Hybrid Logic

Asta Halkjær From

Kongens Lyngby 2020

Updated 5 February 2020 with text and layout corrections.

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

I formalize a Seligman-style tableau system for basic hybrid logic in the proof assistant Isabelle/HOL. Unlike the original system, I name every block on the tableau to avoid the risk of needing to backtrack when constructing a tableau. The tableau rules are restricted to only allow extensions of the tableau that include a new formula and I show via a strengthening lemma that this restriction preserves completeness. Showing strengthening under this restriction requires relaxing the Nom rule to allow the shared nominal to occur on any block, not just the current one as originally done. The ($\Diamond$) rule is restricted to only apply to unwitnessed $\Diamond$-formulas and this restriction is lifted by proving a substitution lemma for the calculus. General versions of the @-rules are derived from their restricted counterparts using a structural lemma that allows weakening, factoring and reordering at the level of blocks and a similar lemma that works inside the current block. These lemmas show that forward referencing in the tableau is admissible as long as it is acyclic. The GoTo rule is restricted using a notion of coins where each application of GoTo requires spending a coin and coins are earned through applications of the remaining rules. If a branch can be closed then it can be closed starting from a single coin. This restriction of GoTo works better with rule induction than the simpler scheme of disallowing two applications in a row. I show that the Bridge rule is admissible using a set of indices into the branch that satisfies a particular descendant relation. This set effectively guides what formula occurrences should be modified when one accessibility formula is replaced by another. Next, I apply an existing synthetic completeness proof based on maximally consistent sets of blocks to show that the negation of any valid formula has a closing tableau. Using this approach for the restricted system is novel. Finally, in pursuit of a terminating system, I suggest a restriction of the Nom rule based on tags that directly encodes the notion of copying in the right direction and sketch a strategy for lifting this restriction.

Preface

This thesis was prepared at DTU Compute in fulfillment of the requirements for acquiring a MSc in Engineering (Computer Science and Engineering). The thesis was prepared from 19 August 2019 to 19 January 2020 amounting to 30 ECTS.

I have no previous experience with the tableau system used in this thesis but have previously looked at Braüner's treatment of false-belief tasks using a natural deduction system for hybrid logic [Bra13]. This was for an assignment in Nina Gierasimczuk's course Logical Theories for Uncertainty and Learning.

I have previous experience with Isabelle/HOL, including published formalizations of soundness and completeness of System K for epistemic logic [Fro18] and of a one-sided sequent calculus for first-order logic [Fro19].

Supervisors

- Jørgen Villadsen
- Alexander Birch Jensen (co-supervisor)
- Patrick Blackburn (external supervisor, Roskilde University)

Roskilde, 19 January 2020,
Andreas "Asta" Halkjær From

Acknowledgments

I am grateful to my supervisor Jørgen Villadsen for suggesting a thesis on hybrid logic and for helping me through the five-month process of completing it. Jørgen is excellent at letting you work at your own pace while also keeping an eye on the overall schedule. Jørgen introduced me to Isabelle/HOL when I started at DTU and I would not be where I am today without his guidance since then.

I want to thank my external supervisor Patrick Blackburn for sharing his knowledge of hybrid logic with me as well as patiently listening to my troubles formalizing this or that lemma. Being supervised by a co-author of the system you are working with is a privilege and in my case it also occasioned some pleasant bike rides to Roskilde University (RUC).

Thanks to co-supervisor Alexander Birch Jensen for regular meetings and for comments on the thesis.

I also want to thank Klaus Frovin Jørgensen and Thomas Bolander for discussions about a preliminary version of the formalization and for sharing some insight into the development of the system.

Similarly, thank you to Jerry Seligman for a chat about the work at a Prior Studies event at RUC.

Finally, thank you to my family and friends for patiently listening to me ramble about hybrid logic one day and grumble about some flawed proof idea the next.

Contents

CHAPTER 1

Introduction

This thesis focuses on a proof system for basic hybrid logic [Bra17] and formalizes that proof system in Isabelle/HOL [NPW02]. In the first half of this chapter I introduce the overall ideas of hybrid logic and motivate the focus on it. In the second half I will argue for the utility of a proof assistant like Isabelle/HOL when working with a proof system and detail some of the benefits of formalizing work like this.

