma, @_s t \vdash \Delta}{\Gamma[t/s] \vdash \Delta[t/s]}\,[\mathrm{L\ sub}]$$

The first two rules are exactly what they seem, the tableau equivalents of **Agree**. With the next two we have a significant divergence from [6]. First we have a different take on [L ref]. Because we know $@_s s$ is true for any s, there is no need to explicitly keep the formula around. However, we need to complement this choice with the addition of a second closing rule: [T].

The last rule in the box '@ rules', [L sub], takes the place of two rules used in [6] ([L nom] and [L bridge]), and functions as a more general substitution rule. Again this allows us to throw out formulas without losing important information.

7.4 ↓ rules

The ↓ is the last operator we need to construct rules for. Again our goal is to try and eliminate it as much as possible.

↓ rules

$$\frac{\Gamma, @_s \downarrow_w \varphi \vdash \Delta}{\Gamma, @_s\varphi[s/w] \vdash \Delta}\text{[L Name]} \qquad \frac{\Gamma \vdash @_s \downarrow_w \varphi, \Delta}{\Gamma \vdash @_s\varphi[s/w], \Delta}\text{[R Name]}$$

These rules correspond to **DA** (via **Ded** and **AIM**). The nature of these rules is somewhat deceptive. The rules suggest we can simply eliminate ↓; however, it does not work in every situation. The modal operators throw a spanner in the works. As we saw there, certain formulas cannot be broken down into subformulas. If those subformulas contain the ↓ it means we cannot eliminate them either. When this is the case we may run the risk of getting trapped in loops, trying to build infinite counter-models.

In some cases this is avoidable. If we examine e.g. $@_s\Diamond t \vdash @_s\Diamond \downarrow_k @_s\Box k$ with the tableau system, then the naive application of rules would generate more and more 'new' worlds. However, some of those can be unified: they are not necessarily new. So in this case we can avoid looping by checking whether worlds can be unified, or by using a model checker to see if the partial counter-model suffices as it is.

However, an infinite counter-model becomes unavoidable with sequents such as $@_s\Diamond\top, @_s\Box\Diamond\top, @_s\Box\Box \downarrow_t @_s\Diamond t, @_s\Box \downarrow_t \Box\Box \downarrow_u @_t\Diamond u \vdash @_s\Diamond\downarrow_t\Diamond t$, (see figure 7.1). Although, evidently, it may still be found in some cases.

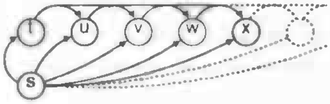


Figure 7.1, When the tableau system is used to examine the sequent $@_s\Diamond\top, @_s\Box\Diamond\top, @_s\Box\Box \downarrow_t @_s\Diamond t, @_s\Box \downarrow_t \Box\Box \downarrow_u @_t\Diamond u \vdash @_s\Diamond \downarrow_t\Diamond t$, it will proceed to generate every world in the infinite model above.

The largest problem stems from the fact that in general hybrid logic with ↓ is undecidable [1] (except under certain conditions). So we may need to try uncountably many models before ruling out there is a counter-model.

7.5 Axiom introduction

In addition to the basic hybrid logic system, we also want to be able to cope with (pure) axioms. For this we need a rule to introduce (instances) of axioms into the tableau.

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash @_s\varphi, \Delta \mid \Gamma, @_t\psi \vdash \Delta}\text{[Ax: } @_s\varphi \vdash @_t\psi\text{]}$$

At any given moment there are only a finite number of instances to pick from, because the number of nominals that are in use is finite. However, an axiom may give rise to new nominals later on, and so there is again the risk of getting stuck in a loop. Therefore it is important to choose carefully which axiom and instance thereof is used.

7.6 Infinitary terms

Infinitary sequents pose the final challenge for our system. Recall that infinitary sequents contain a term which is the conjunction of a set of countably infinitely many formulas. In this last section we will introduce two methods for handling them.

Our first strategy deals with an infinitary term on the left side of the sequent. The strategy here it to extract an appropriate instance from the set.

$$\boxed{\dfrac{@_s\{\varphi[k] \mid k \in \mathbb{N}\}, \Gamma \vdash \Delta}{@_s\varphi[a], @_s\{\varphi[k] \mid k \in \mathbb{N}\}, \Gamma \vdash \Delta \text{ \{for some } a \in \mathbb{N}\}}\,[\mathsf{Inst}]}$$

With an infinitary term on the right side, the procedure is a bit more complicated. Most likely induction will be needed, but the number of base cases and induction cases may vary. When just one base and induction case is necessary the following rule for strong induction can be used:

$$\boxed{\dfrac{\Gamma \vdash @_s\{\varphi[k] \mid k \in \mathbb{N}\}, \Delta}{\Gamma \vdash @_s\varphi[0], \Delta \;\mid\; \Gamma, @_s\{\varphi[k] \mid k < n \in \mathbb{N}\} \vdash @_s\varphi[n], \Delta \text{ \{if } n > 0\}}\,[\mathsf{Ind}]}$$

The tableau system laid out in this chapter will be the basis for the rest of the work in this thesis.

References

[1] Carlos Areces, Patrick Blackburn, and Maarten Marx. Hybrid logic: Characterization, interpolation and complexity. *Journal of Symbolic Logic*, 66(3):977–1010, 2001.
website: `http://www.loria.fr/~blackbur/pub.html`
doc: `http://www.loria.fr/~blackbur/papers/charac.pdf`

[3] Patrick Blackburn. Modal logic as dialogical logic. *Synthese*, 127:57–93, 2001.
website: `http://www.loria.fr/~blackbur/pub.html`
doc: `http://www.loria.fr/~blackbur/papers/synthese.pdf`

[6] Patrick Blackburn and Maarten Marx. Tableaux for quantified hybrid logic. In U. Egly and C. Fernmüller, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 38–52. International Conference, TABLEAUX 2002, 2002.
website: `http://www.loria.fr/~blackbur/pub.html`
doc: `http://www.loria.fr/~blackbur/papers/tableaux02.pdf`

[7] Patrick Blackburn and Johan van Benthem. Modal logic: A semantic perspective. In *Handbook of Modal Logic*. Elsevier Science, 2006.
website: `http://staff.science.uva.nl/~johan/seminar.html`
doc: `http://staff.science.uva.nl/~johan/hml-blackburnvanbenthem.pdf`

[11] Barteld Kooi, Gerard Renardel de Lavalette, and Rineke Verbrugge. Hybrid logics with infinitary proof systems. *Journal of Logic Computation*, 16(2):161–175, 2006.
website: `http://logcom.oxfordjournals.org/cgi/content/abstract/16/2/161-a`
doc: `http://logcom.oxfordjournals.org/cgi/reprint/16/2/161-a.pdf`

Act IV

The Act of Creation

Contents

8 Design

"There is no such thing as intelligent design."

In this chapter we will discuss the hopes and dreams for our theorem prover (many of which may be cruelly dashed when we arrive at the actual implementation in the following chapter). Some attention will go to the input and output, and the representation of data and rules, but the most important aspect will be the global structure of the theorem prover and the choices that have to be made there.

8.1 Design goals

Before design can start we should lay out our goals for the theorem prover and take inventory of various choices and problems we may face.

The main goal is of course to prove theorems of hybrid logic, but we can be more specific by looking at the various fragments.

The basic hybrid logic with only nominals and @ should not be much of a problem. This fragment is decidable, although the run-time of a tableau based algorithm may still be exponential in the length of the formulas. Next we have the fragment of hybrid logic which also allows $\downarrow$. This one is more problematic because it is undecidable and therefore it may take arbitrarily long to find a proof if there is one, and longer still if there is not. Nevertheless, we can give it a good shot. Lastly we allow infinitary terms. Now this is the most unsettling, because we have as many variants here as there are induction structures. Especially with respect to this last class, the theorem prover will have to be limited in scope.

The prover will be allowed three acceptable types of output. If it can find a proof it should return this in a readable format. If a counter-model is found instead, we should instead be given this. The last case relates to the intractability of the problem; if the program takes too long to decide either way, it is allowed to give up.

8.2 Data structures

The normal hybrid formulas can be represented by a simple grammar, but two more complex types warrant some extra attention. First, we have parametrized formulas where a modal operator may be repeated a fixed, but possibly unknown, number of times, e.g. formulas like $\Box^n\varphi$ and $\Diamond^{n+m+5}\varphi$. And next, related to these, are infinitary terms, where we have an enumeration of an infinite number of (parametrized) formulas, e.g. $\{@_i\Diamond^n j | n \in \mathbb{N}\}$.

The nodes of the tableau are represented with eight sets of formula: the main split is between formulas on the left and right side of the turnstile; then each side is split further into four: simple formulas that cannot be split up, infinitary terms, composite formulas which may be split up into simple formulas, and lastly 'boxed' formulas which can only be used together with the other two types and cannot be reduced.

The last class of data we need to represent are the tableau rules. The rules could be woven into the algorithm, but by keeping the rule base separate we can more easily modify the behaviour of the prover later on. Each rule should consist of a number of elements: a name, whether it operates on left or right formulas, what form of input it applies to and what the result will be and optionally we could give each rule a priority or specify a set of conditions in which it can be used (for example, we might have rules that apply for specific frames). If we give rules priorities we can also consider allowing these to be changed during the running of the algorithm.

8.3 Preprocessing

After reading the input, it needs to be preprocessed to be able to use it effectively. First, we distinguish the axioms, and the theorem we want to prove. The terms in the theorem will have to be grounded in a world[19] (because each tableau rule only applies to grounded formulas). Next we can split the terms in the theorem up in several classes; first, according to whether they occur on the left or right side of the turnstile (the '$\vdash$' in a sequent), and then, according to how the tableau rules apply to them (can they be broken down further, and if so can they be replaced by the result).

8.4 Searching for a proof

An important part of the algorithm is the search procedure employed to find a proof. At each step there are a number of ways a proof might be continued, depending on which tableau rules apply. One option is to examine each possible continuation at the same time. One benefit of this is that you can do as much work as possible in each step, without having to do things repeatedly. However, there is a large cost when rules split the node, because you will have to keep them all in memory; in the case of complex theorems you can easily go beyond the limits of memory.

Another option is iterative deepening search. As in the case of breadth-first search you will find the shallowest proof first, but without the cost in

[19] A formula is 'grounded in a world i' if it has the form $@_i\varphi$.
To ground formulas, we choose a nominal x which does not occur in the theorem and then prefix the ungrounded formulas with $@_x$. This is allowed because of the rules **SNec** (Ch. 3.1), and **VS** (App. A.1.3).

memory. Instead we have to trade off time: each deeper iteration has to redo the work of the previous iteration.

When doing an iterative deepening search it makes sense to consider prioritizing which sub-searchtree to examine first. And as this is determined by the application of a rule, one way would be to give each rule a score; this score could even be changed dynamically based on past successes. Up front we can tell that closing a branch whenever a closing rule applies should have the priority, because nothing shortens a search like stopping.

8.5 Finalizing the proof

Once we have a proof it generally is not pleasant to look at; e.g. at each step we have kept all the subformulas of previous steps, where in the end we need at most two to close a branch. We can trim the fat off the proof by tracing back from the leaves to the root, and keeping only those subformulas that are necessary, which will reduce the length of the sequents at each step of the proof.

In some cases it might be useful to take two nodes together; e.g. the end node is typically a tautology $\varphi \vdash \varphi$ and together with the next step it might be the instantiation of an axiom.

In other cases we might want to split a node, to make it explicit that weakening is used to introduce extra subformulas (which were previously trimmed away).

Finally before final printing, we can rename the nominals; the automatic generation may have created unnecessarily long, weird names, and it would be preferable to have nominals like i, j, k, s, t etc.

9 Implementation

The theorem prover is implemented as a prolog program using SWI prolog. It consists of six units, as shown in figure 9.1. The arrows in the picture represent the interdependence of units. For example, the 'prover' unit is the top-level unit which ties everything together, and the 'input' unit is required by all other units for the representation of formulas. The unit 'misc' (for miscellaneous) is a catch-all for predicates that do not fit well in any of the other units.

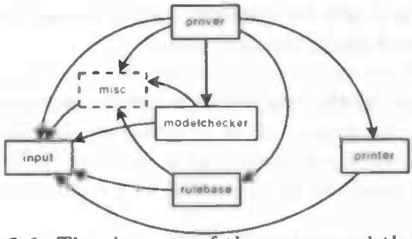


Figure 9.1, The six parts of the prover and their interdependence

The programming code for the prover can be found at the end of the appendix section of this thesis, and is available upon request by email.

9.1 Input

Not much needs to be said about the 'input' unit; mainly two things are handled here. Firstly, this unit defines a number of unary prefix operators and binary infix operators to allow a more natural input of sequents that one wishes to prove. Aside from the normal prolog convention to use terms like `implies(a, b)` and `not(a)`, the definition of prefix and infix operators allows the more natural notation of `a => b`[20] and `~ a`.
Following is a list of acceptable notations using operators:

formula	**representation(s)**
$\neg a$	`not a, ~ a`
$\Diamond a$	`dia a`
$\Box a$	`box a`
$a \vee b$	`a or b, a \/ b`
$a \wedge b$	`a and b, a /\ b`
$a \rightarrow b$	`a implies b, a => b`
$a \leftrightarrow b$	`a equals b, a <=> b`
$@_i a$	`@ i : a, at i : a`
$\downarrow_i a$	`! i : a, bind i : a, name i : a, downarrow i : b`

The hybrid operators pose a bit of a problem here, because the way they are normally used cannot be well replicated in SWI prolog. To fix this we

[20] '->' might have been preferred as operator but unfortunately it is an operator used by prolog itself and redefining it causes problems in some cases.

use a second operator, ':', to tie the hybrid operator+nominal to the rest of the formula.

Three further types of formulas cannot be represented with operators, and can only be written as regular prolog terms:

formula	representation(s)
$\Diamond^8 a$	`dia(a, [8])`
$\Box^{n+k+m} a$	`box(a, [k,m,n])`
$\{a \vert n,k \in \mathbb{N}\}$	`forall(a, [k,n])`, `set(a, [k,n])`

It should be noted that the order in which the parameter list is given does not really matter, as it will be sorted in preprocessing (any numbers will be summed together and made the first element).

The second part of the 'input' unit defines predicates to govern the standardization of the input. This is necessary because as seen above a number of alternative input notations is allowed for some formulas, and therefore the input needs to be converted to a common internal format.

9.2 Printing

When we have found a proof, we next want a way to print it in a legible manner; to accomplish this we make use of the typesetting system LaTeX. The bussproofs package is used to allow the creation of sequent-style proof trees. The proof tree from the tableau prover is converted step-by-step into LaTeX commands and written to a temporary file. Once this translation is finished the temporary file is compiled and a pdf viewer is called to show the result.

9.3 Model checking

The model checker incorporated in the system is very rudimentary and only serves to pick out the most common counter-models. It works by taking the formulas of the form $@_i\Diamond j$ from the left side of the node and using them to construct a candidate model, built from the perspective of a root world (which is the world used to ground formulas in the theorem). It then proceeds to test whether the other formulas from the left side are satisfied in this candidate model (at the root world), and whether the formulas from the right side fail to be satisfied. If these conditions are met we have a counter-model.

The modelchecker is limited in a number of ways. Firstly, it does not ascribe a valuation of propositional variables to the different worlds, so it fails when anything but pure formulas is involved. Consequently frame-checker might be a more appropriate name.

A second limitation lies in its assumptions of its input. This input is presumed to come from the tableau prover. For example, formulas like $@_i j$ are expected to have been dealt with by the tableau; which allows the checker to assume all worlds are uniquely named.

9.4 Rulebase

The rulebase contains the rules for the tableau system. The head of each rule consists of three parts, a signature, an argument and a result. The signature gives the name of the rule, as well as information which will be used by the prover to select the argument. The argument contains one or two formulas, selected from the tableau node to which the rule is to be applied. The formulas from the argument are then used to determine the result of the rule, either solely via pattern matching in the head or by the optional body of the rule. Once the result of the rule is known it can be used to make the next step in the tableau.

To give a better idea of the rulebase we will discuss a few examples. We start with one of the simplest rules from the rulebase:

```
rule( sig(r, 'T-@', s),
      arg(@ S : S ),
      res(close)
      ).
```

The signature part of the rule tells us that if we want to apply the 'T-@' rule, then we need to find an argument among the set of simple formulas on the right side of the node. The argument part tells us that the argument needs to be of the form $@_s s$. And lastly, the result part tells us that if we have succeeded in finding the argument, then we can close the current branch of the tableau (because obviously $@_i i$ can never be false).

The most important difference in our second example, compared to the first, is the result of the rule. Often the branch of a tableau will not be closed but will grow in some way.

```
rule( sig(l, 'L imply', c),
      arg( @ S : Phi implies Psi ),
      res( split(add([], [@ S : Phi]),
                 add([@ S : Psi], [])
                 ) )
      ).
```

When the 'left imply' rule applies the branch of the tableau splits in two, and the prover will need to complete both branches if it is to find a proof.

The last example shows two important differences with the previous examples. First, we have two formulas in the argument part of the rule. Secondly, there is a significant body to the rule, relating the argument to the result.

```
rule( sig(l, 'L box-n', sb),
      arg( @ S : box(Phi,P) , @ S : dia(nom(T),[L])),
      res( add([@ nom(T) : Res], []))
      ):-
      (number(L) -> selectnum(L, P, R) ; select(L,P,R)),
      ((R == []) -> (Phi \= nom(T), Res = Phi) ;
                    (Res = box(Phi, R))).
```

This rule deals with formulas that have a repeated modal operator. If we have $@_s\Diamond^l t$ and $@_s\Box^p\varphi$, where $p \geq l$ then we can add $@_t\Box^{p-l}\varphi$ to the node of the tableau. However, l might be either a number or a parameter and p can be the sum of a number and several parameters; so some work is needed to construct the right result.

After the declaration of the entire rulebase, the signatures of the rules are collected to allow easy selection of a rule. The prover can then pick the rules one at a time in the order in which they were declared, choosing the arguments based on the rule that was selected rather than the other way around. This way simple or non-branching rules can be given precedence over more complicated ones.

9.5 Prover

The engine of the theorem prover is contained in the 'prover' unit, and ties the other units together.

We query the prover by providing a sequent to the predicate `start`; the prover then proceeds by standardizing the notation of the sequent, and prepares the sequent further by grounding each term to a world. The final step of preprocessing entails sorting out the terms into their various bins.

After preprocessing we start the real work: an iterative deepening search for a proof or counter-model. We run processing cycles of increasing depths, starting at a depth of one, until we either get a result or reach the maximum depth (which defaults to 15, but can be changed by the user). In this way we should get the shallowest, most concise proof, if there is one.

The processing cycle falls into several steps again: First, we check whether we have reached the maximum depth, in which case the prover will let the user know its limit has been reached and stop. If we have not gone beyond the maximum depth, then the next step is for the modelchecker to try and distill a counter-model from the current tableau node. If we do not find a counter-model then we will need to continue work on building the tableau.

To make the next step in the tableau we first pick a rule by selecting a rule signature. Based on the signature we try to select the required arguments; if they can be replaced they are also extracted from the node. Once we have the arguments we apply the rule to determine how to make the next step. If the rule calls for closing the current branch we do so; otherwise we will need to update the tableau node in some way for the next step.

There are three types of updates a rule can call for: the usual case will be that we need to add formulas to the left and/or right side of the node. The second case calls for splitting the tableau into two branches; which then reduces to two different instances of the previous case. Lastly we may need to replace one nominal by another throughout the node.

Once we have the updated tableau node(s), we recursively apply the processing cycle with a decreased depth.

On return from the recursion we may find that there was a counter-model down the road, or that we reached the maximum depth. In those cases we simply pass on that result. We may also find that we successfully closed the branch(es) beneath the current node. When this happens we proceed by cleaning up the current node, ridding it of formulas that were not used further down, and then finish the current proof-subtree to pass on up.