1.1 Hybrid Logic

Logic is the study of reasoning and in this thesis especially the study of which statements can be considered correct. Propositional logic treats statements as simply true or false and allows us to build larger statements with connectives like *not*, *and*, *or*, etc.

Modal logic takes this a step further and considers the truth value of a statement relative to a specific point, like a possible world, a time, a location or something similar. Thus a statement like "it rains" is not just considered true or false, but possibly true at one point and false at another. Given a relation between points, this extra generality makes modalities like *possibly* and *necessarily* meaningful:

We can express that it "*necessarily* rains" if it rains at all points relative to the considered one, for instance in all neighboring cities. Or we may consider the transitions between states taken by a computer program and express that it is "*not possible* to get stuck" if, regardless of the starting state, there is no reachable state where the program cannot progress.

Hybrid logic arises with the idea of allowing statements that name these points using so-called nominals instead of only making opaque references to them through the modalities [Bra17]. As such we may form a statement like "it is Christmas Eve *and* it is snowing" where "it is Christmas Eve" is a nominal and "it is snowing" is a regular proposition. If we wish to insist on considering something at a certain point we can use the satisfaction operator, *at*, and say for instance that "*at* Christmas Eve, it is snowing" [Bra17]. Further additions are possible but these are beyond the scope of this thesis.

There is a lot more to say about basic hybrid logic and especially its connection to first-order logic, but let us instead focus on some applications. Blackburn points out that, more so than standard modal logic, hybrid logic is well-suited for representing temporal logic which can be used for expressing properties about software and hardware systems, enabling formal verification of their correctness [Bla00]. Another application of hybrid logic is representing feature logics [Bla00], a class of logics for classifying and constraining feature structures that appear in both linguistics, the theory of data structures and that of databases [Rou97]. Finally, Blackburn points out that description logic, used in artificial intelligence, as logical formalism for the Web Ontology Language for the semantic web and in biomedical informatics [HGS07], can be seen as a hybrid logic [Bla00]. In a more philosophical direction, Braüner uses hybrid logic to reason about false-belief tasks with a proof system similar to the one considered in this thesis [Bra13].

1.2 Isabelle/HOL

This thesis contains a lot of definitions and proofs but definitions can be misinterpreted and proofs can be incorrect. To mitigate this, all the work, unless otherwise noted, has been formalized in the proof assistant Isabelle/HOL.

Isabelle is a generic proof assistant and Isabelle/HOL is its instance based on higher-order logic [NPW02]; I will use the two terms interchangeably. A proof assistant like Isabelle provides the means to express mathematical statements and proofs in a formal language that can be mechanically verified. In other words, definitions become unambiguous and every step of a proof is checked to ensure that it follows from the steps before it. In the case of Isabelle which uses the LCF architecture, this process works by compiling every statement down to a minimal language of axioms and inference rules that is checked by a trusted kernel [NPW02]. These axioms and inference rules belong to the meta-logic whereas the formalized logic is known as the object logic. The artifact obtained by working in a proof assistant is known as a *formalization*.

With a formalized proof system the computer can help us verify that any proof within the system is correct, i.e. that rules are applied correctly, side conditions are met and so on: We obtain a method of semi-automatic proof checking simply through the specification. Moreover, we can now formally verify properties about the proof system like soundness, completeness or the admissibility of a particular rule. Readers of the formalization can focus on the definitions, lemma statements and overall ideas, trusting that the proofs are correct since they have been verified by the machine. And if the reader decides to delve into the proofs they will find every case covered and every detail attended to, at least down to a level where the automatic proof search of the proof assistant can take over. In a formalized proof there are no exercises left for the reader, no cases that should follow analogously to others but turn out not to and no proofs have been omitted because they are deemed trivial. This means that formalizing a proof is a significant effort but increases the trust we can put in the result. The level of detail also means that studying a formalization can be very educational. For readability, most of the lemmas in this thesis have been written in standard mathematical syntax.

1.3 Archive of Formal Proofs

The full formalization is available in the Archive of Formal Proofs which collects refereed submissions and keeps them up to date with the current Isabelle version:

`https://www.isa-afp.org/entries/Hybrid_Logic.html`

The latest version of the formalization (4820 lines) is available at the development site where compatibility is only guaranteed for a repository or snapshot version of Isabelle so it may not work on the latest stable version:

`https://devel.isa-afp.org/browser_info/current/AFP/Hybrid_Logic/index.html`

The page above includes four links. The first, *theory_dependencies*, shows the transitive dependencies of the theory. The second, *document*, links to a PDF obtained by rendering the Isabelle proofs in LaTeX (100+ pages). The third, *outline*, is an abridged version of the full document that only includes lemma statements, not proofs. Finally, *Hybrid_Logic* links to the full formalization rendered in the browser with syntax highlighting.

1.4 Chapter Overview

In Chapter 2 I formalize the syntax and semantics of basic hybrid logic. In Chapter 3 I introduce the tableau system considered in the rest of the thesis along with a number of restrictions imposed to rule out sources of nontermination. Chapter 4 proves a number of results about the system, including strengthening, substitution and a structural property, and uses these to lift the termination restrictions. Working with the unrestricted rules, Chapter 5 shows that the Bridge rule is admissible using the notion of a descendant relation on a set of indices into a tableau branch. In Chapter 6 I formalize the completeness of the proof system using the synthetic approach by Jørgensen et al. [JBBB16]. Finally Chapter 7 concludes the thesis by sketching a possible restriction on the so-called Nom rule and by pointing at related and future work.

Chapter 2

Syntax and Semantics

In this chapter I introduce the syntax and semantics of basic hybrid logic and show how it can be formalized in Isabelle. I choose a deep embedding of the logic where the syntax is represented as a datatype in the meta-logic and the semantics is modeled as a function on this datatype. As Wildmoser and Nipkow point out, this requires more work up front than a shallow embedding where object-logic formulas are translated directly into the meta-logic. For instance, with the deep embedding we are required to write explicit substitution functions [WN04]. In return, the deep embedding allows us to prove theorems by induction over the structure of formulas and this will be useful for formalizing the completeness theorem by Jørgensen et al. [JBBB16].

2.1 Syntax

The syntax is parameterized over two universes, one of propositional symbols and one of nominals. Propositional symbols occur rarely and I write them as x. The more frequent nominals are written a, b, c, i, j or k. The well-formed formulas are given by the following grammar:

$$\phi, \psi ::= x \mid i \mid \neg\phi \mid \phi \vee \psi \mid \Diamond\phi \mid @_i\phi$$

A formula $\neg\phi$ is read as *not* ϕ, $\vee$ is meant to symbolize the connective *or* and $\Diamond$ is the modality *possibly*. The satisfaction operator is now written @ and formulas of the form $@_i\phi$ are called satisfaction statements.

The datatype in Figure 2.1 encodes the syntax in Isabelle. It is parameterized over two types represented by the type variables *'a* and *'b*. The first stands for propositional symbols and the second for nominals. The parameterization gives us the freedom to use whatever types we want for these symbols and also to only assume an infinite universe of nominals when necessary. The default syntax is given by the constructors, *Pro*, *Nom*, etc. but infix syntax can be specified to the right in parentheses. This allows us to use our usual syntax to write formulas in the proof assistant, albeit in bold to not conflict with the built-in syntax. Note that I will use p, q etc. for formulas in Isabelle since they are easier to type than ϕ and ψ.

```
datatype ('a, 'b) fm
  = Pro 'a
  | Nom 'b
  | Neg ‹('a, 'b) fm› (‹¬ -› [40] 40)
  | Dis ‹('a, 'b) fm› ‹('a, 'b) fm› (infixr ‹∨› 30)
  | Dia ‹('a, 'b) fm› (‹◇ -› 10)
  | Sat 'b ‹('a, 'b) fm› (‹@ - -› 10)
```

Figure 2.1: Hybrid logic syntax as a datatype in Isabelle/HOL.