After the iterative deepening search has finished, all that is left is to show the result. Proofs are displayed by generating a pdf; and a counter-model, or notice of reaching the maximum depth, is shown in plain text on the terminal.

10 Results and evaluation

In this chapter we will give a few examples of the prover at work. All the sequents used in these examples are taken from among the examples elsewhere in this thesis; however, in a few cases they have had to be adapted slightly, because unfortunately the prover still lacks proper functionality for handling axioms.

First, we start the program, by starting up the prolog interpreter, and loading the `hyloprover` unit (which then loads the other units).

```
Welcome to SWI-Prolog (Multi-threaded, Version 5.6.35)
Copyright (c) 1990-2007 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?- [hyloprover].
%  input.pl compiled 0.00 sec, 8,996 bytes
%  print.pl compiled 0.00 sec, 12,152 bytes
%  misc.pl compiled 0.00 sec, 10,956 bytes
%  rulebase.pl compiled 0.01 sec, 23,404 bytes
%  modelchecker.pl compiled 0.00 sec, 10,644 bytes
% hyloprover compiled 0.01 sec, 82,696 bytes

Yes
?-
```

10.1 Example 1, CD: $@_i\Diamond^m k, @_j\Diamond^n k \vdash @_i\Diamond^m \downarrow_s @_j\Diamond^n s$

Our first example will deal with the **Common Descendant** theorem, for which a hand-made proof is given in appendix A.2.3. To get the prover working on this theorem we query it with the command:

```
start([@ nom(i):dia(nom(k), [m]), @ nom(j) :dia(nom(k), [n])] ::-
       @ nom(i):dia(!  nom(s):(@ nom(j):dia(nom(s),[n])),[m])).
```

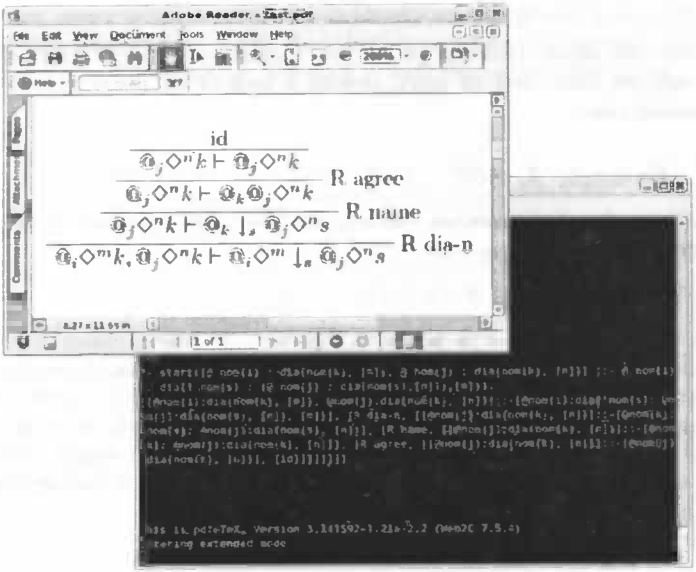
Almost immediately the prover will have found a proof, and calls the pdfviewer to display it, as shown in the screenshot in figure 10.1 on the next page. Most notable of this example is that the prover works on a generalized sequent: the theorem holds for any m and n.

10.2 Example 2, `reflexivity` $\Rightarrow AS3 : \{\neg @_i\Diamond^n i | n \in \mathbb{N}^+\} \vdash \bot$

Our second example comes from section 4.4.1, and is an infinitary theorem. Our goal is to prove

`reflexivity` $\Rightarrow AS3 : \{\neg @_i\Diamond^n i | n \in \mathbb{N}^+\} \vdash \bot$

We face an important problem here: we cannot use axioms. So we need to adapt the theorem slightly. Instead of using an axiom for reflexivity, we take only a single specific instance of a reflexive axiom, and add this to the



$$\dfrac{\dfrac{\dfrac{\dfrac{\text{id}}{@_j\Diamond^n k \vdash @_j\Diamond^n k}}{@_j\Diamond^n k \vdash @_k @_j\Diamond^n k}\;\text{R agree}}{@_j\Diamond^n k \vdash @_k \downarrow_s @_j\Diamond^n s}\;\text{R name}}{@_i\Diamond^m k, @_j\Diamond^n k \vdash @_i\Diamond^m \downarrow_s @_j\Diamond^n s}\;\text{R dia-n}$$

```
?- start([@ nom(i) : dia(nom(k), [m]), @ nom(j) : dia(nom(k), [n])] ::- @ nom(i)
 : dia(! nom(s) : (@ nom(j) : dia(nom(s),[n])),[m])).
[[@nom(i):dia(nom(k), [m]), @nom(j):dia(nom(k), [n])]::-[@nom(i):dia(!nom(s): @n
om(j):dia(nom(s), [n]), [m])], [R dia-n, [[@nom(j):dia(nom(k), [n])]::-[@nom(k):
!nom(s): @nom(j):dia(nom(s), [n])], [R name, [[@nom(j):dia(nom(k), [n])]::-[@nom
(k): @nom(j):dia(nom(k), [n])], [R agree, [[@nom(j):dia(nom(k), [n])]::-[@nom(j)
:dia(nom(k), [n])], [id]]]]]]]



This is pdfeTeX, Version 3.141592-1.21a-2.2 (Web2C 7.5.4)
entering extended mode
```

Figure 10.1, screenshot of the prover after finding the proof in the first example.

premise of the sequent. Now we can query our prover with:

```
start([@nom(i):dia nom(i),
      forall(@nom(i):not dia(nom(i), [n, 1]), [n])]::-false).
```

In the terminal window we will get the following output:

```
?- start([@ nom(i):dia nom(i),
         forall(@ nom(i):not dia(nom(i), [n,1]), [n])] ::- false).
[[@nom(•):forall(@nom(i):not dia(nom(i), [1, n]), [n]), @nom(i):dia(nom
(i), [1])]::-[], [L inf-agree, [[@nom(i):dia(nom(i), [1]), @nom(i):fora
ll(not dia(nom(i), [1, n]), [n])]::-[], [L inf-dual, [[@nom(i):dia(nom(
i), [1]), @nom(i):forall(box(not nom(i), [1, n]), [n])]::-[], [L inf-b,
 [[@nom(i):box(not nom(i), [1]), @nom(i):dia(nom(i), [1])]::-[], [L box
-n, [[@nom(i):not nom(i)]::-[], [L not, [[]::-[@nom(i):nom(i)], [T-@]]]
]]]]]]]]
```

And in our pdfviewer this proof is displayed in a much more humane manner as:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{T-@}}{\vdash @_i i}}{@_i \neg i \vdash \bot}\;\text{L not}}{@_i\Box\neg i, @_i\Diamond i \vdash \bot}\;\text{L box-n}}{@_i\Diamond i, @_i\{\Box^{1+n}\neg i | n \in N\} \vdash \bot}\;\text{L inf-b}}{@_i\Diamond i, @_i\{\neg\Diamond^{1+n} i | n\in N\}\vdash\bot}\;\text{L inf-dual}}{@.\{@_i\neg\Diamond^{1+n} i | n\in N\}, @_i\Diamond i\vdash\bot}\;\text{L inf-agree}$$

This example shows that our prover can indeed prove infinitary theorems. Although we may need to work around a lack of support for axioms to accomplish this.

10.3 Example 3, $AS2^+ : \{\neg @_i \Diamond^k j | k \in \mathbb{N} \wedge k \geq 1\} \vdash \bot \ \Rightarrow \vdash \Diamond\top$

The third example is another theorem that involves infinitary terms, it is the theorem from section 4.4.2.

$$AS2^+ : \{\neg @_i \Diamond^k j | k \in \mathbb{N} \wedge k \geq 1\} \vdash \bot \ \Rightarrow \vdash \Diamond\top$$

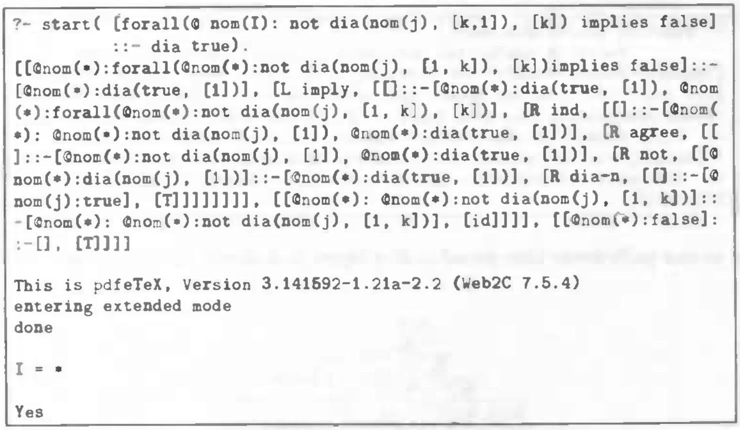
Again we have an axiom to deal with, but we solve the problem differently this time. Instead of taking specific instances of the axiom, we can leave nominals uninstantiated by making use of prolog variables. In the prover process these variable nominals can then be instantiated when they are unified with another nominal. However, such an 'axiom' would still need to be added for every different instance of it which is used throughout the proof.

Our query to the prover is given as follows:

```
start([forall(@ nom(I):not dia(nom(j), [k,1]), [k]) implies false]
      ::-dia true).
```

As can be seen, we left one nominal from the axiom uninitialized, giving it the variable I. The reason we did not leave j uninitialized is purely to get a better looking proof: j is printed in the proof and if it had been a variable we would have gotten something like G_143 in its place.

On finding the proof, prolog lets us know which instantiations where used for any variables (in this case I). The terminal output is shown below:



```
?- start( [forall(@ nom(I): not dia(nom(j), [k,1]), [k]) implies false]
          ::- dia true).
[[@nom(*):forall(@nom(*):not dia(nom(j), [1, k]), [k])implies false]::-
[@nom(*):dia(true, [1])], [L imply, [[]::-[@nom(*):dia(true, [1]), @nom
(*):forall(@nom(*):not dia(nom(j), [1, k]), [k])], [R ind, [[]::-[@nom(
*): @nom(*):not dia(nom(j), [1]), @nom(*):dia(true, [1])], [R agree, [[
]::-[@nom(*):not dia(nom(j), [1]), @nom(*):dia(true, [1])], [R not, [[@
nom(*):dia(nom(j), [1])]::-[@nom(*):dia(true, [1])], [R dia-n, [[]::-[@
nom(j):true], [T]]]]]]]], [[@nom(*): @nom(*):not dia(nom(j), [1, k])]::
-[@nom(*): @nom(*):not dia(nom(j), [1, k])], [id]]]], [[@nom(*):false]:
:-[], [T]]]]

This is pdfeTeX, Version 3.141592-1.21a-2.2 (Web2C 7.5.4)
entering extended mode
done

I = *

Yes
```

In our pretty-printed proof we can clearly see our prover using induction to eliminate an infinitary term. And there is no sign of the variable nominal, as it has been unified with $*$ (the root world) throughout the proof.

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{\top}{\vdash @_j\top}
}{@_*\Diamond j \vdash @_*\Diamond\top}\;\text{R dia-n}
}{\vdash @_*\neg\Diamond j \vee @_*\Diamond\top}\;\text{R not}
}{\vdash @_*@_*\neg\Diamond j \vee @_*\Diamond\top}\;\text{R agree}
\qquad
\dfrac{\text{id}}{@_*@_*\neg\Diamond^{1+k}j \vdash @_*@_*\neg\Diamond^{1+k}j}
}{\vdash @_*\Diamond\top \vee @_*\{@_*\neg\Diamond^{1+k}j|k \in N\}}\;\text{R ind}
\qquad
\dfrac{\top}{@_*\bot \vdash \bot}
}{@_*(\{@_*\neg\Diamond^{1+k}j|k \in N\} \rightarrow \bot) \vdash @_*\Diamond\top}\;\text{L imply}
$$

10.4 Example 4, a counter-model for $@_s\Diamond t \vdash @_s\Diamond \downarrow_k @_s\Box k$

Our last example in this chapter comes from section 7.4, where we used $@_s\Diamond t \vdash @_s\Diamond \downarrow_k @_s\Box k$ as an example of problems a pure tableau prover might face. Without any countermeasures, a tableau could keep applying rules ad infinitum. For this reason we have included a small model checker to our tableau system.

We set our prover to work on $@_s\Diamond t \vdash @_s\Diamond \downarrow_k @_s\Box k$ using the following query

```
start(@nom(s):dia nom(t)::-@nom(s):dia(!nom(k):@nom(s):box nom(k))).
```

After a few moments, the prover displays its findings in the terminal: there is a counter-model.

```
?- start(@nom(s):dia nom(t)::- @nom(s):dia(!nom(k): @nom(s):box nom(k))).
Counter-model:
[[[[[world(nom(s)), world(nom(nC)), world(nom(t)), link(nom(s), nom(nC),
[1]), link(nom(s), nom(t), [1])]]]]]
```

The output is not pretty, but it is clear enough. If we have worlds s, c and t, and the only links are from s to c and from s to t, then the sequent cannot be true: the premise is true, but the consequent is not.

10.5 Final remarks

As the examples in this chapter show, our prover comes a long way to meet the goals we have set. It can prove theorems and display them in a format that is comfortable to read. It can also find counter-models. On down side, a regrettable deficiency is the inability to deal with axioms in a proper way; although, as we have seen, there are workarounds in some cases.

References

[2] Carlos Areces and Juan Heguiabehere. HyloRes: Direct resolution for hybrid logics. In C. Areces and M. de Rijke, editors, *Proceedings of Methods for Modalities 2*, Amsterdam, The Netherlands, November 2001.
website: `http://www.loria.fr/~areces/content/papers/sort_date.php`
doc: `http://www.loria.fr/~areces/content/papers/files/m4m02.pdf`

[6] Patrick Blackburn and Maarten Marx. Tableaux for quantified hybrid logic. In U. Egly and C. Fernmüller, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 38–52. International Conference, TABLEAUX 2002, 2002.
website: `http://www.loria.fr/~blackbur/pub.html`
doc: `http://www.loria.fr/~blackbur/papers/tableaux02.pdf`

[8] Robert S. Boyer and J. Strother Moore. *A Computational Logic*. ACM monograph series, ISSN 0572-4252. New York [etc.] : Academic Press, 1979.

[9] Ricardo Caferra, Alexander Leitsch, and Nicolas Peltier. *Automated Model Building*, chapter Automated Model Building (Ch.1 Introduction), page 334. Applied Logic Series , Vol. 31. Kluwer, 2004.
website: `http://www.logic.at/people/leitsch/`
doc: `http://www.logic.at/people/leitsch/mbbook.pdf`

[14] Lawrence C. Paulson. Designing a theorem prover. In Abramsky & Gabbay & Maibaum (Eds.), editor, *Handbook of Logic in Computer Science*, volume 2, pages 415–475. Clarendon, 1992.
website: `https://www.publications.cl.cam.ac.uk/116/`
doc: `https://www.publications.cl.cam.ac.uk/116/01/TR192-lcp-designing.pdf`

Part V
Reflections

Contents

11 Discussion

In this thesis we have developed, to our knowledge, the first theorem prover for infinitary hybrid logic. It should be mentioned, however, that there are provers for other flavours of hybrid logic (e.g. HyloRes [2]), but none of them deal with infinitary hybrid logic.

Rather than adapting an existing prover, we have built a new one from the ground up. One of the reasons for this was simply the problem of getting access to existing provers and their source code. Another reason lies in that our goal is to provide a human-readable proof, and not, as is often the case, just a yes or no answer as to whether a sequent is a theorem.

11.1 The Proof Method

Our first real challenge was to find a way to adapt the theoretical proof system based on [11] in such a way that we could use it to prove theorems. A direct approach is problematic, because bottom-up there is no telling which axioms, or how many, we need to combine to reach our candidate theorem; and top-down we have to deal with the cut rule, which gives us an infinite number of choices to proceed.

To avoid these problems we have had to adapt the infinitary proof system to one of several possible proof methods. An important reason for choosing a tableaux method based on [6] comes, again, from our goal of providing intelligible proofs: as discussed in section 6.2, a tableau proof can be easily transformed to a proof in sequent calculus. So by employing the tableau method we can remain close to the original proof system. If on the other hand we had chosen a different proof method which was further removed from sequent calculus, e.g. resolution [2], then a lot more effort would be needed to transform the result into a readable deductive proof.

However, there are also disadvantages to the approach we have taken. The tableau method is not the most efficient proving method, and by focusing on creating a readable proof we have had to trade in even more efficiency: we have dozens of rules for specific situations (and still have not exhausted all options), where we might have done with fewer if we had only needed to answer yes or no.

11.2 The Implementation Language

The language we chose to implement our prover in is Prolog. One reason for this is obvious: Prolog is a logical programming language, and we are dealing with a logic project, so it is a natural first choice. A more important reason is that a logical language generally allows smaller programs that are easier to read and understand [21], compared to imperative programming

[21] Functional languages like Haskell also share the same advantage of allowing small programs, but unfamiliarity with Haskell prevented it from being a viable option.

languages like C/C++. Related to this is the fact that Prolog enables us to do a number of things without much effort, e.g.:

- We can provide a half-decent notation for hybrid logic formulas by defining operators.
- Rules can be simply defined as predicates and still be used as a separate rulebase.
- Matching formulas can be done via prolog unification
- Backtracking is automatic.

In other areas Prolog is not as congenial, however.

For one, Prolog is relatively slow, but this can easily be forgiven; speed is not the main purpose, and its advantages outweigh this small disadvantage. More problematic is another handicap, which rears its head when we have to deal with complex data-structures.

In the normal style of Prolog programming data has to be passed on in each step. When there are a lot of different pieces of data, we can either use predicates with many arguments, which does little for intelligibility, or we can try to group the data in some structure. But once we have built a complex data structure, we find that accessing it becomes a burden. In our prover this is especially apparent with the structure of the tableau nodes and the predicate `selectarg` which accesses and 'modifies'[22] it.

Data management has proven to be a real headache in Prolog, in no small part detracting from the readability we desire from a Prolog program. In an imperative programming language like C/C++ this would be much less of a problem, because we could have direct access to parts of data and hide away processes in objects; but in other areas, like matching formulas, things would be much more complicated again. Overall it is hard to say which would be preferable without having tried the alternatives.

[22] There is no real modification of data, instead one piece of data is put in relation to a piece of data which can be considered a modified copy.

12 Further work

We have made a long and arduous journey on a path from theory to implementation, but the path continues ever onwards. This thesis may be nearly at its end, but there is always further work to be done in projects such as these. In this final chapter we explore the road which lies ahead for anyone willing to venture there.

12.1 Axioms

As we noted in chapter 10, one of the shortcomings of our theorem prover is that it lacks a mechanism for dealing with axioms. Although we can work around this in some cases, it would be much more convenient if such a mechanism were included.

The main problem with axioms is finding the right initialization. Using variable nominals[23] that are instantiated once they are matched to actual nominals poses the problem that we have to clean up uninitialized nominals at the end. On the other hand, initializing axioms entirely the moment they are invoked means we may have to try every combination of nominals that occur in the current tableau branch.

12.2 Infinitary Terms

The current theorem prover only covers a limited range of infinitary terms. An obvious improvement to the system would be to extend this range. For example, one could think of generalizing the induction rule. It should be possible to use induction structures to define formulas, and then use those same structures in a general induction rule to decide which base cases are needed and which induction steps.

We could also think about better heuristics for extracting instances from infinite sets. At the moment we have a lot of ad hoc rules for combining infinitary terms with normal formulas; cleaning this up would be a marked improvement to the intelligibility of the program.

12.3 Model checking

Although the current model checker embedded in the system does not, in general, contribute a lot to the results, it may be worth exploring the combination further. Combining two methods in this way may help cover blind spots in either: A tableau system tries to exclude the possibility there is a counter-model, but might not necessarily notice when there is one. A model checker (or rather model builder [9]) approaches the problem from the other direction — it tries to give a counter-model but might fail to notice there cannot be one.