2.2 Semantics

The language is interpreted on models consisting of a frame (W, R) and a valuation of propositional symbols V. The first part of the frame, W, is a non-empty set of points/worlds and the second part, R, is an accessibility relation between them. Models are represented as concrete datatypes in the formalization:

```
datatype ('w, 'a) model = Model (R: ‹'w ⇒ 'w set›) (V: ‹'w ⇒ 'a ⇒ bool›)
```

In the above, I use the type variable *'w* to represent W, making use of the fact that types in HOL are non-empty. The accessibility relation, R, is then a function from a world to a set of reachable worlds and the valuation, V, is a predicate on a world and a propositional symbol. Given a model M, we can write $R\ M$ and $V\ M$ to access the reachability relation and valuation, respectively.

An *assignment* g for a model whose worlds are W is a function from a nominal to an element of W. That is, the assignment tells us which semantic world each syntactic nominal denotes. For brevity I will sometimes describe nominals as worlds, implicitly referring to the output of some assignment.

Figure 2.2 gives the basic hybrid logic semantics as a HOL predicate. The first three lines give the name, type signature and infix syntax of the predicate. As evident by the type signature it takes four arguments: A model, an assignment, a world and a formula, where each argument lines up type-wise with the rest.

```
primrec semantics
  :: ‹('w, 'a) model ⇒ ('b ⇒ 'w) ⇒ 'w ⇒ ('a, 'b) fm ⇒ bool›
  (‹-, -, - ⊨ -› [50, 50, 50] 50) where
  ‹(M, -, w ⊨ Pro x) = V M w x›
| ‹(-, g, w ⊨ Nom i) = (w = g i)›
| ‹(M, g, w ⊨ ¬ p) = (¬ M, g, w ⊨ p)›
| ‹(M, g, w ⊨ (p ∨ q)) = ((M, g, w ⊨ p) ∨ (M, g, w ⊨ q))›
| ‹(M, g, w ⊨ ◇ p) = (∃v ∈ R M w. M, g, v ⊨ p)›
| ‹(M, g, - ⊨ @ i p) = (M, g, g i ⊨ p)›
```

Figure 2.2: Semantics of hybrid logic as a predicate in Isabelle/HOL.

The following six lines list each of the cases for the structure of a formula. Three cases are worth mentioning: nominals, diamonds and satisfaction statements. First, a nominal i is only true in the world it denotes. Second, $\Diamond\phi$ is true at world w if there exists a world v reachable from w that models ϕ. Finally, a satisfaction statement, $@_i\phi$, shifts perspective, causing the truth of ϕ to be determined at the world denoted by i.

A formula is valid if it is true in all models, assignments and worlds and satisfiable if there exists some combination of the three in which it is true.

Chapter 3

Proof System

We are interested in a syntactic means of showing the validity of hybrid logic formulas. Blackburn et al. provide exactly such a means with their Seligman-style tableau calculus ST [BBBJ17]. In this chapter I introduce a variant of ST along with termination restrictions equivalent to their restrictions R1-R5.

Blackburn et al. give an example of nontermination due to the Nom rule which they resolve by splitting it into three parts [BBBJ17]. The example applies to the presented system as well. Instead of splitting the rule, I propose in Section 7.1 on page 55 a restriction on the applicability of Nom. There, I also sketch a proof of why the restriction preserves completeness but work on this restriction is not yet formalized and so it is not considered until that chapter.

3.1 Rules

Since ST is a tableau system, we typically start from the negation of the formula we want to prove and apply rules to break down the formula into smaller formulas that follow from it, in search of a contradiction on all branches.

The rules are based on a subdivision of the branches of the tableau into blocks. Each pair of blocks is separated by a horizontal line and the first formula on each block is a nominal dubbed the *opening* nominal. The intuition is that the formulas on a block are true in the world denoted by its opening nominal. We may view a block as a macro that expands into a number of satisfaction statements as illustrated in Figure 3.1. Where satisfaction statements encode global knowledge, the blocks allow a local perspective where we work within one world at a time and then explicitly switch to another one. The opening nominal plays a special role in the system, but is still just a formula and may participate in rules like any other. If Θ is a branch and ϕ occurs on an i-block in Θ then I say that ϕ occurs *at i in* Θ. I occasionally refer to the opening nominal of a block as its name or type.