[23] Which could be implemented with prolog variables, as we did in chapter 10, or by some other means over which we have more direct control.

And at the very least, it would seem fitting to have a hybrid prover for a hybrid logic.

The model checker that is currently integrated in the prover system is very minimalistic, and so there are a number of avenues open for further development. Firstly, there is the issue of propositional variables. The checker only builds a frame; it ignores the valuation function (and consequently cannot give a conclusive answer when propositional variables are involved). By extending the model checker to take propositional variables into consideration it would become a lot more versatile.

Another area for improvement lies in how the model (frame) is built. The checker only uses formulas of the type $@_i \Diamond j$, but it could also use other formulas. For example, if we have $@_i \Box \Diamond i$, then every time we would create a new link from i to another world, we should make a link in the reverse direction. In the current setup, the model checker waits for the tableau system to create $@_x \Diamond i$ for every x such that $@_i \Diamond x$.

A final option we will mention here is including a heuristic for recognizing infinite models. Some sequents only have counter-models of an infinite size, which can pose a real problem for a theorem prover because it might work diligently on a proof indefinitely (safe for time limits and similar safeguards). People on the other hand can sometimes quickly spot that there is such a counter-model; giving a model checker that same intuition would therefore be a helpful addition.

12.4 Hybrid Sahlqvist formulas

Although the chapter about hybrid logic and Sahlqvist formulas was 'on the side', so to speak, there are some interesting opportunities for future work. On the theoretical side one can consider working out the details of the translation for the extensions of Sahlqvist formulas that exist. On the practical side, it would be interesting to implement the algorithm for translating between Sahlqvist, Kracht, and hybrid formulas. And, when this has been accomplished, such an implementation could be used as part of theorem prover that internally works only with pure axioms.

References

[2] Carlos Areces and Juan Heguiabehere. HyloRes: Direct resolution for hybrid logics. In C. Areces and M. de Rijke, editors, *Proceedings of Methods for Modalities 2*, Amsterdam, The Netherlands, November 2001.
website: `http://www.loria.fr/~areces/content/papers/sort_date.php`
doc: `http://www.loria.fr/~areces/content/papers/files/m4m02.pdf`

[6] Patrick Blackburn and Maarten Marx. Tableaux for quantified hybrid logic. In U. Egly and C. Fernmüller, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 38–52. International Conference, TABLEAUX 2002, 2002.
website: `http://www.loria.fr/~blackbur/pub.html`
doc: `http://www.loria.fr/~blackbur/papers/tableaux02.pdf`

[9] Ricardo Caferra, Alexander Leitsch, and Nicolas Peltier. *Automated Model Building*, chapter Automated Model Building (Ch.1 Introduction), page 334. Applied Logic Series , Vol. 31. Kluwer, 2004.
website: `http://www.logic.at/people/leitsch/`
doc: `http://www.logic.at/people/leitsch/mbbook.pdf`

[11] Barteld Kooi, Gerard Renardel de Lavalette, and Rineke Verbrugge. Hybrid logics with infinitary proof systems. *Journal of Logic Computation*, 16(2):161–175, 2006.
website: `http://logcom.oxfordjournals.org/cgi/content/abstract/16/2/161-a`
doc: `http://logcom.oxfordjournals.org/cgi/reprint/16/2/161-a.pdf`

Appendices

Contents

A Additional hand proofs

In this appendix the additional hand proofs that were constructed during the writing of this thesis will be included. Some of the theorems, rules and equivalences between frame axioms that are proved here have already been used or mentioned in chapter 4, but many others are new.

As we did in chapter 4, we start with a number of relatively simple rules and sequents. Next we proceed to a number of generalized sequents and rules. And finally we look at a few pure formulas corresponding to modal axioms that characterize certain frame properties. (There are no further examples involving infinitary sequents.)

A.1 Derivable rules and sequents

A.1.1 DedRev (reverse deduction): $\Gamma \vdash \varphi \to \psi \Rightarrow \Gamma, \varphi \vdash \psi$

DedRev is a simple rule which offers the reverse functionality of **Ded**. It is quite useful for simplifying proofs by streamlining sequents for use with **InfCut**.

$$\dfrac{\dfrac{\text{given}}{\Gamma \vdash \varphi \to \psi} \qquad \dfrac{\text{MP}}{\varphi, \varphi \to \psi \vdash \psi}}{\Gamma, \varphi \vdash \psi}\ \text{InfCut}$$

A.1.2 NR (name rule): $\Gamma, i \vdash \varphi \;\Rightarrow\; \Gamma \vdash \varphi$ provided $i \notin \mathtt{fnom}(\Gamma, \varphi)$

The **name rule** provides the means to eliminate superfluous nominals from the premise of a sequent. When no claims are made regarding a world named by a nominal its elimination does not influence the validity of the theorem.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{given } \{i \notin \mathtt{fnom}(\Gamma, \varphi)\}}{\Gamma, i \vdash \varphi}}{@_i\Gamma, @_i i \vdash @_i\varphi}\ \text{SNec}_@ \qquad \dfrac{\text{T}_@}{\vdash @_i i}}{@_i\Gamma \vdash @_i\varphi}\ \text{InfCut}}{\downarrow_i @_i\Gamma \vdash \downarrow_i @_i\varphi}\ \text{SNec}_\downarrow}{\Gamma \vdash \varphi}\ \text{Name,Namerev}$$

A.1.3 $\text{distr}^@_\wedge$ (distribution of @ over $\wedge$): $\vdash @_i(\varphi \wedge \psi) \leftrightarrow (@_i\varphi \wedge @_i\psi)$

As is the case with every normal modal operator[24] , $@_i$ distributes over $\wedge$; nevertheless, for completeness, it is proven here. This theorem has already been used in chapter 4 for the proof of $\text{distr}^@_\vee$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{taut}}{\varphi \wedge \psi \vdash \varphi}}{@_i(\varphi \wedge \psi) \vdash @_i\varphi}\ \text{SNec}_@ \qquad \dfrac{\dfrac{\text{taut}}{\varphi \wedge \psi \vdash \psi}}{@_i(\varphi \wedge \psi) \vdash @_i\psi}\ \text{SNec}_@}{@_i(\varphi \wedge \psi) \vdash @_i\varphi \wedge @_i\psi}}{\vdash @_i(\varphi \wedge \psi) \to (@_i\varphi \wedge @_i\psi)}\ \text{Ded} \qquad \dfrac{\dfrac{\dfrac{\text{taut}}{\varphi, \psi \vdash \psi \wedge \varphi}}{@_i\varphi, @_i\psi \vdash @_i(\psi \wedge \varphi)}\ \text{SNec}_@}{\vdash (@_i\varphi \wedge @_i\psi) \to @_i(\psi \wedge \varphi)}\ \text{Ded}}{\vdash @_i(\varphi \wedge \psi) \leftrightarrow (@_i\varphi \wedge @_i\psi)}$$

[24] A normal modal operator is one for which strong necessitation holds, or, equivalently, both the distribution axiom (also called K axiom) and (weak) necessitation; e.g. □ in basic modal logic, K in epistemic logic, G and H in temporal logic and $[\pi]$ in dynamic logic, but not, in general, their duals.

A.1.4 $\text{distr}^{@}_{\to}$ (distribution of @ over $\to$): $\vdash @_i(\varphi \to \psi) \leftrightarrow (@_i\varphi \to @_i\psi)$

If we expand the definition of → the theorem **$\text{distr}^{@}_{\to}$** follows directly from **$\text{distr}^{@}_{\vee}$** (which was proven in chapter 4).

$$\dfrac{\dfrac{\dfrac{\text{distr}^{@}_{\vee}}{\vdash @_i(\neg\varphi \vee \psi) \leftrightarrow (@_i\neg\varphi \vee @_i\psi)}}{\vdash @_i(\neg\varphi \vee \psi) \leftrightarrow (\neg @_i\varphi \vee @_i\psi)}\;\text{SD}_{@}}{\vdash @_i(\varphi \to \psi) \leftrightarrow (@_i\varphi \to @_i\psi)}$$

A.1.5 $\text{distr}^{@}_{\leftrightarrow}$ (distribution of @ over $\leftrightarrow$): $\vdash @_i(\varphi \leftrightarrow \psi) \leftrightarrow (@_i\varphi \leftrightarrow @_i\psi)$

The previous two theorems together help to prove the distribution theorem for the last common connective.

$$\dfrac{\dfrac{\dfrac{\text{distr}^{@}_{\wedge}}{\vdash @_i([\varphi \to \varphi] \wedge [\varphi \to \varphi]) \leftrightarrow (@_i[\varphi \to \varphi] \wedge @_i[\varphi \to \varphi])}}{\vdash @_i([\varphi \to \varphi] \wedge [\varphi \to \varphi]) \leftrightarrow ([@_i\varphi \to @_i\varphi] \wedge [@_i\varphi \to @_i\varphi])}\;\text{distr}^{@}_{\to}}{\vdash @_i(\varphi \leftrightarrow \varphi) \leftrightarrow ([@_i\varphi \leftrightarrow @_i\varphi)}$$

Similar distribution theorems for ↓ (**$\text{distr}^{\downarrow}_{\wedge}$**, **$\text{distr}^{\downarrow}_{\vee}$**, **$\text{distr}^{\downarrow}_{\to}$** and **$\text{distr}^{\downarrow}_{\leftrightarrow}$**) can be proven along the same lines as the distribution theorems for @; simply replace the @ by ↓ through the proofs (in fact this holds for any modal operator which is a self-dual).

A.1.6 VB (vacuous binding): $\vdash \downarrow_i \varphi \leftrightarrow \varphi$ provided $i \notin \texttt{fnom}(\varphi)$

Vacuous binding is very similar to the **name rule** and follows directly from it. Again, the intuition is that attributing a name to a world is meaningless if you never use the name.

$$\dfrac{\dfrac{\text{DA}\ \{i \notin \texttt{fnom}(\varphi)\}}{i \vdash \downarrow_i \varphi \leftrightarrow \varphi}}{\vdash \downarrow_i \varphi \leftrightarrow \varphi}\;\text{NR}$$

A.1.7 $\text{VB}^{@}$ (vacuous binding after @): $\vdash @_i \downarrow_i \varphi \leftrightarrow @_i\varphi$

The theorem **vacuous binding after @** effectively says it is unnecessary to state that you will henceforth refer to a world by a certain name if that name already refers to it.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{DA}}{i \vdash \downarrow_i \varphi \leftrightarrow \varphi}}{@_i i \vdash @_i(\downarrow_i \varphi \leftrightarrow \varphi)}\;\text{SNec}_{@}}{@_i i \vdash @_i \downarrow_i \varphi \leftrightarrow @_i\varphi}\;\text{distr}^{@}_{\leftrightarrow} \qquad \dfrac{\text{T}_{@}}{\vdash @_i i}}{\vdash @_i \downarrow_i \varphi \leftrightarrow @_i\varphi}$$

A.1.8 VS (vacuous satisfaction): $\Gamma \vdash @_i\varphi \Rightarrow \Gamma \vdash \varphi$ and $@_i\Gamma \vdash \varphi \Rightarrow \Gamma \vdash \varphi$, provided $i \notin \mathtt{fnom}(\Gamma, \varphi)$

The vacuous satisfaction rule provides a shortcut in the frequent case where you want to remove an @ from part of a sequent.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{given}}{\Gamma \vdash @_i\varphi}}{\downarrow_i \Gamma \vdash \downarrow_i @_i\varphi}\;\text{SNec}_\downarrow}{\Gamma \vdash \downarrow_i @_i\varphi}\;\text{VB}}{\Gamma \vdash \varphi}\;\text{Name} \qquad\qquad \dfrac{\dfrac{\dfrac{\dfrac{\text{given}}{@_i\Gamma \vdash \varphi}}{\downarrow_i @_i\Gamma \vdash \downarrow_i \varphi}\;\text{SNec}_\downarrow}{\downarrow_i @_i\Gamma \vdash \varphi}\;\text{VB}}{\Gamma \vdash \varphi}\;\text{Name}$$

A.1.9 Nom: $\vdash (@_i j \wedge @_j\varphi) \rightarrow @_i\varphi$

Nom captures the intuition that if two nominals can be identified with each other, then if a formula is true at the world referred to be the first, then it must also be true for the world referred to by the other (because the nominals in fact refer to the same world).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Intr}}{\vdash j \wedge \neg\varphi \rightarrow @_j\neg\varphi}}{\vdash j \wedge \neg @_j\neg\varphi \rightarrow \varphi}\;\text{prop}}{\vdash j \wedge @_j\varphi \rightarrow \varphi}\;\text{SD}_@}{\vdash @_i(j \wedge @_j\varphi \rightarrow \varphi)}\;\text{SNec}_@}{\vdash @_i(j \wedge @_j\varphi) \rightarrow @_i\varphi}\;\text{Distr}^@_\rightarrow}{\vdash (@_i j \wedge @_i @_j\varphi) \rightarrow @_i\varphi}\;\text{Distr}^@_\wedge}{\vdash (@_i j \wedge @_j\varphi) \rightarrow @_i\varphi}\;\text{Agree}$$

A.1.10 SD$_\downarrow$ (self-duality of $\downarrow$): $\vdash \neg \downarrow_i \varphi \leftrightarrow \downarrow_i \neg\varphi$

As has been mentioned before $\downarrow$ is a self-dual, just like @. Now, we finally get around to proving it; validating earlier claims that depend on it.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{DA}}{i \vdash \downarrow_i \neg\varphi \leftrightarrow \neg\varphi} \quad \dfrac{\dfrac{\text{DA}}{i \vdash \downarrow_i \varphi \leftrightarrow \varphi}}{i \vdash \neg \downarrow_i \varphi \leftrightarrow \neg\varphi}}{i \vdash \neg \downarrow_i \varphi \leftrightarrow \downarrow_i \neg\varphi}}{@_i i \vdash @_i(\neg \downarrow_i \varphi \leftrightarrow \downarrow_i \neg\varphi)}\;\text{SNec}_@ \qquad \dfrac{\text{T}_@}{\vdash @_i i}}{\vdash @_i(\neg \downarrow_i \varphi \leftrightarrow \downarrow_i \neg\varphi)}\;\text{InfCut}}{\vdash \downarrow_i @_i(\neg \downarrow_i \varphi \leftrightarrow \downarrow_i \neg\varphi)}\;\text{SNec}_\downarrow}{\vdash \neg \downarrow_i \varphi \leftrightarrow \downarrow_i \neg\varphi}\;\text{Name}$$

A.1.11 DB (double bind): $\vdash \downarrow_j \downarrow_j \varphi \leftrightarrow \downarrow_j \varphi$

The theorem double bind states that giving the same name to a world twice is not better than doing it once. In hindsight this case is already covered by vacuous binding; but waste not, want not.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{DA}}{j \vdash \downarrow_j \varphi \leftrightarrow \varphi}}{\downarrow_j j \vdash \downarrow_j (\downarrow_j \varphi \leftrightarrow \varphi)} \qquad \dfrac{\text{T}_\downarrow}{\vdash \downarrow_j j}}{\vdash \downarrow_j (\downarrow_j \varphi \leftrightarrow \varphi)}\;\text{InfCut}}{\vdash \downarrow_j \downarrow_j \varphi \leftrightarrow \downarrow_j \varphi}\;\text{distr}^\downarrow_\leftrightarrow$$

A.1.12 BC (binder commutativity): $\vdash \downarrow_i \downarrow_j \varphi \leftrightarrow \downarrow_j \downarrow_i \varphi$

A more interesting theorem regarding the binder involves the case where a world is given multiple names: the order of consecutive binders does not matter.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{DA}}{i \vdash \downarrow_i \downarrow_j \varphi \leftrightarrow \downarrow_j \varphi} \quad \dfrac{\text{DA}}{j \vdash \downarrow_j \varphi \leftrightarrow \varphi}}{i,j \vdash \downarrow_i \downarrow_j \varphi \leftrightarrow \varphi} \qquad \dfrac{\dfrac{\text{DA}}{j \vdash \downarrow_j \downarrow_i \varphi \leftrightarrow \downarrow_i \varphi} \quad \dfrac{\text{DA}}{i \vdash \downarrow_i \varphi \leftrightarrow \varphi}}{i,j \vdash \downarrow_j \downarrow_i \varphi \leftrightarrow \varphi}}{i,j \vdash \downarrow_i \downarrow_j \varphi \leftrightarrow \downarrow_j \downarrow_i \varphi}}{i \vdash \downarrow_i \downarrow_j \varphi \leftrightarrow \downarrow_j \downarrow_i \varphi}\,\text{NR}}{\vdash \downarrow_i \downarrow_j \varphi \leftrightarrow \downarrow_j \downarrow_i \varphi}\,\text{NR}$$

A.1.13 BVS (bound variable substitution): $\vdash \downarrow_i \varphi[j := i] \leftrightarrow \downarrow_j \varphi[i := j]$

A useful theorem that follows directly from the nature of a binder is that you can rename a bound variable without consequences. This can be very useful to avoid possible confusion between free and bound variables (by renaming the latter).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{DA}}{i \vdash \downarrow_j \varphi \leftrightarrow \varphi[j := i]}}{\downarrow_i i \vdash \downarrow_i (\downarrow_j \varphi \leftrightarrow \varphi[j := i])} \quad \dfrac{\text{T}_\downarrow}{\vdash \downarrow_i i}}{\vdash \downarrow_i (\downarrow_j \varphi \leftrightarrow \varphi[j := i])}\,\text{InfCut}}{\vdash \downarrow_i \downarrow_j \varphi \leftrightarrow \downarrow_i \varphi[j := i]}\,\text{distr}^{\downarrow}_{\leftrightarrow} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{DA}}{j \vdash \downarrow_i \varphi \leftrightarrow \varphi[i := j]}}{\downarrow_j j \vdash \downarrow_j (\downarrow_i \varphi \leftrightarrow \varphi[i := j])} \quad \dfrac{\text{T}_\downarrow}{\vdash \downarrow_j j}}{\vdash \downarrow_j (\downarrow_i \varphi \leftrightarrow \varphi[i := j])}\,\text{InfCut}}{\vdash \downarrow_j \downarrow_i \varphi \leftrightarrow \downarrow_j \varphi[i := j]}\,\text{distr}^{\downarrow}_{\leftrightarrow}}{\vdash \downarrow_i \downarrow_j \varphi \leftrightarrow \downarrow_i \varphi[i := j]}\,\text{BC}}{\vdash \downarrow_i \varphi[j := i] \leftrightarrow \downarrow_j \varphi[i := j]}$$

A.2 Generalized rules and sequents

A.2.1 BG$^+$ (generalized bounded generalization): $\vdash @_i \Box^n \downarrow_j @_i \Diamond^n j$ provided $i \neq j$

Another nice example of a generalized sequent is generalized bounded generalization. Its proof is more complicated than the proof for Back$^+$, but simpler then Bridge$^+$. As in those cases, we rely on induction for the proof.

We start with the base case, $\vdash @_i \downarrow_j @_j i$, which follows directly from **NameRev**

base case:

$$\dfrac{\dfrac{\text{NameRev}}{\vdash i \rightarrow \downarrow_j @_j i}}{\vdash @_i \downarrow_j @_j i}\,\text{ImpAtEq}$$

For the induction step we will use the following induction hypothesis: $\forall_{0 \leq m < n} (\vdash @_i \Box^m \downarrow_j @_i \Diamond^m j)$. From **Bridge$^+$** and **Back$^+$** we can deduce that we can take a step from case $n - 1$ to n. By invoking the induction hypothesis we can then finish the proof.

induction step:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Bridge+}}{\Diamond^{n-1}k, @_k\Diamond j \vdash \Diamond^n j}}{@_i\Diamond^{n-1}k, @_i@_k\Diamond j \vdash @_i\Diamond^n j}\,\text{SNec}_{@}}{@_i\Diamond^{n-1}k, @_k\Diamond j \vdash @_i\Diamond^n j}\,\text{Agree}}{\downarrow_j @_i\Diamond^{n-1}k, \downarrow_j @_k\Diamond j \vdash \downarrow_j @_i\Diamond^n j}\,\text{SNec}_{\downarrow}}{@_i\Diamond^{n-1}k, \downarrow_j @_k\Diamond j \vdash \downarrow_j @_i\Diamond^n j}\,\text{VB}}{\Box @_i\Diamond^{n-1}k, \Box\downarrow_j @_k\Diamond j \vdash \Box\downarrow_j @_i\Diamond^n j}\,\text{SNec}_{\Box} \quad \dfrac{\text{Back}^{+}}{@_i\Diamond^{n-1}k \vdash \Box @_i\Diamond^{n-1}k}}{@_i\Diamond^{n-1}k, \Box\downarrow_j @_k\Diamond j \vdash \Box\downarrow_j @_i\Diamond^n j}\,\text{IC}}{\downarrow_k @_i\Diamond^{n-1}k, \downarrow_k\Box\downarrow_j @_k\Diamond j \vdash \downarrow_k\Box\downarrow_j @_i\Diamond^n j}\,\text{SNec}_{\downarrow} \quad \dfrac{\dfrac{\dfrac{\text{BG}}{k \vdash \Box\downarrow_j @_k\Diamond j}}{\vdash k \to \Box\downarrow_j @_k\Diamond j}\,\text{Ded}}{\vdash \downarrow_k \Box\downarrow_j @_k\Diamond j}\,\text{IDE}}{\downarrow_k @_i\Diamond^{n-1}k \vdash \downarrow_k\Box\downarrow_j @_i\Diamond^n j}\,\text{IC}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\downarrow_k @_i\Diamond^{n-1}k \vdash \downarrow_k\Box\downarrow_j @_i\Diamond^n j}{\downarrow_k @_i\Diamond^{n-1}k \vdash \Box\downarrow_j @_i\Diamond^n j}\,\text{VB}}{\Box^{n-1}\downarrow_k @_i\Diamond^{n-1}k \vdash \Box^{n-1}\Box\downarrow_j @_i\Diamond^n j}\,\text{SNec}_{\Box}^{(n-1)}}{@_i\Box^{n-1}\downarrow_k @_i\Diamond^{n-1}k \vdash @_i\Box^n\downarrow_j @_i\Diamond^n j}\,\text{SNec}_{@} \quad \dfrac{\text{induction hypothesis}}{\vdash @_i\Box^{n-1}\downarrow_k @_i\Diamond^{n-1}k}}{\vdash @_i\Box^n\downarrow_j @_i\Diamond^n j}\,\text{IC}$$

A.2.2 PR$^+$ (generalized Paste Rule): $\Gamma, @_i\Diamond^n j \vdash @_j\varphi \Leftrightarrow \Gamma \vdash @_i\Box^n\varphi$ provided $j \notin \mathtt{fnom}(\Gamma,\varphi) \cup \{i\}$

The next obvious candidate for generalization is the **Paste Rule**. Because this generalized rule is an equivalence, it means we have two directions to prove.

The right direction of the proof for the **generalized Paste Rule** follows the same lines as the normal **Paste Rule**, except that we use generalized rules in a number of places.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{given } \{j \notin \mathtt{fnom}(\Gamma,\varphi) \cup \{i\}\ \}}{\Gamma, @_i\Diamond^n j \vdash @_j\varphi}}{@_k\Gamma, @_k@_i\Diamond^n j \vdash @_k@_j\varphi}\,\text{SNec}_{@}\text{, fresh k}}{@_k\Gamma, @_i\Diamond^n j \vdash @_j\varphi}\,\text{Agree}}{\downarrow_j @_k\Gamma, \downarrow_j @_i\Diamond^n j \vdash \downarrow_j @_j\varphi}\,\text{SNec}_{\downarrow}}{@_k\Gamma, \downarrow_j @_i\Diamond^n j \vdash \varphi}\,\text{VB, Name}}{\Box^n @_k\Gamma, \Box^n\downarrow_j @_i\Diamond^n j \vdash \Box^n\varphi}\,\text{SNec}_{\Box}^{(n)} \quad \dfrac{\text{Back}^{+}}{@_k\Gamma \vdash \Box^n @_k\Gamma}}{@_k\Gamma, \Box^n\downarrow_j @_i\Diamond^n j \vdash \Box^n\varphi}\,\text{InfCut}}{@_i@_k\Gamma, @_i\Box^n\downarrow_j @_i\Diamond^n j \vdash @_i\Box^n\varphi}\,\text{SNec}_{@}}{@_k\Gamma, @_i\Box^n\downarrow_j @_i\Diamond^n j \vdash @_i\Box^n\varphi}\,\text{Agree} \quad \dfrac{\text{BG}^{+}}{\vdash @_i\Box^n\downarrow_j @_i\Diamond^n j}}{\dfrac{@_k\Gamma \vdash @_i\Box^n\varphi}{\Gamma \vdash @_i\Box^n\varphi}\,\text{VS}}\,\text{InfCut}$$

The left direction of the proof comes down to applying **generalized Bridge**.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Bridge}^{+}}{\Diamond^n j, @_j\neg\varphi \vdash \Diamond^n\neg\varphi}}{\Diamond^n j, \Box^n\varphi \vdash @_j\varphi}\,\text{cp}}{@_i\Diamond^n j, @_i\Box^n\varphi \vdash @_i@_j\varphi}\,\text{SNec}_{@}}{@_i\Diamond^n j, @_i\Box^n\varphi \vdash @_j\varphi}\,\text{Agree} \quad \dfrac{\text{given}}{\Gamma \vdash @_i\Box^n\varphi}}{\Gamma, @_i\Diamond^n j \vdash @_j\varphi}\,\text{InfCut}$$

A.2.3 CD (common descendant): $@_i\Diamond^m k, @_j\Diamond^n k \vdash @_i\Diamond^m \downarrow_s @_j\Diamond^n s$

The common descendant theorem states that if you can reach a common descendant from two worlds, in m and n steps respectively, then from the first world you can reach a descendant in m steps which you can reach from the other one in n steps.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Agree}}{\vdash @_j\Diamond^n k \to @k@_j\Diamond^n k}}{\vdash @_j\Diamond^n k \to @_k \downarrow_k @_j\Diamond^n k}\;\text{VB}^{@}}{@_j\Diamond^n k \vdash @_k \downarrow_k @_j\Diamond^n k}\;\text{DedRev} \qquad \dfrac{\dfrac{\text{Bridge}^+}{@_i\Diamond^m k, @_k \downarrow_k @_j\Diamond^n k \vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k}}{@_i\Diamond^m k, @_k \downarrow_k @_j\Diamond^n k \vdash @_i\Diamond^m \downarrow_s @_j\Diamond^n s}\;\text{BVS}}{@_i\Diamond^m k, @_j\Diamond^n k \vdash @_i\Diamond^m \downarrow_s @_j\Diamond^n s}\;\text{InfCut}$$

A.2.4 CDeq (com. desc. equality): $\vdash @_i\Diamond^m\downarrow_k @_j\Diamond^n k \leftrightarrow @_j\Diamond^n\downarrow_k @_i\Diamond^m k$

The last generalized sequent we will deal with is an equality related to the previous theorem. When two worlds share a descendant it is equivalent to say either that we can reach a world from i which can be reached from j or that we can reach a world from j which we can reach from i.

The proof is broken down into two parts due to its size and because it allows us to reuse an intermediary result. First, we prove Lemma 1: $@_j\Diamond^n \downarrow_k @_i\Diamond^m k \vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{CD}}{@_i\Diamond^m k, @_j\Diamond^n k \vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k}}{\downarrow_k @_i\Diamond^m k, \downarrow_k @_j\Diamond^n k \vdash \downarrow_k @_i\Diamond^m \downarrow_k @_j\Diamond^n k}\;\text{SNec}_{\downarrow}}{\downarrow_k @_i\Diamond^m k, \downarrow_k @_j\Diamond^n k \vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k}\;\text{VB}}{\neg @_i\Diamond^m \downarrow_k @_j\Diamond^n k, \downarrow_k @_j\Diamond^n k \vdash \neg \downarrow_k @_i\Diamond^m k}\;\text{cp}}{\Box^n \neg @_i\Diamond^m \downarrow_k @_j\Diamond^n k, \Box^n \downarrow_k @_j\Diamond^n k \vdash \Box^n \neg \downarrow_k @_i\Diamond^m k}\;\text{SNec}_{\Box}^{(n)} \qquad \dfrac{\dfrac{\text{Back}^+}{@_i\neg\Diamond^m \downarrow_k @_j\Diamond^n k \vdash \Box^n @_i\neg\Diamond^m \downarrow_k @_j\Diamond^n k}}{\neg @_i\Diamond^m \downarrow_k @_j\Diamond^n k \vdash \Box^n \neg @_i\Diamond^m \downarrow_k @_j\Diamond^n k}}{\neg @_i\Diamond^m \downarrow_k @_j\Diamond^n k, \Box^n \downarrow_k @_j\Diamond^n k \vdash \Box^n \neg \downarrow_k @_i\Diamond^m k}\;\text{IC}}{\Diamond^n \downarrow_k @_i\Diamond^m k, \Box^n \downarrow_k @_j\Diamond^n k \vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k}\;\text{cp}}{@_j\Diamond^n \downarrow_k @_i\Diamond^m k, @_j\Box^n \downarrow_k @_j\Diamond^n k \vdash @_j @_i\Diamond^m \downarrow_k @_j\Diamond^n k}}{@_j\Diamond^n \downarrow_k @_i\Diamond^m k, @_j\Box^n \downarrow_k @_j\Diamond^n k \vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k} \qquad \dfrac{\text{BG}^+}{\vdash @_j\Box^n \downarrow_k @_j\Diamond^n k}}{\text{Lemma 1: } @_j\Diamond^n \downarrow_k @_i\Diamond^m k \vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k}\;\text{IC}$$

Then using two complementary instantiations of lemma 1 we show that our theorem holds.

$$\dfrac{\dfrac{\dfrac{\text{Lemma 1}}{@_j\Diamond^n \downarrow_k @_i\Diamond^m k \vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k}}{\vdash @_j\Diamond^n \downarrow_k @_i\Diamond^m k \to @_i\Diamond^m \downarrow_k @_j\Diamond^n k}\;\text{Ded} \qquad \dfrac{\dfrac{\text{Lemma 1}}{@_i\Diamond^m \downarrow_k @_j\Diamond^n k \vdash @_j\Diamond^n \downarrow_k @_i\Diamond^m k}}{\vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k \to @_j\Diamond^n \downarrow_k @_i\Diamond^m k}\;\text{Ded}}{\vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k \leftrightarrow @_j\Diamond^n \downarrow_k @_i\Diamond^m k}\;\text{Con}$$

A.3 Frames

A.3.1 Symmetry

Symmetric frames have the property that any two worlds that are connected, are connected in both directions; this property is characterized by the modal axiom $\varphi \to \Box\Diamond\varphi$. There are a number of alternative hybrid axioms to choose from; e.g. $i \to \Box\Diamond i$, $@_i\Box\Diamond i$ and $@_j\Diamond i \to @_i\Diamond j$. The first two are equivalent by **AIM**, and $i \to \Box\Diamond i$ follows from $\varphi \to \Box\Diamond\varphi$ by instantiation with i. So for their part we are only left to prove that the modal axiom follows from either $i \to \Box\Diamond i$ or $@_i\Box\Diamond i$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Bridge \{ with fresh i \}}}{\Diamond i, \Box\neg\varphi \vdash @_i\neg\varphi}}{\Diamond i, \neg @_i\neg\varphi \vdash \neg\Box\neg\varphi}}{\Diamond i, @_i\varphi \vdash \Diamond\varphi}}{\Box\Diamond i, \Box @_i\varphi \vdash \Box\Diamond\varphi}\,\text{SNec}_\Box \qquad \dfrac{\dfrac{\dfrac{\dfrac{\text{Back \{ with fresh i \}}}{\vdash \Diamond @_i\neg\varphi \to @_i\neg\varphi}}{\vdash \neg @_i\neg\varphi \to \neg\Diamond @_i\neg\varphi}}{\vdash @_i\varphi \to \Box @_i\varphi}}{@_i\varphi \vdash \Box @_i\varphi}\,\text{DedRev}}{@_i\Box\Diamond i, @_i\varphi \vdash @_i\Box\Diamond\varphi}\,\text{InfCut} \qquad \dfrac{\text{given}}{\vdash @_i\Box\Diamond i}}{\dfrac{@_i\varphi \vdash @_i\Box\Diamond\varphi}{\dfrac{\vdash @_i \to @_i\Box\Diamond\varphi}{\dfrac{\vdash @_i(\varphi \to \Box\Diamond\varphi)}{\dfrac{\vdash \downarrow_i @_i(\varphi \to \Box\Diamond\varphi)}{\vdash \varphi \to \Box\Diamond\varphi}\,\text{Name}}\,\text{Snec}_\downarrow}\,\text{Distr}^{@}_{\to}}\,\text{Ded}}\,\text{InfCut}$$

The third hybrid axiom, $@_j\Diamond i \to @_i\Diamond j$, can be proven equivalent to $@_i\Box\Diamond i$ in a fairly straightforward way.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Bridge}}{\Diamond j, \Box\Diamond i \vdash @_j\Diamond i}}{@_i\Diamond j, @_i\Box\Diamond i \vdash @_i @_j\Diamond i}\,\text{SNec}_@}{@_i\Diamond j, @_i\Box\Diamond i \vdash @_j\Diamond i}\,\text{Agree} \qquad \dfrac{\text{given}}{\vdash @_i\Box\Diamond i}}{@_i\Diamond j \vdash @_j\Diamond i}\,\text{InfCut}}{\vdash @_i\Diamond j \to @_j\Diamond i}\,\text{Ded} \qquad\qquad \dfrac{\dfrac{\dfrac{\text{given}}{\vdash @_i\Diamond j \to @_j\Diamond i}}{@_i\Diamond j \vdash @_j\Diamond i}\,\text{DedRev}}{\vdash @_i\Box\Diamond i}\,\text{PR}$$

A.3.2 Serial

Seriality means there is always a next world. A common modal axiom used to characterize it is $\Box\varphi \to \Diamond\varphi$, but a simpler one is $\Diamond\top$. $\Diamond\top$ is a pure axiom as well as a regular modal axiom; another pure axiom for seriality is $\Box i \to \Diamond i$. First we will prove $\Diamond\top$ follows from $\Box\varphi \to \Diamond\varphi$, then the reverse; which together means they are equivalent. Finally it is shown $\Box i \to \Diamond i$ leads to $\Box\varphi \to \Diamond\varphi$ (and the reverse follows by instantiating φ with i), making all these axioms equivalent.

$$\dfrac{\dfrac{\dfrac{\text{given}}{\vdash \Box\top \to \Diamond\top}}{\Box\top \vdash \Diamond\top}\,\text{DedRev} \qquad \dfrac{\dfrac{\text{Taut}}{\vdash \top}}{\vdash \Box\top}\,\text{SNec}_\Box}{\vdash \Diamond\top}\,\text{InfCut} \qquad\qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{given}}{\vdash \Diamond\top}}{\vdash \Diamond(\neg\varphi \vee \varphi)}}{\vdash \Diamond\neg\varphi \vee \Diamond\varphi}}{\vdash \neg\Diamond\neg\varphi \to \Diamond\varphi}}{\vdash \Box\varphi \to \Diamond\varphi} \qquad\qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{given}}{\vdash \Box i \to \Diamond i}}{\vdash \neg\Box i \vee \Diamond i}}{\vdash \Diamond\neg i \vee \Diamond i}}{\vdash \Diamond(\neg i \vee i)}}{\vdash \Diamond\top}$$

A.3.3 Transitive

In transitive frames every world reachable in two (or more) steps can also be reached in one step. Transitivity is characterized by the modal axiom $\Box\varphi \to \Box\Box\varphi$, and the hybrid candidates we consider are $\Diamond\Diamond i \to \Diamond i$ and $\Diamond i \wedge @_i\Diamond j \to \Diamond j$. Like numerous times before, the first axiom follows from an instantiation of the modal axioms (in this case with $\neg i$), and we are only left to show the reverse.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Bridge } \{j \notin \text{fnom}(\varphi)\}}{\Diamond j, \Box\varphi \vdash @_j\varphi} \qquad \dfrac{\text{given}}{\Diamond\Diamond j \vdash \Diamond j}}{\Diamond\Diamond j, \Box\varphi \vdash @_j\varphi}\,\text{InfCut}}{@_i\Diamond\Diamond j, @_i\Box\varphi \vdash @_i@_j\varphi}\,\text{SNec}_{@}\text{, fresh i}}{@_i\Diamond\Diamond j, @_i\Box\varphi \vdash @_j\varphi}\,\text{Agree}}{@_i\Box\varphi \vdash @_i\Box\Box\varphi}\,\text{PR}^{+}}{\vdash @_i\Box\varphi \to @_i\Box\Box\varphi}\,\text{Ded}}{\vdash @_i(\Box\varphi \to \Box\Box\varphi)}\,\text{Distr}^{@}_{\to}}{\vdash \Box\varphi \to \Box\Box\varphi}\,\text{VS}$$

In line with previous examples, we continue with demonstrating that the second pure axiom, $\vdash \Diamond i \wedge @_i\Diamond j \to \Diamond j$, is equivalent to the previous one.

$$\dfrac{\dfrac{\dfrac{\text{Bridge}}{\Diamond i, @_i\Diamond j \vdash \Diamond\Diamond j} \qquad \dfrac{\dfrac{\text{given}}{\vdash \Diamond\Diamond j \to \Diamond j}}{\Diamond\Diamond j \vdash \Diamond j}\,\text{DedRev}}{\Diamond i, @_i\Diamond j \vdash \Diamond j}\,\text{InfCut}}{\vdash \Diamond i \wedge @_i\Diamond j \to \Diamond j}\,\text{Ded}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Bridge}}{\Diamond j, \Box i \vdash @_j i} \qquad \dfrac{\dfrac{\text{given}}{\vdash \Diamond k \wedge @_k\Diamond j \to \Diamond j}}{\Diamond k, @_k\Diamond j \vdash \Diamond j}}{\Diamond k, @_k\Diamond j, \Box i \vdash @_j i}\,\text{InfCut}}{\Diamond k, \Box i \vdash @_k\Box i}\,\text{PR}}{@_x\Diamond k, @_x\Box i \vdash @_x@_k\Box i}\,\text{SNec}_{@}}{@_x\Diamond k, @_x\Box i \vdash @_k\Box i}\,\text{Agree}}{@_x\Box i \vdash @_x\Box\Box i}\,\text{PR}}{\vdash @_x\Box i \to @_x\Box\Box i}\,\text{Ded}}{\vdash @_x(\Box i \to \Box\Box i)}\,\text{Distr}^{@}_{\to}}{\vdash \Box i \to \Box\Box i}\,\text{VS}$$

A.3.4 Euclidean

When we have a frame in which every two worlds reachable from a given world are connected then we call it euclidean. Exhaustively following the same procedure we have performed so many times before, we note the modal axiom $\Diamond\varphi \to \Box\Diamond\varphi$ for euclideanness and consider its equivalence to the pure axioms $\Diamond i \to \Box\Diamond i$ and $\Diamond i \wedge \Diamond j \to @_i\Diamond j$, starting with the former.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Bridge } \{i \notin \text{fnom}(\varphi)\}}{\Box\Diamond i, @_i\varphi \vdash \Box\Diamond\varphi} \qquad \dfrac{\dfrac{\text{given}}{\vdash \Diamond i \to \Box\Diamond i}}{\Diamond i \vdash \Box\Diamond i}\,\text{DedRev}}{\Diamond i, @_i\varphi \vdash \Box\Diamond\varphi}\,\text{InfCut}}{\Diamond i, \neg\Box\Diamond\varphi \vdash \neg @_i\varphi}\,\text{cp}}{\Diamond i, \neg\Box\Diamond\varphi \vdash @_i\neg\varphi}\,\text{SD}_{@}}{@_k\Diamond i, @_k\neg\Box\Diamond\varphi \vdash @_k@_i\neg\varphi}\,\text{SNec}_{@}\text{, fresh k}}{@_k\Diamond i, @_k\neg\Box\Diamond\varphi \vdash @_i\neg\varphi}\,\text{Agree}}{@_k\neg\Box\Diamond\varphi \vdash @_k\Box\neg\varphi}\,\text{PR}}{\vdash @_k\neg\Box\Diamond\varphi \to @_k\Box\neg\varphi}\,\text{Ded}}{\vdash @_k(\neg\Box\Diamond\varphi \to \Box\neg\varphi)}\,\text{Distr}^{@}_{\to}}{\vdash \neg\Box\Diamond\varphi \to \Box\neg\varphi}\,\text{VS}}{\vdash \Diamond\varphi \to \Box\Diamond\varphi}\,\text{cp}$$

And finally, again, we show the two pure axioms are equivalent.

$$\dfrac{\dfrac{\dfrac{\text{Bridge}}{\Diamond i, \Box\Diamond j \vdash @_i\Diamond j} \qquad \dfrac{\dfrac{\text{given}}{\vdash \Diamond j \to \Box\Diamond j}}{\Diamond j \vdash \Box\Diamond j}\,\text{DedRev}}{\Diamond i, \Diamond j \vdash @_i\Diamond j}\,\text{InfCut}}{\vdash \Diamond i \wedge \Diamond j \to @_i\Diamond j}\,\text{Ded}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{given}}{\vdash \Diamond i \wedge \Diamond j \to @_j\Diamond i}}{\Diamond i, \Diamond j \vdash @_j\Diamond i}\,\text{DedRev}}{@_k\Diamond i, @_k\Diamond j \vdash @_k@_j\Diamond i}\,\text{SNec}_{@}\text{, fresh k}}{@_k\Diamond i, @_k\Diamond j \vdash @_j\Diamond i}\,\text{Agree}}{@_k\Diamond i, \vdash @_k\Box\Diamond i}\,\text{PR}}{\vdash @_k\Diamond i \to @_k\Box\Diamond i}\,\text{Ded}}{\vdash @_k(\Diamond i \to \Box\Diamond i)}\,\text{Distr}^{@}_{\to}}{\vdash \Diamond i \to \Box\Diamond i}\,\text{VS}$$

A.3.5 Functional

Functionality is nothing more than the combination of seriality and determinism; modally it is characterized by $\Diamond\varphi \leftrightarrow \Box\varphi$, and a pure axiom for it is $\Diamond i \leftrightarrow \Box i$. The proof from modal to pure again follows from instantiation, and in the other direction it follows from combining the results of seriality and determinism.