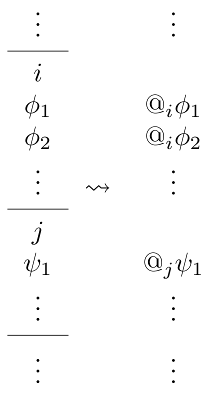


Figure 3.1: Blocks as macros.

The propositional rules given in Figure 3.2 are standard but now work within the blocks. The input to the rule is given above the vertical line and the output below it. Unlike Blackburn et al. I explicitly write the opening nominals in the rules, while still suppressing the rest of the formulas on the block. If the opening nominals match then the output block may be the same as an input block. Consider the $(\neg\neg)$ rule: If $\neg\neg\phi$ occurs on an a-block and the current block is an a-block, then ϕ is a legal extension of the branch.

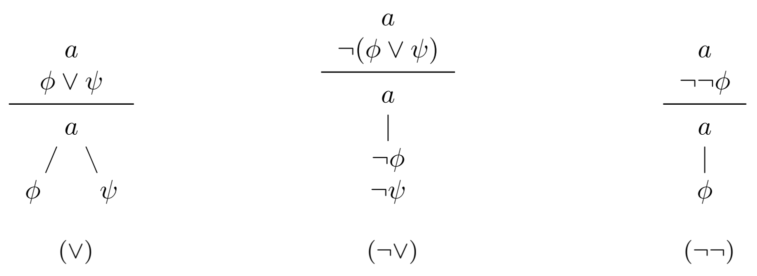
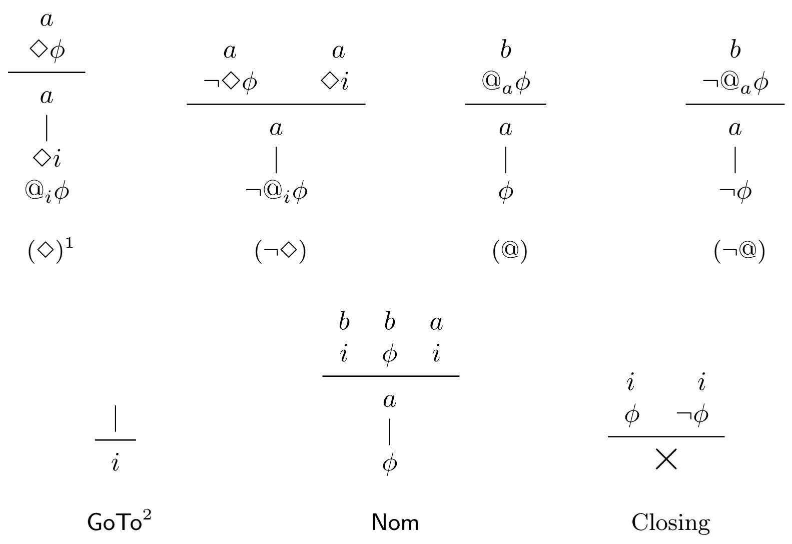


Figure 3.2: Propositional rules.

The remaining rules are given in figure 3.3 on the next page. The first row contains the rules for the hybrid logic connectives and the second row contains rules for working with the blocks. Consider the $(\neg\Diamond)$ rule: If from world a we cannot reach a world where ϕ is satisfied ($\neg\Diamond\phi$), but we can reach world i ($\Diamond i$), then it is safe to conclude that ϕ does not hold at i ($\neg@_i\phi$). The notation makes it explicit that the two premises may belong to separate blocks and that these can appear in any order on the branch.



[1] i is fresh, ϕ is not a nominal.
[2] i is not fresh.

Figure 3.3: Hybrid rules.

The GoTo rule allows us to open a new block whose opening nominal already occurs somewhere on the branch, either on its own or within a formula.

The Nom rule allows us to copy formulas between blocks that denote the same world, as evidenced by the same nominal occurring on blocks of both types. Note that nominal i may itself be either a or b.