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{given}}{\vdash \Diamond i \leftrightarrow \Box i}}{\vdash \Diamond i \rightarrow \Box i}}{\vdash \Diamond\varphi \rightarrow \Box\varphi}\,\{\text{determinism}\} \qquad \dfrac{\dfrac{\dfrac{\text{given}}{\vdash \Diamond i \leftrightarrow \Box i}}{\vdash \Box i \rightarrow \Diamond i}}{\vdash \Box\varphi \rightarrow \Diamond\varphi}\,\{\text{serial}\}}{\vdash \Diamond\varphi \leftrightarrow \Box\varphi}$$

A.3.6 Weakly dense

The weakly dense frameclass forms the second to last class we treat here. A frame is weakly dense when every world reachable in one step can also be reached in two; so e.g. reflexivity implies weak density.

The modal axiom characterizing this frame class is $\Box\Box\varphi \rightarrow \Box\varphi$. The only (simple) pure axioms that presents itself is $\Diamond i \rightarrow \Diamond\Diamond i$, which follows from instantiation with $\neg i$. This leaves only the reverse direction to be proven

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Bridge}^{+}}{\Diamond\Diamond i, @_i\neg\varphi \vdash \Diamond\Diamond\neg\varphi}}{\Diamond\Diamond i, \Box\Box\varphi \vdash @_i\varphi}\,\text{cp} \qquad \dfrac{\dfrac{\text{given}}{\vdash \Diamond i \rightarrow \Diamond\Diamond i}}{\Diamond i \vdash \Diamond\Diamond i}\,\text{DedRev}}{\Diamond i, \Box\Box\varphi \vdash @_i\varphi}\,\text{InfCut}}{@_k\Diamond i, @_k\Box\Box\varphi \vdash @_k @_i\varphi}\,\text{SNec}_{@}\text{, fresk k}}{@_k\Diamond i, @_k\Box\Box\varphi \vdash @_i\varphi}\,\text{Agree}}{@_k\Box\Box\varphi \vdash @_k\Box\varphi}\,\text{PR}}{\vdash @_k\Box\Box\varphi \rightarrow @_k\Box\varphi}\,\text{Ded}}{\vdash @_k(\Box\Box\varphi \rightarrow \Box\varphi)}\,\text{Distr}^{@}_{\rightarrow}}{\vdash \Box\Box\varphi \rightarrow \Box\varphi}\,\text{VS}$$

A.3.7 Weakly connected

The final class of frames we will deal with consists of the weakly connected frames; in these frames any two worlds reachable from a third must be either connected or they must be the same world.

Recall that this frameclass already briefly came up in the example of chapter 5, where we used the method developed in that chapter to show, among other things, that $\Box(\varphi \wedge \Box\varphi \rightarrow \psi) \vee \Box(\psi \wedge \Box\psi \rightarrow \varphi)$ is equivalent to $\Diamond i \wedge \Diamond j \rightarrow (@_i\Diamond j \vee @_i j \vee @_j\Diamond i)$. Here we will show this equivalence again, but without the detour via first-order and second-order logic. Another pure axiom we consider, is $\Box(j \rightarrow [i \vee \Diamond i]) \vee \Box(i \rightarrow [j \vee \Diamond j])$.

We start by proving the two pure axioms equivalent. The first part of this proof consists of deducing $\Diamond i \wedge \Diamond j \rightarrow (@_i\Diamond j \vee @_i j \vee @_j\Diamond i)$ under assumption of $\Box(j \rightarrow [i \vee \Diamond i]) \vee \Box(i \rightarrow [j \vee \Diamond j])$.

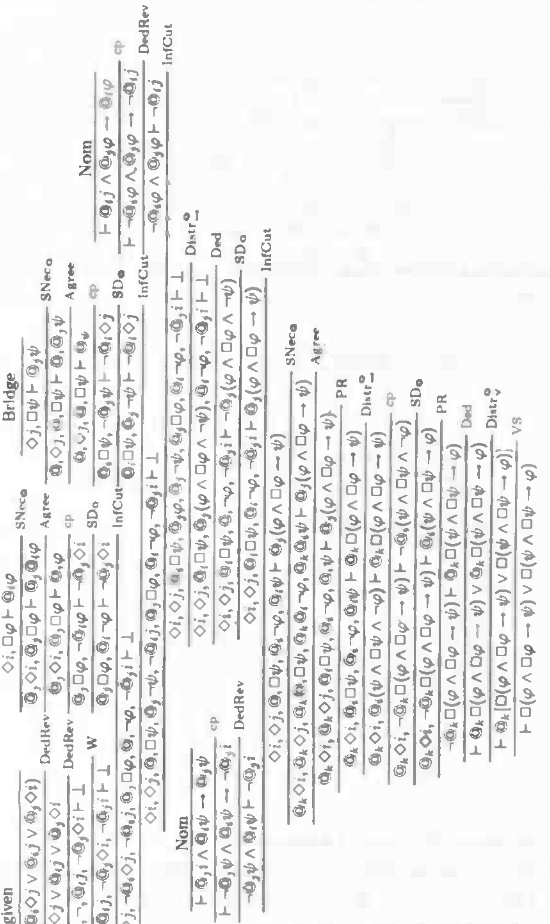
$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{T}_{@}}{\vdash @_j j} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{T}_{@}}{\vdash @_i i} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{given}}{\vdash \Box(j \to [i \lor \Diamond i]) \lor \Box(i \to [j \lor \Diamond j])}}{\neg\Box(j \to [i \lor \Diamond i]) \vdash \Box(i \to [j \lor \Diamond j])}\,\text{DedRev}}{\Diamond\neg(j \to [i \lor \Diamond i]) \vdash \Box(i \to [j \lor \Diamond j])} \quad \dfrac{\text{Bridge}}{\Diamond i, \Box(i \to [j \lor \Diamond j]) \vdash @_i(i \to [j \lor \Diamond j])}}{\Diamond i, \Diamond\neg(j \to [i \lor \Diamond i]) \vdash @_i(i \to [j \lor \Diamond j])}\,\text{IC}}{\Diamond i, \Diamond\neg(j \to [i \lor \Diamond i]) \vdash @_i i \to @_i[j \lor \Diamond j])}\,\text{Distr}^{@}_{\to}}{\Diamond i, \Diamond\neg(j \to [i \lor \Diamond i]), @_i i \vdash @_i[j \lor \Diamond j])}\,\text{DedRev}}{\Diamond i, \Diamond\neg(j \to [i \lor \Diamond i]) \vdash @_i[j \lor \Diamond j])}\,\text{IC} \quad \dfrac{\text{Bridge+}}{\Diamond j, @_j\neg(j \to [i \lor \Diamond i]) \vdash \Diamond\neg(j \to [i \lor \Diamond i])}}{\Diamond i, \Diamond j, @_j\neg(j \to [i \lor \Diamond i]) \vdash @_i[j \lor \Diamond j]}\,\text{IC}}{\Diamond i, \Diamond j, \neg @_i[j \lor \Diamond j] \vdash \neg @_j\neg(j \to [i \lor \Diamond i])}\,\text{cp}}{\Diamond i, \Diamond j, \neg @_i[j \lor \Diamond j] \vdash @_j(j \to [i \lor \Diamond i])}\,\text{SD}_{@}}{\Diamond i, \Diamond j, \neg @_i[j \lor \Diamond j] \vdash @_j j \to @_j[i \lor \Diamond i]}\,\text{Distr}^{@}_{\to}}{\Diamond i, \Diamond j, @_j j, \neg @_i[j \lor \Diamond j] \vdash @_j[i \lor \Diamond i]}\,\text{DedRev}}{\Diamond i, \Diamond j, \neg @_i[j \lor \Diamond j] \vdash @_j[i \lor \Diamond i]}\,\text{IC}}{\Diamond i, \Diamond j \vdash @_i[j \lor \Diamond j] \lor @_j[i \lor \Diamond i]}\,\text{Ded}}{\Diamond i, \Diamond j \vdash @_i j \lor @_i\Diamond j \lor @_j i \lor @_j\Diamond i}\,\text{Distr}^{@}_{\lor}}{\Diamond i, \Diamond j \vdash @_i j \lor @_i\Diamond j \lor @_j\Diamond i}\,\text{B}_{@}}{\vdash \Diamond i \land \Diamond j \to (@_i\Diamond j \lor @_i j \lor @_j\Diamond i)}\,\text{Ded}
$$

Next we consider the other direction, and show that we can also derive $\Box(j \to [i \lor \Diamond i]) \lor \Box(i \to [j \lor \Diamond j])$ from $\Diamond i \land \Diamond j \to (@_i\Diamond j \lor @_i j \lor @_j\Diamond i)$.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{given}}{\vdash \Diamond i \land \Diamond j \to (@_i\Diamond j \lor @_i j \lor @_j\Diamond i)}}{\Diamond i, \Diamond j \vdash @_i\Diamond j \lor @_i j \lor @_j\Diamond i}\,\text{DedRev}}{@_k\Diamond i, @_k\Diamond j \vdash @_k@_i\Diamond j \lor @_k@_i j \lor @_k@_j\Diamond i}\,\text{SNec}_{@}\text{, fresh k}}{@_k\Diamond i, @_k\Diamond j \vdash @_i\Diamond j \lor @_i j \lor @_j\Diamond i}\,\text{Agree}}{@_k\Diamond i, @_k\Diamond j \vdash @_i\Diamond j \lor @_i j \lor @_j i \lor @_j\Diamond i}\,\text{B}_{@}}{@_k\Diamond i, @_k\Diamond j, \neg(@_j i \lor @_j\Diamond i) \vdash @_i\Diamond j \lor @_i j}\,\text{DedRev}}{@_k\Diamond i, @_k\Diamond j, \neg(@_j i \lor @_j\Diamond i), @_i i \vdash @_i\Diamond j \lor @_i j}\,\text{W}}{@_k\Diamond i, @_k\Diamond j, \neg(@_j i \lor @_j\Diamond i) \vdash @_i i \to @_i\Diamond j \lor @_i j}\,\text{Ded}}{@_k\Diamond i, @_k\Diamond j, \neg(@_j i \lor @_j\Diamond i) \vdash @_i(i \to [\Diamond j \lor j])}\,\text{Distr}^{@}_{\lor}\text{, Distr}^{@}_{\to}}{@_k\Diamond j, \neg(@_j i \lor @_j\Diamond i) \vdash @_k\Box(i \to [\Diamond j \lor j])}\,\text{PR}}{@_k\Diamond j, \neg @_k\Box(i \to [\Diamond j \lor j]) \vdash @_j i \lor @_j\Diamond i}\,\text{cp}}{@_k\Diamond j, \neg @_k\Box(i \to [\Diamond j \lor j]), @_j j \vdash @_j i \lor @_j\Diamond i}\,\text{W}}{@_k\Diamond j, \neg @_k\Box(i \to [\Diamond j \lor j]) \vdash @_j j \to (@_j i \lor @_j\Diamond i)}\,\text{Ded}}{@_k\Diamond j, \neg @_k\Box(i \to [\Diamond j \lor j]) \vdash @_j(j \to [i \lor \Diamond i])}\,\text{Distr}^{@}_{\lor}\text{, Distr}^{@}_{\to}}{\neg @_k\Box(i \to [\Diamond j \lor j]) \vdash @_k\Box(j \to [i \lor \Diamond i])}\,\text{PR}}{\vdash @_k\Box(i \to [\Diamond j \lor j]) \lor @_k\Box(j \to [i \lor \Diamond i])}\,\text{Ded}}{\vdash @_k[\Box(i \to [\Diamond j \lor j]) \lor \Box(j \to [i \lor \Diamond i])]}\,\text{Distr}^{@}_{\lor}}{\vdash \Box(i \to [j \lor \Diamond j]) \lor \Box(j \to [i \lor \Diamond i])}\,\text{VS}
$$

Finally, we show the modal axiom $\Box(\varphi \land \Box\varphi \to \psi) \lor \Box(\psi \land \Box\psi \to \varphi)$ can be derived from one of the pure axioms. In this case our choice falls on $\Diamond i \land \Diamond j \to (@_i\Diamond j \lor @_i j \lor @_j\Diamond i)$. Bearing in mind that both pure axioms are equivalent and that $\Box(i \to [j \lor \Diamond j]) \lor \Box(j \to [i \lor \Diamond i])$ is an instantiation of the modal axiom (with $\neg i$ and $\neg j$), this suffices to show all three axioms are all equivalent.

(Due to its size the proof had to be rotated to fit the page)



B Additional computer proofs

In this appendix a few more (unannotated) computer proofs are collected.

B.1 Example 1, a simple induction

Query:

```
start([dia nom(i),@nom(i):dia nom(i)]::-forall(dia(dia nom(i),[n]),[n])).
```

Terminal output:

```
?- start([dia nom(i),@nom(i):dia nom(i)]::-forall(dia(dia nom(i),[n]),
[n])).
[[@nom(*):dia(nom(i), [1]), @nom(i):dia(nom(i), [1])]::-[@nom(*):forall
(dia(dia(nom(i), [1]), [n]), [n])], [R ind, [[@nom(*):dia(nom(i), [1])]
::-[@nom(*):dia(nom(i), [1])], [id]], [[@nom(*):dia(dia(nom(i), [1]), [
n]), @nom(i):dia(nom(i), [1])]::-[@nom(*):dia(dia(nom(i), [1]), [1, n])
], [L dia-n, [[@nom(*):dia(nom(nW), [n]), @nom(i):dia(nom(i), [1]), @no
m(nW):dia(nom(i), [1])]::-[@nom(*):dia(dia(nom(i), [1]), [1, n])], [R d
ia-n, [[@nom(i):dia(nom(i), [1]), @nom(nW):dia(nom(i), [1])]::-[@nom(nW
):dia(dia(nom(i), [1]), [1])], [R dia-n, [[@nom(i):dia(nom(i), [1])]::-
[@nom(i):dia(nom(i), [1])], [id]]]]]]]]]]
```

Formatted proof:

$$\dfrac{\dfrac{\text{id}}{@.\Diamond i \vdash @.\Diamond i} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\text{id}}{@_i\Diamond i \vdash @_i\Diamond i}}{@_i\Diamond i, @_{nw}\Diamond i \vdash @_{nw}\Diamond\Diamond i}\ \text{R dia-n}}{@.\Diamond^n nW, @_i\Diamond i, @_{nw}\Diamond i \vdash @.\Diamond^{1+n}\Diamond i}\ \text{R dia-n}}{@.\Diamond^n\Diamond i, @_i\Diamond i \vdash @.\Diamond^{1+n}\Diamond i}\ \text{L dia-n}}{@.\Diamond i, @_i\Diamond i \vdash @.\{\Diamond^n\Diamond i | n \in N\}}\ \text{R ind}$$

B.2 Example 2, BG$^+$

A computer version of A.2.1

Query:

```
start(::- @ nom(i):  box(!  nom(j):  @ nom(i):  dia(nom(j),[n]), [n])).
```

Terminal output:

```
?- start(::- @ nom(i): box(! nom(j): @ nom(i): dia(nom(j),[n]), [n])).
[[]::-[@nom(i):box(!nom(j): @nom(i):dia(nom(j), [n]), [n])], [R box-n,
[[@nom(i):dia(nom(nC), [n])]::-[@nom(nC):!nom(j): @nom(i):dia(nom(j), [
n])], [R name, [[@nom(i):dia(nom(nC), [n])]::-[@nom(nC): @nom(i):dia(no
m(nC), [n])], [R agree, [[@nom(i):dia(nom(nC), [n])]::-[@nom(i):dia(nom
(nC), [n])], [id]]]]]]]]
```

Formatted proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{id}}{@_i\Diamond^n nC \vdash @_i\Diamond^n nC}}{@_i\Diamond^n nC \vdash @_{nC}@.\Diamond^n nC}\ \text{R agree}}{@_i\Diamond^n nC \vdash @_{nC} \downarrow_j @.\Diamond^n j}\ \text{R name}}{\vdash @_i\Box^n \downarrow_j @_i\Diamond^n j}\ \text{R box-n}$$

B.3 Example 3, symmetry

A computer version of A.3.1

Query:

```
start(@nom(I):(nom(J) => box dia nom(J))::-varphi => box dia varphi).
```

Terminal output:

```
?- start(@nom(I):(nom(J) => box dia nom(J))::-varphi => box dia varphi).
[[@nom(*):nom(*)implies box(dia(nom(*), [1]), [1])]::-[@nom(*):varphi i
mplies box(dia(varphi, [1]), [1])], [R imply, [[@nom(*):varphi, @nom(*)
:nom(*)implies box(dia(nom(*), [1]), [1])]::-[@nom(*):box(dia(varphi, [
1]), [1])], [R box-n, [[@nom(*):varphi, @nom(*):dia(nom(nX), [1]), @nom
(*):nom(*)implies box(dia(nom(*), [1]), [1])]::-[@nom(nX):dia(varphi, [
1])], [L imply, [[]::-[@nom(*):nom(*)], [T-@]], [[@nom(*):varphi, @nom(
*):box(dia(nom(*), [1]), [1]), @nom(*):dia(nom(nX), [1])]::-[@nom(nX):d
ia(varphi, [1])], [L box-n, [[@nom(*):varphi, @nom(nX):dia(nom(*), [1])
]::-[@nom(nX):dia(varphi, [1])], [R dia-n, [[@nom(*):varphi]::-[@nom(*)
:varphi], [id]]]]]]]]]]]


This is pdfeTeX, Version 3.141592-1.21a-2.2 (Web2C 7.5.4)
entering extended mode
done

I = *,
J = *

Yes
```

Formatted proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{T-}@}{\vdash @.\ast} \qquad \dfrac{\dfrac{\dfrac{\text{id}}{@.\varphi \vdash @.\varphi}\ \text{R dia-n}}{@.\varphi, @_{nX}\Diamond\ast \vdash @_{nX}\Diamond\varphi}\ \text{L box-n}}{@.\varphi, @.\Box\Diamond\ast, @.\Diamond nX \vdash @_{nX}\Diamond\varphi}}{@.\varphi, @.\Diamond nX, @.(\ast \to \Box\Diamond\ast) \vdash @_{nX}\Diamond\varphi}\ \text{L imply}}{@.\varphi, @.(\ast \to \Box\Diamond\ast) \vdash @.\Box\Diamond\varphi}\ \text{R box-n}}{@.(\ast \to \Box\Diamond\ast) \vdash @.(\varphi \to \Box\Diamond\varphi)}\ \text{R imply}$$

B.4 Example 4, CD-eq

A computer version of A.2.4

Query:

```
start( ::-
       (@ nom(i):dia(!  nom(k):@ nom(j):dia(nom(k),[n]),[m])) <=>
       (@ nom(j):dia(!  nom(k):@ nom(i):dia(nom(k),[m]),[n])) ).
```

Terminal output:

```
?- start(::-
|        (@ nom(i):dia(! nom(k):@ nom(j):dia(nom(k),[n]),[m])) <=>
|        (@ nom(j):dia(! nom(k):@ nom(i):dia(nom(k),[m]),[n]))
|     ).
[[]::-[@nom(*): (@nom(i):dia(!nom(k): @nom(j):dia(nom(k), [n]), [m])) e
quals (@nom(j):dia(!nom(k): @nom(i):dia(nom(k), [m]), [n]))], [R equal,
 [[@nom(*): @nom(i):dia(!nom(k): @nom(j):dia(nom(k), [n]), [m])]::-[@no
m(*): @nom(j):dia(!nom(k): @nom(i):dia(nom(k), [m]), [n])], [L agree, [
[@nom(i):dia(!nom(k): @nom(j):dia(nom(k), [n]), [m])]::-[@nom(*): @nom(
j):dia(!nom(k): @nom(i):dia(nom(k), [m]), [n])], [L dia-n, [[@nom(i):di
a(nom(nT), [m]), @nom(nT):!nom(k): @nom(j):dia(nom(k), [n])]::-[@nom(*)
: @nom(j):dia(!nom(k): @nom(i):dia(nom(k), [m]), [n])], [R agree, [[@no
m(i):dia(nom(nT), [m]), @nom(nT):!nom(k): @nom(j):dia(nom(k), [n])]::-[
@nom(j):dia(!nom(k): @nom(i):dia(nom(k), [m]), [n])], [L name, [[@nom(i
):dia(nom(nT), [m]), @nom(nT): @nom(j):dia(nom(nT), [n])]::-[@nom(j):di
a(!nom(k): @nom(i):dia(nom(k), [m]), [n])], [L agree, [[@nom(i):dia(nom
(nT), [m]), @nom(j):dia(nom(nT), [n])]::-[@nom(j):dia(!nom(k): @nom(i):
dia(nom(k), [m]), [n])], [R dia-n, [[@nom(i):dia(nom(nT), [m])]::-[@nom
(nT):!nom(k): @nom(i):dia(nom(k), [m])], [R name, [[@nom(i):dia(nom(nT)
, [m])]::-[@nom(nT): @nom(i):dia(nom(nT), [m])], [R agree, [[@nom(i):di
a(nom(nT), [m])]::-[@nom(i):dia(nom(nT), [m])], [id]]]]]]]]]]]]]]]]]],
[[@nom(*): @nom(j):dia(!nom(k): @nom(i):dia(nom(k), [m]), [n])]::-[@nom
(*): @nom(i):dia(!nom(k): @nom(j):dia(nom(k), [n]), [m])], [L agree, [[
@nom(j):dia(!nom(k): @nom(i):dia(nom(k), [m]), [n])]::-[@nom(*): @nom(i
):dia(!nom(k): @nom(j):dia(nom(k), [n]), [m])], [L dia-n, [[@nom(j):dia
(nom(nU), [n]), @nom(nU):!nom(k): @nom(i):dia(nom(k), [m])]::-[@nom(*):
 @nom(i):dia(!nom(k): @nom(j):dia(nom(k), [n]), [m])], [R agree, [[@nom
(j):dia(nom(nU), [n]), @nom(nU):!nom(k): @nom(i):dia(nom(k), [m])]::-[@
nom(i):dia(!nom(k): @nom(j):dia(nom(k), [n]), [m])], [L name, [[@nom(j)
:dia(nom(nU), [n]), @nom(nU): @nom(i):dia(nom(nU), [m])]::-[@nom(i):dia
(!nom(k): @nom(j):dia(nom(k), [n]), [m])], [L agree, [[@nom(i):dia(nom(
nU), [m]), @nom(j):dia(nom(nU), [n])]::-[@nom(i):dia(!nom(k): @nom(j):d
ia(nom(k), [n]), [m])], [R dia-n, [[@nom(j):dia(nom(nU), [n])]::-[@nom(
nU):!nom(k): @nom(j):dia(nom(k), [n])], [R name, [[@nom(j):dia(nom(nU),
 [n])]::-[@nom(nU): @nom(j):dia(nom(nU), [n])], [R agree, [[@nom(j):dia
(nom(nU), [n])]::-[@nom(j):dia(nom(nU), [n])], [id]]]]]]]]]]]]]]]]]]]]
```

Formatted proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{id}}{@_i\Diamond^m nT \vdash @_i\Diamond^m nT}}{@_i\Diamond^m nT \vdash @_{nT}@_i\Diamond^m nT}\,\text{R agree}}{@_i\Diamond^m nT \vdash @_{nT}\downarrow_k @_i\Diamond^m k}\,\text{R name}}{@_i\Diamond^m nT, @_j\Diamond^n nT \vdash @_j\Diamond^n \downarrow_k @_i\Diamond^m k}\,\text{R dia-n}}{@_i\Diamond^m nT, @_{nT}@_j\Diamond^n nT \vdash @_j\Diamond^n \downarrow_k @_i\Diamond^m k}\,\text{L agree}}{@_i\Diamond^m nT, @_{nT}\downarrow_k @_j\Diamond^n k \vdash @_j\Diamond^n \downarrow_k @_i\Diamond^m k}\,\text{L name}}{@_i\Diamond^m nT, @_{nT}\downarrow_k @_j\Diamond^n k \vdash @.@_j\Diamond^n \downarrow_k @_i\Diamond^m k}\,\text{R agree}}{@_i\Diamond^m \downarrow_k @_j\Diamond^n k \vdash @.@_j\Diamond^n \downarrow_k @_i\Diamond^m k}\,\text{L dia-n}}{@.@_i\Diamond^m \downarrow_k @_j\Diamond^n k \vdash @.@_j\Diamond^n \downarrow_k @_i\Diamond^m k}\,\text{L agree} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{id}}{@_j\Diamond^n nU \vdash @_j\Diamond^n nU}}{@_j\Diamond^n nU \vdash @_{nU}@_j\Diamond^n nU}\,\text{R agree}}{@_j\Diamond^n nU \vdash @_{nU}\downarrow_k @_j\Diamond^n k}\,\text{R name}}{@_i\Diamond^m nU, @_j\Diamond^n nU \vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k}\,\text{R dia-n}}{@_j\Diamond^n nU, @_{nU}@_i\Diamond^m nU \vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k}\,\text{L agree}}{@_j\Diamond^n nU, @_{nU}\downarrow_k @_i\Diamond^m k \vdash @_i\Diamond^m \downarrow_k @_j\Diamond^n k}\,\text{L name}}{@_j\Diamond^n nU, @_{nU}\downarrow_k @_i\Diamond^m k \vdash @.@_i\Diamond^m \downarrow_k @_j\Diamond^n k}\,\text{R agree}}{@_j\Diamond^n \downarrow_k @_i\Diamond^m k \vdash @.@_i\Diamond^m \downarrow_k @_j\Diamond^n k}\,\text{L dia-n}}{@.@_j\Diamond^n \downarrow_k @_i\Diamond^m k \vdash @.@_i\Diamond^m \downarrow_k @_j\Diamond^n k}\,\text{L agree}}{\vdash @.(@_i\Diamond^m \downarrow_k @_j\Diamond^n k \leftrightarrow @_j\Diamond^n \downarrow_k @_i\Diamond^m k)}\,\text{R equal}$$

C Program code

C.1 modelchecker.pl

```
%%% The framechecker may reject possible counter-models, but never accepts something that is not a counter-model


:- ensure_loaded('input.pl').
:- ensure_loaded('misc.pl').

:- dynamic world/1, link/3.

%%% check(+Node): check whether we can easily make acounter-model based on Node
check(node(left(S,C,B,F),right(S2,C2,D,F2))):- destroy_model, build_model(S), check_that_formulas_succeed(C),
                                               check_that_formulas_succeed(B),check_that_formulas_succeed(F),
                                               check_that_formulas_fail(S2), check_that_formulas_fail(C2),
                                               check_that_formulas_fail(D), check_that_formulas_fail(F2).

build_model([]).
build_model([@ S : true| Rest]):- makeworld(S), build_model(Rest).
build_model([@ S : (dia T)| Rest]):- makeworld(S), makeworld(T), makalink(S,T, [1]), build_model(Rest).
build_model([@ S : dia(T, P)| Rest]):- makeworld(S), makeworld(T), makelink(S,T, P), build_model(Rest).
makeworld(S):- (world(S), !); assert(world(S)).
makelink(S,T, P):- (link(S,T,P),!) ; assert(link(S,T,P)).
destroy_model:- retractall(world(_)), retractall(link(_,_,_)).


check_that_formulas_fail([]).
check_that_formulas_fail([Formula|OtherFormulas]):- check_fail(Formula), check_that_formulas_fail(OtherFormulas).

check_fail(@  nom(S) : nom(T) ):- !, S \== T.          %% the assumption is all nominals denote distinct worlds in the model.
check_fail(@ _S : false):- !.
check_fail(@  S : _              ):- \+ world(S), !, fail.   %% claims about non-existant worlds, besides identity and truth, make no sense
check_fail(@  S : ( not F)     ):- check_succeed(@ S : F ).
check_fail(@  S : ( dia F)     ):- check_succeed(@ S : box not F  ).
check_fail(@  S : ( box F)     ):- check_succeed(@ S : dia not F  ).
check_fail(@  S :  dia(F,P)    ):- check_succeed(@ S : box(not F,P)  ).
check_fail(@  S :  box(F,P)    ):- check_succeed(@ S : dia(not F,P)  ).
check_fail(@  S : ( P or Q )   ):- check_fail(@ S : P ), check_fail(@ S : Q ).
check_fail(@  S : ( P and Q ) ):- check_fail(@ S : P ); check_fail(@ S : Q ).
check_fail(@  S : ( P implies Q)):- check_succeed(@ S : P ), check_fail(@ S : Q ).
check_fail(@  S : ( P equals Q)):- check_fail(@ S : ( P and Q)), check_succeed(@ S : ( P or Q )).
check_fail(@ _S : ( @ T : F ) ):- check_fail(@ T : F ).
check_fail(@  S : ( ! T : F ) ):- replace(T, S, F, F2), check_fail(@ S : F2 ).

check_fail(@  S : forall( box(F, P2), P) ):- ord_subset(P, P2), parameters(F, Ps), ord_disjoint(P, Ps),
                                              ord_subtract(P2, P, Prest), reachableworld(S, Worlds),
                                              (Prest = [] -> (member(T, Worlds), check_fail(@ T : F ))
                                                           ; (member(T, Worlds), check_fail(@ T : dia(F, Prest)))).

check_that_formulas_succeed([]).
check_that_formulas_succeed([Formula|OtherFormulas]):- check_succeed(Formula), check_that_formulas_succeed(OtherFormulas).

check_succeed(@  nom(S) : nom(T)    ):- !, S == T.       %% the assumption is all nominals denote distinct worlds in the model.
check_succeed(@ _S : true):- !.
check_succeed(@  S : _    ):- \+ world(S), !, fail.    %% claims about non-existant worlds, besides identity and truth, make no sense
check_succeed(@  S : ( not F )    ):- check_fail(@ S : F).
check_succeed(@  S : ( dia F )    ):- link(S,T,[1]), check_succeed(@ T : F ).
check_succeed(@  S : ( box F )    ):- forall(link(S,T,[1]), check_succeed(@ T : F)).
check_succeed(@  S :  dia( F,P )    ):- link(S,T,P), check_succeed(@ T : F ).
check_succeed(@  S :  box( F,P )    ):- forall((link(S,T,[L]), select(L,P,P2) ), (P2=[] -> check_succeed(@ T : F) ; check_succeed(@ T : box(F,P2) ) )).
check_succeed(@  S : ( P or Q )  ):- check_succeed(@ S : P ); check_succeed(@ S : Q ).
check_succeed(@  S : ( P and Q ) ):- check_succeed(@ S : P ), check_succeed(@ S : Q ).
check_succeed(@  S : ( P implies Q)):- check_fail(@ S : P ); check_succeed(@ S : Q ).
check_succeed(@  S : ( P equals Q)):- check_succeed(@ S : ( P and Q)); check_fail(@ S : ( P or Q )).
check_succeed(@ _S : ( @ T : F ) ):- check_succeed( @ T : F ).
check_succeed(@  S : ( ! T : F ) ):- replace(T, S, F, F2), check_succeed(@ S : F2 ).


check_succeed(@  S : forall( box(F, P2), P) ):- ord_subset(P, P2), parameters(F, Ps), ord_disjoint(P, Ps),
                                                 ord_subtract(P2, P, Prest), reachableworld(S, Worlds),
                                                 (Prest = [] -> forall(member(T, Worlds), check_succeed(@ T : F ))
                                                              ; forall(member(T, Worlds), check_succeed(@ T : box(F, Prest)))).

%%% find all reachable worlds from a given starting worl in the current model.
reachableworld(S, Worlds):-growreachableset([S], [S], Worlds).
growreachableset(Worlds, [], Worlds):- !.
growreachableset(OldSet, NewSet, Worlds):-findall(W, S^(member(S, NewSet), link(S,W,_)), NewerSet), sort(NewerSet,NewestSet),
                                          ord_union(OldSet, NewestSet, Total, ReallyNew), growreachableset(Total, ReallyNew, Worlds).
```

C.2 input.pl

```prolog
%%% new infix and prefix operators to make writing formulas in prolog easier.
:- op(460, xfx, ['|-', ::- ]).
:- op(460, fx, ['|-', ::- ]).


:- op(400, fy, [not,~,dia,box]).
:- op(440, xfy, [implies, =>]).
:- op(430, xfy, [or, \/]).
:- op(420, xfy, [and, /\]).
:- op(410, xfy, [equal, equals, <=>]).

:- op(454, fx, [@, at, bind, !, name, downarrow]).
:- op(455, xfy, : ).


%%% translate formulas to common internal format
standardize( A,             A) :- atomic(A) ; A = nom(_) ; A = var(_).
standardize( not A,         not A2)        :- standardize(A, A2).
standardize( ~ A,           not A2)        :- standardize(A, A2).
standardize( box A,         box(A2, [1]))  :- standardize(A, A2).
standardize( dia A,         dia(A2, [1]))  :- standardize(A, A2).
standardize( A and B,       A2 and B2)     :- standardize(A, A2), standardize(B, B2).
standardize( A /\ B,         A2 and B2)    :- standardize(A, A2), standardize(B, B2).
standardize( A or B,        A2 or B2)      :- standardize(A, A2), standardize(B, B2).
standardize( A \/ B,         A2 or B2)     :- standardize(A, A2), standardize(B, B2).
standardize( A implies B, A2 implies B2) :- standardize(A, A2), standardize(B, B2).
standardize( A => B,        A2 implies B2) :- standardize(A, A2), standardize(B, B2).
standardize( A equals B,  A2 equals B2)  :- standardize(A, A2), standardize(B, B2).
standardize( A <=> B,       A2 equals B2)  :- standardize(A, A2), standardize(B, B2).
standardize( A equal B,   A2 equals B2)  :- standardize(A, A2), standardize(B, B2).

standardize( @ I : B,       @ I : B2)      :- standardize(B, B2).
standardize( at I : B,      @ I : B2)      :- standardize(B, B2).
standardize( at(I, B),      @ I : B2)      :- standardize(B, B2).
standardize( ! I : B,       ! I : B2)      :- standardize(B, B2).
standardize( bind I : B,    ! I : B2)      :- standardize(B, B2).
standardize( bind(I, B),    ! I : B2)      :- standardize(B, B2).
standardize( name I : B,    ! I : B2)      :- standardize(B, B2).
standardize( name(I, B),    ! I : B2)      :- standardize(B, B2).
standardize( downarrow I : B,! I : B2)     :- standardize(B, B2).
standardize( downarrow(I,B),! I : B2)      :- standardize(B, B2).
standardize( A ::- B,       A2 ::- B2)     :- standardize(A, A3), (is_list(A3)-> A2=A3 ; A2=[A3]), standardize(B, B3), (is_list(B3)-> B2=B3 ; B2=[B3]).
standardize( ::- B,         [] ::- B2)     :- standardize(B, B3), (is_list(B3)-> B2=B3 ; B2=[B3]).
standardize( A '|-' B,      A2 ::- B2)     :- standardize(A, A3), (is_list(A3)-> A2=A3 ; A2=[A3]), standardize(B, B3), (is_list(B3)-> B2=B3 ; B2=[B3]).
standardize( '|-' B,        [] ::- B2)     :- standardize(B, B3), (is_list(B3)-> B2=B3 ; B2=[B3]).
standardize([],             []).
standardize([A|R],          [A2|R2])       :- standardize(A, A2), standardize(R, R2).


standardize( box(A, []),    A2)            :- standardize(A, A2),!.
standardize( dia(A, []),    A2)            :- standardize(A, A2),!.
standardize( forall(A, []), A2)            :- standardize(A, A2),!.
standardize( set(A, []), A2)          :- standardize(A, A2),!.
standardize( box(A, P),   box(A2,P2))      :- standardize(A, A2), (is_list(P) -> nsort(P, P2); P2= [P]). %% par_box
standardize( dia(A, P),   dia(A2,P2))      :- standardize(A, A2), (is_list(P) -> nsort(P, P2); P2= [P]). %% par_dia
standardize( set(A, P),forall(A2,P2)) :- standardize(A, A2), (is_list(P) -> nsort(P, P2); P2= [P]). %% inf_term
standardize( forall(A, P),forall(A2,P2)) :- standardize(A, A2), (is_list(P) -> nsort(P, P2); P2= [P]). %% inf_term


%%% sort the list, and sum any numbers at the head together
nsort(L, L2):- sort(L, L3), sumfront(L3,L2).
sumfront([A],[A]).
sumfront([A,B|T], L):- (number(A), number(B)) -> (R is A + B, sumfront([R|T], L))
                                                ;  L = [A,B|T].
```