A branch closes if the same formula occurs on the same type of block both positively and negatively. Otherwise, if the branch is exhausted but does not close it is called open. A tableau is closed if all its branches are closed and open if one of its branches is. If a closed tableau can be obtained starting from the branch Θ then I write $\vdash \Theta$. In the vocabulary of Jørgensen et al. the blocks of Θ can be thought of as the root blocks [JBBB16]. Later, it will be useful to have an inline syntax for branch extensions. If Θ is a branch and the current block has opening nominal a, then I write the branch obtained by extending Θ with ϕ as $\phi -_a \Theta$. The hyphen is meant to resemble the vertical rule in the figures.

When constructing a tableau it is useful to consider the rules from top to bottom as descriptions of what extensions are legal in the search for a contradiction. However, when proving properties of the system using rule induction, it will be helpful to view them the other way around: As descriptions of when we can remove a formula from a branch and still know that it can be closed.

3.1.1 A way home

The original system ST allows for the first block to be unnamed but includes a Name rule for introducing a fresh nominal to be treated as the opening nominal. Since I require every block to be named by construction, I omit this rule.

The unnamed block is neat when we want to prove a purely propositional formula where hybrid reasoning is unnecessary, but it comes at the cost of having to backtrack if we forgot to name the initial block and subsequently apply the wrong rule. Consider the formula $\neg(i \vee \neg i)$ which has a closing tableau obtained by applying the $(\neg\vee)$ rule. However, as seen in figure 3.4a, if we erroneously start off with a GoTo then we lose the ability to finish the tableau: We get stuck on a branch that is neither saturated nor closes. Figure 3.4b illustrates how the present system ensures that we can return to the world we started in.

$$\begin{array}{lcl}
1. & \neg(i \vee \neg i) & \\
\cline{2-2}
2. & i & \mathsf{GoTo}
\end{array}$$

(a) Original system.

$$\begin{array}{lcl}
1. & a & \\
2. & \neg(i \vee \neg i) & \\
\cline{2-2}
3. & i & \mathsf{GoTo} \\
\cline{2-2}
4. & a & \mathsf{GoTo} \\
5. & \neg i & (\neg\vee)\ 2 \\
6. & \neg\neg i & (\neg\vee)\ 2 \\
 & \times &
\end{array}$$

(b) Present system.

Figure 3.4: Taking a wrong turn.

The possibility of getting stuck also matters when we start from more than a single formula on a single block. We would like to know that if a closing tableau exists for a set of blocks then it also exists for any superset, but if one of the blocks is unnamed then, as just illustrated, this may no longer be the case. We might assume for this lemma that all blocks are named but then we have to carry this assumption transitively, making the formalization more cumbersome.

3.2 Termination Restrictions

Blackburn et al. outline five restrictions imposed on the rules that along with a different Nom rule give a terminating system [BBBJ17]. They are as follows:

R1 A formula is never added to an i-block if it already occurs on an i-block on the same branch.

R2 The ($\Diamond$) rule cannot be applied twice to the same formula occurrence.

R3 The Name rule is only ever applied as the very first rule in a tableau.

R4 The GoTo rule cannot be applied twice in a row.

R5 (@) and ($\neg$@) can only be applied to premises i and $@_i\phi$ ($\neg @_i\phi$) when the current block is an i-block.

For R1 and R2 I will give alternative formulations suitable for formalization and show that the unrestricted versions of the rules can be proved within the system. Restriction R3 does not apply to the present system. In Section 3.3 on page 16 I will show why R4 as stated makes some properties difficult to prove by standard rule induction and give a different termination restriction that is easier to work with. In Section 4.1 on page 25 I show that the reformulation is also easier to address. Restriction R5 is already incorporated structurally in the presented @-rules and I will show how to obtain the general versions.

3.2.1 Reformulations

I will say that ϕ is *new* to a in Θ if ϕ does not occur at a in Θ.