C.3 misc.pl

```
:- ensure_loaded('input.pl').

%%% create fresh nominals
:-abolish(curnom,2).
:-assert(curnom([],65)).

addto([], [65]).
addto([N|T], [Nplus|T2]):- Nplus is (N + 14) mod 26 + 65, (Nplus = 65 -> addto(T, T2) ; T=T2).

newnom(nom(A)):- curnom(T,N), atom_chars(A,[110,N|T]),
                 Nplus is (N + 14) mod 26 + 65, retract(curnom(T,N)),
                 (Nplus = 65
                 -> addto(T, T2), assert(curnom(T2, Nplus))
                  ; assert(curnom(T, Nplus))).

%%% replace one nominal by another throughout a formula
replace(S, S, Phi, Phi):- !.
replace(S, T, S, T)     :- !.
replace(_, _, A, A)     :- atomic(A) ; A = nom(_) ; A = var(_).

replace(S, T, Phi and Psi,     PhiRep and PsiRep)     :- replace(S, T, Phi, PhiRep), replace(S, T, Psi, PsiRep).
replace(S, T, Phi or Psi,      PhiRep or PsiRep)      :- replace(S, T, Phi, PhiRep), replace(S, T, Psi, PsiRep).
replace(S, T, Phi implies Psi, PhiRep implies PsiRep) :- replace(S, T, Phi, PhiRep), replace(S, T, Psi, PsiRep).
replace(S, T, Phi equals Psi,  PhiRep equals PsiRep)  :- replace(S, T, Phi, PhiRep), replace(S, T, Psi, PsiRep).

replace(S, T, not Phi,         not PhiRep)            :- replace(S, T, Phi, PhiRep).
replace(S, T, dia Phi,         dia PhiRep)            :- replace(S, T, Phi, PhiRep).
replace(S, T, box Phi,         box PhiRep)            :- replace(S, T, Phi, PhiRep).

replace(S, T, @ S : Phi,       @ T : PhiRep)          :- !, replace(S, T, Phi, PhiRep).
replace(S, T, @ I : Phi,       @ I : PhiRep)          :- replace(S, T, Phi, PhiRep).

replace(S, T,  ! T : Phi,      ! Q : PhiRepRep)       :- !, newnom(Q), replace(T, Q, Phi, PhiRep),  replace(S, T, PhiRep, PhiRepRep).
replace(S, _T, ! S : Phi,      ! S : Phi)             :- !.
replace(S, T,  ! I : Phi,      ! I : PhiRep)          :- replace(S, T, Phi, PhiRep).

replace(_, _,  [],             []).
replace(S, T, [Phi|Phis],      [PhiRep| PhiReps])     :- replace(S, T, Phi, PhiRep), replace(S, T, Phis, PhiReps).

replace(S, T, dia(Phi, P),     dia(PhiRep, PRep))     :- replace(S, T, Phi, PhiRep),replace(S, T, P, PRep).
replace(S, T, box(Phi, P),     box(PhiRep, PRep))     :- replace(S, T, Phi, PhiRep),replace(S, T, P, PRep).

replace(S, T, forall(Phi, P),  forall(PhiRep, PRep))  :- (T=P ; member(T,P))
                                                          ->  (newnom(nom(Q)), replace(T, Q, Phi, PhiTmp),replace(T, Q, P, PTmp),
                                                               replace(S, T, PhiTmp, PhiRep), replace(S, T, PTmp, PRep) )
                                                          ; (  (S=P ; member(S,P))
                                                               -> (PhiRep=Phi, PRep=P)
                                                               ;  replace(S, T, Phi, PhiRep),replace(S, T, P, PRep)
                                                          ).
replece(S, T, node(Left,Right), node(LeftRep,RightRep)):- replace(S, T, Left, LeftRep),replace(S, T, Right, RightRep).
replace(S, T, left(Si, C, B, I), left(Si2, C2, B2, I2)):- replace(S, T, Si, Si2), replace(S, T, C, C2), replace(S, T, B, B2), replace(S, T, I, I2).
replace(S, T, right(Si,C, D, I), right(Si2,C2, D2, I2)):- replace(S, T, Si, Si2), replace(S, T, C, C2), replace(S, T, D, D2), replace(S, T, I, I2).

%%% collect the list of all parameters in a formula
parameters( A, [])             :- atomic(A) ; A = nom(_) ; A = var(_).
parameters( not A, Ps)         :- parameters( A, Ps).
parameters( dia A, Ps)         :- parameters( A, Ps).
parameters( box A, Ps)         :- parameters( A, Ps).
parameters( A and B, Ps)       :- parameters( A, P1s), parameters( B, P2s), union(P1s,P2s,Ps).
parameters( A or B, Ps)        :- parameters( A, P1s), parameters( B, P2s), union(P1s,P2s,Ps).
parameters( A implies B, Ps)   :- parameters( A, P1s), parameters( B, P2s), union(P1s,P2s,Ps).
parameters( A equals B, Ps)    :- parameters( A, P1s), parameters( B, P2s), union(P1s,P2s,Ps).
parameters( @ _I : A, Ps)      :- parameters( A, Ps).
parameters( ! _I : A, Ps)      :- parameters( A, Ps).
parameters( box(A, P), Ps)     :- parameters( A, P1s), union(P1s,P,Ps).
parameters( dia(A, P), Ps)     :- parameters( A, P1s), union(P1s,P,Ps).
parameters( forall(A, P), Ps)  :- parameters( A, P1s), union(P1s,P,Ps).

%%% instantiate the parameter k with the list L
inst(_K, _L, A, A)                       :- atomic(A) ; A = nom(_) ; A = var(_).
inst(K, L, not A, not A2)                :- inst(K, L, A, A2).
inst(K, L, A and B, A2 and B2 )          :- inst(K, L, A, A2), inst(K, L, B, B2).
inst(K, L, A or B, A2 or B2)             :- inst(K, L, A, A2), inst(K, L, B, B2).
inst(K, L, A implies B, A2 implies B2)   :- inst(K, L, A, A2), inst(K, L, B, B2).
inst(K, L, A equals B, A2 equals B2)     :- inst(K, L, A, A2), inst(K, L, B, B2).
```

```
   inst(K, L, @ I : A, @ I : A2)          :- inst(K, L, A, A2).
   inst(K, L, ! I : A, ! I : A2)          :- inst(K, L, A, A2).
   inst(K, L, box(A, P), Res )            :- inst(K, L, A, A2), (select(K, P, Ptmp) -> (ord_union(Ptmp, L, P3), sumfront(P3,P2)) ; P2=P),
                                             ((P2=[];P2=[0]) -> Res = A2 ; Res = box(A2, P2)).
** inst(K, L, dia(A, P), Res )            :- inst(K, L, A, A2), (select(K, P, Ptmp) -> (ord_union(Ptmp, L, P3), sumfront(P3,P2)); P2=P),
                                             ((P2=[];P2=[0]) -> Res = A2 ; Res = dia(A2, P2)) .
   inst(K, L, forall(A, P), forall(A2, P)):- (member(K, P) -> A2=A ; inst(K, L, A, A2)).

   %%% select some 'amount' (numeric) from a list of parameters
** selectnum(L, [N|P], R):- number(N), N >= L, N2 is N - L, (N2 = 0 -> R = P ; R = [N2|P]).
```

C.4 print.pl

```
:- ensure_loaded('input.pl').
:- discontiguous pprint/3.

%%% print proofs as pdf, for improved legibility
printpdf(Formula) :- catch((open('test.tex' , write, Stream),
                        write(Stream, '\\documentclass{article}\\usepackage{bussproofs} \\usepackage{latexsym} \\begin{document}'),
                        write(Stream, '\\newcommand{\\implies}{\\rightarrow}\\newcommand{\\equals}{\\leftrightarrow}\\newcommand{\\Dia}{\\Diamond}\\begin{prooftree}'),
                        pprint(Stream, Formula, _),
                        write(Stream, '\\end{prooftree} \n \\end{document} '), close(Stream),
                        shell('pdflatex -interaction batchmode test.tex'), shell('acro test.pdf'),
                        write('done')),_, write('printing error')).

printdvi(Formula) :- open('test.tex' , write, Stream),
                        write(Stream, '\\documentclass{article}\\usepackage{bussproofs} \\usepackage{latexsym} \\begin{document}'),
                        write(Stream, '\\newcommand{\\implies}{\\rightarrow}\\newcommand{\\equals}{\\leftrightarrow}\\newcommand{\\Dia}{\\Diamond}\\begin{prooftree}'),
                        pprint(Stream, Formula, _),
                        write(Stream, '\\end{prooftree} \n \\end{document} '), close(Stream),
                        shell('latex test.tex'),
                        write('done').

pprint(Str, Var, _)  :- var(Var), !, write(Str,Var). %%% in case we encounter a variable we don't want to match any of the following cases.


pprint(Str, [Seq, [Rule, Le, Ri]], _):- pprint(Str, Le, _), pprint(Str, Ri, _),
                                        write(Str,'\\RightLabel{'), write(Str,Rule), write(Str,'}'), nl,
                                        write(Str,'\\BinaryInfC{$'), pprint(Str,Seq, 0), write(Str,'$}\n').
pprint(Str, [Seq, [Rule, Ne]], _)     :- pprint(Str, Ne, _), write(Str,'\\RightLabel{'), write(Str,Rule), write(Str,'}') , nl,
                                        write(Str,'\\UnaryInfC{$'), pprint(Str, Seq, _), write(Str,'$}\n').
pprint(Str, [Seq, [Axiom]], _)        :- write(Str,'\\AxiomC{'), write(Str,Axiom), write(Str,'}'),  nl,
                                        write(Str,'\\UnaryInfC{$'), pprint(Str, Seq, _), write(Str,'$}\n').

pprint(Str, true, _)  :- write(Str,'\\top').
pprint(Str, false, _) :- write(Str,'\\bot').
pprint(Str, varphi, _):- write(Str,'\\varphi').
pprint(Str, psi, _)   :- write(Str,'\\psi').
pprint(Str, chi, _)   :- write(Str,'\\chi').

pprint(Str, A, _)     :- atomic(A), write(Str, A).
pprint(Str, nom(I), _):- write(Str, I).

pprint(Str, not A, _) :- write(Str,'\\lnot '), pprint(Str,A, un).
pprint(Str, dia A, _) :- write(Str,'\\Dia '), pprint(Str,A, un).
pprint(Str, box A, _) :- write(Str,'\\Box '), pprint(Str,A, un).

pprint(Str, A and B, and)   :- pprint(Str,A, and), write(Str,' \\land '), pprint(Str,B, and).
pprint(Str, A and B,_)      :- write(Str,'('), pprint(Str,A, and), write(Str,' \\land '), pprint(Str,B, and), write(Str,')').
pprint(Str, A or B, lor)     :- pprint(Str,A, lor), write(Str,' \\lor '), pprint(Str,B, ror).
pprint(Str, A or B, ror)     :- pprint(Str,A, lor), write(Str,' \\lor '), pprint(Str,B, ror).
pprint(Str, A or B,_)       :- write(Str,'('), pprint(Str,A, or), write(Str,' \\lor '), pprint(Str,B, or), write(Str,')').
pprint(Str, A implies B, lor):- pprint(Str,A, imp), write(Str,' \\implies '), pprint(Str,B, ror).
pprint(Str, A implies B,_)  :- write(Str,'('), pprint(Str,A, imp), write(Str,' \\implies '), pprint(Str,B, ror), write(Str,')').
pprint(Str, A equals B,_)   :- write(Str,'('), pprint(Str,A, eq), write(Str,' \\equals '), pprint(Str,B, eq), write(Str,')').

pprint(Str, @ nom(S): B,_) :- write(Str,'@_{'), pprint(Str,S, un), write(Str,'}'), pprint(Str,B, un).
pprint(Str, ! nom(S): B,_) :- write(Str,'\\downarrow_{'), pprint(Str,S, un), write(Str,'}'), pprint(Str,B, un).

pprint(Str, [] ::- B,_):- is_list(B),    write(Str,'\\vdash '), pprintR(Str,B).
pprint(Str, [] ::- B,_):- \+ is_list(B), write(Str,'\\vdash '), pprint(Str,B, _).
pprint(Str, A ::- [],_):- is_list(A),    pprintL(Str, A), write(Str,' \\vdash \\bot').
pprint(Str, A ::- [],_):- \+ is_list(A), pprint(Str, A), write(Str,' \\vdash \\bot ').
pprint(Str, A ::- B,_) :- is_list(A), is_list(B),        pprintL(Str, A), write(Str,' \\vdash '), pprintR(Str,B).
pprint(Str, A ::- B,_) :- is_list(A),  \+ is_list(B),    pprintL(Str, A), write(Str,' \\vdash '), pprint(Str,B, _).
pprint(Str, A ::- B,_) :- \+ is_list(A), is_list(B),     pprint(Str, A, _), write(Str,' \\vdash '), pprintR(Str,B).
pprint(Str, A ::- B,_) :- \+ is_list(A), \+ is_list(B), pprint(Str, A, _), write(Str,' \\vdash '), pprint(Str,B, _).
pprint(Str, ::- B,_)   :- is_list(B),    write(Str,'\\vdash '), pprintR(Str,B).
pprint(Str, ::- B,_)   :- \+ is_list(B), write(Str,'\\vdash '), pprint(Str,B,_).

pprintL(Str,[A])  :- !, pprint(Str,A, _).
pprintL(_Str,[]).
pprintL(Str,[H|T]):- pprint(Str,H, _), write(Str, ', '), pprintL(Str,T).

pprintR(Str,[A])  :- !, pprint(Str,A, _).
pprintR(_Str,[]).
pprintR(Str,[H|T]):- pprint(Str,H, _), write(Str, ' \\lor '), pprintR(Str,T).
```

```
pprint(Str, dia(A, []), _) :- pprint(Str,A, un).
pprint(Str, box(A, []), _) :- pprint(Str,A, un).
pprint(Str, dia(A, [1]), _) :- write(Str,'\\Dia '),  pprint(Str,A, un).
pprint(Str, box(A, [1]), _) :- write(Str,'\\Box '),  pprint(Str,A, un).
pprint(Str, dia(A, Par), _) :- write(Str,'\\Dia^'), write(Str,'{'), pprint_par(Str, Par, ','),write(Str,'}'),  pprint(Str,A, un).
pprint(Str, box(A, Par), _) :- write(Str,'\\Box^'), write(Str,'{'), pprint_par(Str, Par, ','),write(Str,'}'),  pprint(Str,A, un).
pprint(Str, forall(A, Par), _) :- write(Str,'\\{'),  pprint(Str,A, _), write(Str,'}'), pprint_par(Str, Par, ','), write(Str,'\\in N\\}').

pprint_par(Str, Par,_):- \+ is_list(Par), write(Str, Par).
pprint_par(Str, [P], _):- write(Str, P).
pprint_par(Str, [P|Ps], Con):- Ps \== [], write(Str, P), write(Str, Con),  pprint_par(Str, Ps, Con).
```

C.5 rulebase.pl

```
:- ensure_loaded('input.pl').
:- ensure_loaded('misc.pl').

%%%closing rules
rule( sig(r, 'T',    s),      arg(@ _S : true    ),
                              res(close)                   ).
rule( sig(l, 'T',    s),      arg(@ _S : false   ),
                              res(close)                   ).
rule( sig(r, 'T-@',   s),     arg(@ S : S    ),
                              res(close)                   ).
rule( sig(lr, 'id',   aa),    arg(_Phi),
                              res(close)                   ).

%%% prop rules
rule( sig(l, 'L and',    c),  arg(@ S : Phi and Psi      ),
                              res(add([@ S : Phi, @ S : Psi],  [] ))                  ).
rule( sig(r, 'R imply', c),   arg(@ S : Phi implies Psi ),
                              res(add([@ S : Phi],            [@ S : Psi]))           ).
rule( sig(r, 'R or',     c),  arg(@ S : Phi or Psi       ),
                              res(add([],                     [@ S : Phi, @ S : Psi]))).
rule( sig(r, 'R not',    c),  arg(@ S : not Phi          ),
                              res(add([@ S : Phi],            []))                    ).
rule( sig(l, 'L not',    c),  arg(@ S : not Phi          ),
                              res(add([],                     [@ S : Phi]))           ).


%%% dia/box rules
rule( sig(l, 'L dia-n', c),   arg(@ S : dia(Phi,L)       ),
                              res(add([@ S : dia(T,[L1]), Psi],[]))                   ):- newnom(T), select(L1,L,L2),
                                                                                            (L2 =[] -> (Phi \= nom(_) , Psi = @ T : Phi) ;  Psi = @ T : dia(Phi,L2)).
rule( sig(r, 'R box-n', c),   arg(@ S : box(Phi,L)       ),
                              res(add([@ S : dia(T,[L1])],       [Psi]))              ):- newnom(T),  select(L1,L,L2),
                                                                                            (L2 =[] -> Psi = @ T : Phi ;  Psi = @ T : dia(Phi,L2)).


%%% @ rules
rule( sig(l, 'L agree', c),   arg(@ _S : [@ T : Phi]     ),
                              res(add([@ T : Phi],            []))                    ).
rule( sig(r, 'R agree', c),   arg(@ _S : (@ T : Phi)     ),
                              res(add([],                     [@ T : Phi]))           ).

rule( sig(l, 'L ref',    s),  arg(@ S : S                ),                             %% always true, so no need to keep it explicit
                              res(add([],                     []))                    ).
rule( sig(l, 'L ident', s),   arg(@ S : nom(T)           ),                             %% this rule replaces the nom and bridge rule
                              res(replace(S, nom(T)))                                 ):- S \= nom(T).


%%% DA rules
rule( sig(l, 'L name',   c),  arg( @ S : (! T : Phi)     ),
                              res(add([@ S : PhiRep],         []))                    ):- replace(T,S, Phi, PhiRep).
rule( sig(r, 'R name',   c),  arg(@ S : (! T : Phi)      ),
                              res(add([],                     [@ S : PhiRep]))        ):- replace(T,S, Phi, PhiRep).

%%% prop rules
rule( sig(l, 'L imply', c),   arg( @ S : Phi implies Psi ),
                              res( split(add([],                   [@ S : Phi]),
                                         add([@ S : Psi],          [])               )) ).
rule( sig(l, 'L or',     c),  arg( @ S : Phi or Psi   ),
                              res( split(add([@ S : Phi],          []),
                                         add([@ S : Psi],          [])               )) ).
rule( sig(l, 'L equal', c),   arg( @ S : Phi equals Psi ),
                              res( split(add([],                   [@ S : Phi, @ S : Psi]),
                                         add([@ S : Phi, @ S : Psi], [])             )) ).
rule( sig(r, 'R and',    c),  arg( @ S : Phi and Psi ),
                              res( split(add([],                   [@ S : Phi]),
                                         add([],                   [@ S : Psi])      )) ).
rule( sig(r, 'R equal', c),   arg( @ S : Phi equals Psi ),
                              res( split(add([@ S : Phi],          [@ S : Psi]),
                                         add([@ S : Psi],          [@ S : Phi])      )) ).


%%% dia/box rules
rule( sig(l, 'L box-n', sb),   arg( @ S : box(Phi,P)    , @ S : dia(nom(T),[L])),
                               res[  add([@ nom(T) : Res],         []))                ):-  (number(L) -> selectnum(L, P, R) ; select(L,P,R)),
                                                                                                      ((R == []) -> (Phi \= nom(T), Res = Phi) ; (Res = box(Phi, R))).
rule( sig(r, 'R dia-n', sd),  arg( @ S : dia(nom(T),[L]), @ S : dia(Phi,P) ),
                              res(  add([],                          [@ nom(T) : Res]))       ):-  (number(L) -> selectnum(L, P, R) ; select(L,P,R)),
                                                                                                      ((R == []) -> (Res = Phi) ; (Res = dia(Phi, R))).


%%% infinitary rules
rule( sig(l, 'L inf-b', ibd),  arg( @ S : forall(box(Phi, P2),[P])    , @ S : dia(_T, L) ),
                               res( add([@ S: Res ],         []))                              ) :- member(P, P2), inst(P, L, box(Phi, P2), Res ).
rule( sig(l, 'L inf-d', idb),  arg( @ S : forall(dia(_T, P2),[P])     , @ S : box(Phi,L) ),
                               res( add([@ S: Res ],         []))                              ) :- member(P, P2), inst(P, L, dia(Phi, P2), Res ).

rule( sig(r, 'R ind',    i),  arg( @ S : forall(Phi, [K]) ),
                              res( split(add([],                  [@ S: Phi0]),
                                         add([@ S: Phi ],         [@ S: PhiKi])) )            ) :- inst(K, [0],  Phi, Phi0), inst(K, [1,K],  Phi, PhiKl) .

rule( sig(l, 'L inf-and',i), arg( @ S : forall(Phi and Psi, K) ),
                             res( add([@ S : forall(Phi, K), @ S : forall(Psi, K)], []))).
rule( sig(r, 'R inf-and',i), arg( @ S : forall(Phi and Psi, K) ),
                             res( split(add([], [@ S : forall(Phi, K)]),
                                        add([], [@ S : forall(Psi, K)])))).

rule( sig(l, 'L inf-agree',i), arg( @ _ : forall(@ T : Phi, K) ),
                             res( add([@ T : forall(Phi, K)], []))).
rule( sig(r, 'R inf-agree',i), arg( @ _ : forall(@ T : Phi, K) ),
                             res( add([], [@ T : forall(Phi, K)]))).

rule( sig(l, 'L inf-dual',i), arg( @ S : forall(not Phi, K). ),
                              res( add([@ S : forall(NotPhi, K)], []))                 ):- (Phi = box(F,P) -> NotPhi=dia(not F, P));
                                                                                             (Phi = dia(F,P) -> NotPhi=box(not F, P)).

rule( sig(r, 'R inf-dual',i), arg( @ S : forall(not Phi, K) ),
                             res( add([], [@ S : forall(NotPhi, K)]))                  ):- (Phi = box(F,P) -> NotPhi=dia(not F, P));
                                                                                             (Phi = dia(F,P) -> NotPhi=box(not F, P)).


%%% parameter simplification
rule( sig(l, 'L b-simp', b), arg( @ S : box(A, [0])),
                             res( add( [@ S : A], []))).
rule( sig(l, 'L d-simp', d), arg( @ S : dia(A, [0])),
                             res( add( [@ S : A], []))).


rule( sig(l, 'L f-simp', i), arg( @ S : forall(A, P)),
                             res( add( [@ S :  ResA], []) )):- parameters(A, APars), ord_intersection(P, APars, ResPars),
                                                               ((ResPars=[N|Rst], number(N)) -> ResPars2=Rst; ResPars2=ResPars),
                                                               (ResPars2 = [] -> ResA = A ; ResA = forall(A, ResPars2)) , ResA \= forall(A, P).
rule( sig(r, 'R b-simp', b), arg( @ S : box(A, [0])),
                             res( add( [@ S : A], []))).
rule( sig(r, 'R d-simp', d), arg( @ S : dia(A, [0])),
                             res( add( [@ S : A], []))).


rule( sig(r, 'R f-simp', i), arg( @ S : forall(A, P)),
                             res( add( [@ S :  ResA], []) )):- parameters(A, APars), ord_intersection(P, APars, ResPars),
                                                               ((ResPars=[N|Rst], number(N)) -> ResPars2=Rst; ResPars2=ResPars),
                                                               (ResPars2 = [] -> ResA = A ; ResA = forall(A, ResPars2)) , ResA \= forall(A, P).


%%% for effortless rule selection
:- dynamic pickrule/1.
:- forall( ( X =..[rule, A, _, _], clause(X, _)), assert( pickrule(A))).
```

C.6 hyloprover.pl

```
:- ensure_loaded('input.pl').
:- ensure_loaded('print.pl').
:- ensure_loaded('misc.pl').
:- ensure_loaded('rulebase.pl').
:- ensure_loaded('modelchecker.pl').

%%% maximum search depth
:-dynamic maxdepth/1.
setmaxdepth(M):- abolish(maxdepth,1), assert(maxdepth(M)).
:-setmaxdepth(15).


%%% selectarg(+Node, +Sig, -Node2, -Arg) : select/extract arguments from the node
selectarg(node(left(S, C, B, I), Right), sig(l,_,s),
          node(left(S2, C, B, I), Right), arg(Phi) ):-select(Phi, S, S2).
selectarg(node(Left, right(S, C, D, I)), sig(r,_,s),
          node(Left, right(S2, C, D, I)), arg(Phi) ):-select(Phi, S, S2).
selectarg(node(left(S1, C1, B, I1), right(S2, C2, D, I2)), sig(lr,_,aa),
          node(left(S1, C1, B, I1), right(S2, C2, D, I2)), arg(Phi)):-ord_intersection(S1,S2,[Phi|_]) ; ord_intersection(C1,C2,[Phi|_]);
                                                                        ord_intersection(S1,D, [Phi|_]) ; ord_intersection(B, S2,[Phi|_]);
                                                                        ord_intersection(C1,D, [Phi|_]) ; ord_intersection(B, C2,[Phi|_]);
                                                                        ord_intersection(I1, I2,[Phi|_]).
selectarg(node(left(S, C, B, I), Right), sig(l,_,c),
          node(left(S, C2, B, I), Right), arg(Phi) ):-select(Phi, C, C2).
selectarg(node(Left, right(S, C, D, I)), sig(r,_,c),
          node(Left, right(S, C2, D, I)), arg(Phi) ):-select(Phi, C, C2).

selectarg(node(left(S, C, B, I), Right), sig(l,_,i),
          node(left(S, C, B, I2), Right), arg(Phi) ):-select(Phi, I, I2).
selectarg(node(Left, right(S, C, D, I)), sig(r,_,i),
          node(Left, right(S, C, D, I2)), arg(Phi) ):-select(Phi, I, I2).

selectarg(node(left(S, C, B, I), Right), sig(l,_,b),
          node(left(S, C, B2, I), Right), arg(Phi) ):-select(Phi, B, B2).
selectarg(node(left(S, C, B, I), Right), sig(l,_,d),
          node(left(S2, C, B, I), Right), arg(Phi) ):-select(Phi, S, S2).
selectarg(node(Left, right(S, C, D, I)), sig(r,_,b),
          node(Left, right(S2, C, D, I)), arg(Phi) ):-select(Phi, S, S2).
selectarg(node(Left, right(S, C, D, I)), sig(r,_,d),
          node(Left, right(S, C, D2, I)), arg(Phi) ):-select(Phi, D, D2).

selectarg(node(left(S, C, B, I), Right), sig(l,_,ibd),
          node(left(S, C, B, I), Right), arg(Arg1, Arg2)):-member(Arg1, I), member(Arg2, S).
selectarg(node(left(S, C, B, I), Right), sig(l,_,idb),
          node(left(S, C, B, I), Right), arg(Arg2, Arg1)):-member(Arg1, I), member(Arg2, B).

selectarg(node(Left, Right), sig(l,_,sb),
          node(Left, Right), arg(Arg2,Arg )):-selectS(Left, Arg), selectBD(Left, Arg2).
selectarg(node(Left, Right), sig(r,_,sd),
          node(Left, Right), arg(Arg, Arg2)):-selectS(Left, Arg), selectBD(Right, Arg2).

selectS(left(S, _, _, _), Arg):- member(Arg, S).
selectBD(HalfNode, Arg):- HalfNode=..[_, _, _, BD, _], member(Arg, BD).


%%% start the prover process
start(Sequent):-standardize(Sequent, Sq), prepare(Sq, Sq2), sortout(Sq2, Node), run(Node, Result, Type, 1),
                 ((Type=maxdepth, writeln('maximum depth reached, no proof found'));
                  (Type=countermodel, writeln('Counter-model:'), writeln(Result));
                  (Type=tableau, writeln(Result), printpdf(Result))).

%%% run the iterative deepening search process
run(Node, Result, Type, N):- process(Node, Result, Type, N).
run(Node, Result, Type, N):- maxdepth(M), N < M, Nplus is N + 1, run(Node, Result, Type, Nplus).

prepare(L ::- R, L2 ::- R2) :- prepare(L, L2), prepare(R, R2).
prepare([],[]).
prepare([Phi|Rest], [Phi2|Rest2]):- (Phi = @ nom(_) : _ -> Phi2=Phi ; Phi2 = @ nom('*') : Phi), prepare(Rest,Rest2).


sortout(L ::- R, node(Left, Right)):- sortoutleft(L, Left), sortoutright(R, Right).
sortoutleft([], left([],[],[],[])).
sortoutleft([Phi| Rest], left(S,C,B,I) ):- sortoutleft(Rest, left(S2,C2,B2,I2) ),
                                          ((simple1(Phi), !, ord_add_element(S2, Phi, S), C=C2, B=B2, I=I2);
                                          ( infset(Phi) , !, S=S2, C=C2, B=B2, ord_add_element(I2, Phi, I));
```

```
                                        ( box(Phi) , !, S=S2, C=C2, ord_add_element(B2, Phi, B), I=I2);
                                        ( S=S2, ord_add_element(C2, Phi, C), B=B2, I=I2)).

sortoutright([], right([],[],[],[])).
sortoutright([Phi| Rest], right(S,C,B,I) ):- sortoutright(Rest, right(S2,C2,B2,I2) ),
                                           ((simpler(Phi), !, ord_add_element(S2, Phi, S), C=C2, B=B2, I=I2);
                                           ( infset(Phi) , !, S=S2, C=C2, B=B2, ord_add_element(I2, Phi, I));
                                           ( dia(Phi) , !, S=S2, C=C2, ord_add_element(B2, Phi, B), I=I2);
                                           ( S=S2, ord_add_element(C2, Phi, C), B=B2, I=I2)).

simplel(@ nom(_| : Phi)):-atomic(Phi) ; Phi = nom(_) ; Phi = dia( nom(_) ,[_]).
simpler(@ nom(_| : Phi)):-atomic(Phi) ; Phi = nom(_). %% ; Phi = box( nom(_) ,[_]).

box(@ _ : box(_, _)).
dia(@ _ : dia(_, _)).
infset(@ _ : forall(_,_)).


%%% Stop trying to prove when we reach maximum depth
process(_, [], maxdepth, N):- maxdepth(N),!.
%%% Stop trying to prove when we find a counter-model
process(Node, M, countermodel, _N):- ground(Node), check(Node), findall(X, ((X=world(I),world(I)); (X=link(A,B,P), link(A,B,P))) , M), !.
%%% Otherwise
process(Node, Result, Type, N):- pickrule(Sig), selectarg(Node, Sig, Node2, Arg), rule(Sig, Arg, res(RuleRes)),
                                 (RuleRes = close
                                  -> closetree(Sig, Arg, Result, Type)
                                   ; (processrule(Node2, RuleRes, PartResult, Type, N), (Type=tableau
                                                                                        -> cleanup(Node, PartResult, Sig, Arg, RuleRes, Result)
                                                                                         ; Result=PartResult))).

closetree(sig(l,Nm,_), arg(P),  [[P|::-[ ], [Nm]], tableau).
closetree(sig(r,Nm,_), arg(P),  [[ |::-[P], [Nm]], tableau).
closetree(sig(lr,Nm,_), arg(P), [[P|::-[P], [Nm]], tableau).


processrule(Node, split(Br1, Br2), Result, Type, N):- processrule(Node, Br1 , Part1Result, Type1, N),
                                                      (Type1=tableau
                                                       -> (processrule(Node, Br2 , Part2Result, Type, N),
                                                           (Type=tableau
                                                           -> append(Part1Result, Part2Result, Result)
                                                            ; Result=Part2Result))
                                                        ; (Type=Type1, Result=Part1Result)).
processrule(Node, add(Lft,Rgt) , [Result], Type, N):- N > 0, Nmin is N - 1, sortout(Lft ::- Rgt, Node2), mergenodes(Node, Node2, NNode),
                                                      process(NNode, Result, Type, Nmin).
processrule(Node, replace(S,T) , [Result], Type, N):- N > 0, Nmin is N - 1, replace(S,T, Node,NNode),
                                                        process(NNode, Result, Type, Nmin).

mergenodes(node(left(Ls1,Lc1,Lb1,Li1), right(Rs1,Rc1,Rd1,Ri1)),
           node(left(Ls2,Lc2,Lb2,Li2), right(Rs2,Rc2,Rd2,Ri2)),
           node(left(Ls3,Lc3,Lb3,Li3), right(Rs3,Rc3,Rd3,Ri3))):- ord_union(Ls1,Ls2,Ls3), ord_union(Lc1,Lc2,Lc3), ord_union(Lb1,Lb2,Lb3), ord_union(Li1,Li2,Li3),
                                                                  ord_union(Rs1,Rs2,Rs3), ord_union(Rc1,Rc2,Rc3), ord_union(Rd1,Rd2,Rd3), ord_union(Ri1,Ri2,Ri3).


%%% cleanup the currnet node in the proof tree (assuming child trees are already cleaned)
cleanup(node(left(S,C,B,I), right(S2,C2,D,I2)),
        PartResult, sig(_, Name,_), Arg, RuleRes, EndResult):- ord_union(S,C, L1), ord_union(L1,B, L2), ord_union(L2,I, L3),
                                                               ord_union(S2,C2, R1), ord_union(R1,D, R2), ord_union(R2,I2, R3),
                                                               ( (PartResult=[[Le::-Ri|_], [Le2::-Ri2|_]],
                                                                  ord_union(Le, Ri, F1),  ord_union(Le2, Ri2, F2),  ord_union(F1, F2, F) );
                                                                 (PartResult=[[Le::-Ri]_]], ord_union(Le, Ri, F)     ),
                                                               ( (Arg=arg(A), ord_add_element(F,A, Fi) ) ;
                                                                 (Arg=arg(A,A2), ord_add_element(F,A,F2), ord_add_element(F2,A2,Fi))),
                                                                cleanup2(L3 ::- R3, Fi, [Name|PartResult], RuleRes, EndResult).


cleanup2(L ::- R, Fi, PartResult, RuleRes, [L2 ::- R2 , PartResult]):- RuleRes = replace(S,T)
                                                                         -> (findall(F, (member(F,L),(member(F,Fi);(replace(S,T,F,F2), member(F2,Fi)))), L3), sort(L3,L2),
                                                                             findall(P, (member(P,R),(member(P,Fi);(replace(S,T,P,P2), member(P2,Fi)))), R3), sort(R3,R2))
                                                                          ; (ord_intersection(L,Fi,L2), ord_intersection(R,Fi,R2)).
```

Overview of references

[1] Carlos Areces, Patrick Blackburn, and Maarten Marx. Hybrid logic: Characterization, interpolation and complexity. *Journal of Symbolic Logic*, 66(3):977–1010, 2001.
website: `http://www.loria.fr/~blackbur/pub.html`
doc: `http://www.loria.fr/~blackbur/papers/charac.pdf`

[2] Carlos Areces and Juan Heguiabehere. HyloRes: Direct resolution for hybrid logics. In C. Areces and M. de Rijke, editors, *Proceedings of Methods for Modalities 2*, Amsterdam, The Netherlands, November 2001.
website: `http://www.loria.fr/~areces/content/papers/sort_date.php`
doc: `http://www.loria.fr/~areces/content/papers/files/m4m02.pdf`

[3] Patrick Blackburn. Modal logic as dialogical logic. *Synthese*, 127:57–93, 2001.
website: `http://www.loria.fr/~blackbur/pub.html`
doc: `http://www.loria.fr/~blackbur/papers/synthese.pdf`

[4] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*, chapter Hybrid logic, pages 434–445. Cambridge University Press, 2002.
website: `http://www.mlbook.org/`

[5] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*, chapter 3.6 Sahlqvist formulas, 3.7 More about Sahlqvist formulas, pages 156–178. Cambridge University Press, 2002.
website: `http://www.mlbook.org/`

[6] Patrick Blackburn and Maarten Marx. Tableaux for quantified hybrid logic. In U. Egly and C. Fernmüller, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 38–52. International Conference, TABLEAUX 2002, 2002.
website: `http://www.loria.fr/~blackbur/pub.html`
doc: `http://www.loria.fr/~blackbur/papers/tableaux02.pdf`

[7] Patrick Blackburn and Johan van Benthem. Modal logic: A semantic perspective. In *Handbook of Modal Logic*. Elsevier Science, 2006.
website: `http://staff.science.uva.nl/~johan/seminar.html`
doc: `http://staff.science.uva.nl/~johan/hml-blackburnvanbenthem.pdf`

[8] Robert S. Boyer and J. Strother Moore. *A Computational Logic*. ACM monograph series, ISSN 0572-4252. New York [etc.] : Academic Press, 1979.

[9] Ricardo Caferra, Alexander Leitsch, and Nicolas Peltier. *Automated Model Building*, chapter Automated Model Building (Ch.1 Introduction), page 334. Applied Logic Series , Vol. 31. Kluwer, 2004.
website: `http://www.logic.at/people/leitsch/`
doc: `http://www.logic.at/people/leitsch/mbbook.pdf`

[10] L.A. Chagrova. An undecidable problem in correspondence theory. *Journal of Symbolic Logic*, 56:1261–1272, 1991.

[11] Barteld Kooi, Gerard Renardel de Lavalette, and Rineke Verbrugge. Hybrid logics with infinitary proof systems. *Journal of Logic Computation*, 16(2):161–175, 2006.
website: `http://logcom.oxfordjournals.org/cgi/content/abstract/16/2/161-a`
doc: `http://logcom.oxfordjournals.org/cgi/reprint/16/2/161-a.pdf`

[12] Marcus Kracht. *Diamonds and Defaults*, volume 229 of *Synthese library*, chapter How completeness and correspondence got married, pages 175–214. Kluwer Academic Publishers, 1993.

[13] Edward John Lemmon and Dana S. Scott. *The 'Lemmon notes' : An Introduction to Modal Logic*. Oxford : Blackwell, 1977.

[14] Lawrence C. Paulson. Designing a theorem prover. In Abramsky & Gabbay & Maibaum (Eds.), editor, *Handbook of Logic in Computer Science*, volume 2, pages 415–475. Clarendon, 1992.
website: `https://www.publications.cl.cam.ac.uk/116/`
doc: `https://www.publications.cl.cam.ac.uk/116/01/TR192-lcp-designing.pdf`

[15] A. Prior. *Past, Present and Future*. Oxford University Press, 1967.

[16] Henrik Sahlqvist. Completeness and correspondence in the first and second order semantics for modal logic. In Stig Kanger, editor, *Proceedings of the Third Scandinavian Logic Symposium*, volume 82 of *Studies in Logic and the Foundations of Mathematics*, pages 110–143, Amsterdam, Oxford, 1975. North Holland Publishing Co. [etc.].

[17] Balder ten Cate. *Model theory for extended modal languages*, chapter Introduction to hybrid languages, pages 37–44. ILLC Dissertation Series. Institute for Logic, Language and Computation, 2005.
website: `http://www.illc.uva.nl/Publications/reportlist.php?Series=DS`
doc: `http://www.illc.uva.nl/Publications/Dissertations/DS-2005-01.text.pdf`

[18] Balder ten Cate, Maarten Marx, and Petrúcio Viana. Hybrid logic with Sahlqvist axioms. *Logic Journal of IGPL*, 13(3):293–300, 2005.
website: `http://jigpal.oxfordjournals.org/cgi/content/abstract/13/3/293`
doc: `http://jigpal.oxfordjournals.org/cgi/reprint/13/3/293.pdf`

[19] Edward N. Zalta. *Basic Concepts in Modal Logic.* web publication, 1995.
website: `http://mally.stanford.edu/publications.html`
doc: `http://mally.stanford.edu/notes.pdf`

"It's a dark and stormy summer night. In his lab we find Victor Frankenstein hard at work, hacking, cutting and dissecting, sewing the parts together in novel arrangements. It is the moment of his greatest creation. But there is no lightning involved here, somewhere events had taken an awfully wrong turn in this alternate universe. This Frankenstein is no chemist obsessed with the creation of life, but a logician. His creation: a logic composed of borrowed parts: Hybrid Logic.

And thus begins our harrowing tale."

university of
groningen