Now consider the original R1 restriction as it applies to the ($\neg\vee$) rule on input $\neg(\phi \vee \psi)$. If $\neg\phi$ is not new to the block type, does that mean that the branch is only extended with $\neg\psi$? If this is the case, can we apply the rule repeatedly with the same input but extend the branch at most once? Clearly not, as then we would not have a terminating system. And outlawing the rule entirely because half of the extension is not new would be a problem for completeness. The following reformulation aims to avoid this confusion:

R1 The output of a rule must include a formula *new* to the current block type.

Consider for instance $(\lor)$: Both subformulas must be new, since they are independently output on the branch, but for something like $(\neg\lor)$ just one of them needs to be. This still prevents repeatedly applying either rule.

Restriction R2 serves the same purpose as R1 but cannot be formulated in the same way since the output includes the fresh nominal i and as such will always be new. Still, we would like to reformulate the rule to appeal only to the contents of the branch, which we already keep track of in the formalization, not any previous rule applications. One possibility, indeed the chosen one, is the following:

R2 The $(\Diamond)$ rule can only be applied to input $\Diamond\phi$ on an a-block if $\Diamond\phi$ is not already *witnessed* at a.

Here, $\Diamond\phi$ is *witnessed* at a in Θ if for some witnessing nominal i, both $\Diamond i$ and $@_i\phi$ occur at a in Θ. If a diamond is already witnessed, we do not need to witness it again with a fresh nominal. This is proven in Section 4.3 on page 31.

3.2.2 Satisfaction statements

As noted, the $(@)$ and $(\neg@)$ rules in figure 3.3 on page 11 are already R5-restricted structurally as this makes rule induction simpler. The unrestricted rules as presented by Blackburn et al. [BBBJ17] can be seen in figure 3.5. I will show how to prove these in Section 4.4 on page 34.

$$\begin{array}{c}\begin{array}{cc} b & a \\ @_i\phi & i \end{array}\\ \hline \begin{array}{c} a \\ | \\ \phi \end{array}\\ \\ (@)\end{array} \qquad\qquad \begin{array}{c}\begin{array}{cc} b & a \\ \neg @_i\phi & i \end{array}\\ \hline \begin{array}{c} a \\ | \\ \neg\phi \end{array}\\ \\ (\neg @)\end{array}$$

Figure 3.5: Unrestricted $(@)$ and $(\neg@)$ rules.

3.2.3 The Nom rule

The original Nom rule requires the shared nominal i to occur on the current block and not just any block with the same opening nominal [JBBB16]. Combined with R1 and R5 this has the curious effect of causing incompleteness if we use the derived rule $(\neg \rightarrow)$. Consider the tableau in figure 3.6. Every step of the tableau is the only one that causes progress; our hand is forced by the formula. When we reach line 16 we have ϕ on an i-block, $\neg\phi$ on a j-block and i and j on each others type of block. Semantically, there is clearly a contradiction. But exactly because we were forced by the $(\neg \rightarrow)$ rule to add j and i to the second and third block respectively, and by R5 to open new blocks afterwards, we are now prevented from closing the branch. We cannot copy ϕ to the current block because it does not share any nominals with the i-block. And due to R1 the current block *cannot* be made to do so, since i occurs on a j-block at line 9.

With the present, relaxed Nom rule, we *are* allowed to copy ϕ since the shared nominal does not have to occur on the current block. As we shall see in Section 4.2 on page 28, this relaxation of the rule will also make it possible to lift the R1 restriction via strengthening. Note that without termination restrictions the relaxed rule can be derived from two applications of the original.

1.	a	
2.	$\neg @_i(j \rightarrow @_j(i \rightarrow @_i(\phi \rightarrow @_j\phi)))$	
3.	i	GoTo
4.	$\neg(j \rightarrow @_j(i \rightarrow @_i(\phi \rightarrow @_j\phi)))$	$(\neg @)$ 2, 3
5.	j	$(\neg \rightarrow)$ 4
6.	$\neg @_j(i \rightarrow @_i(\phi \rightarrow @_j\phi))$	$(\neg \rightarrow)$ 4
7.	j	GoTo
8.	$\neg(i \rightarrow @_i(\phi \rightarrow @_j\phi))$	$(\neg @)$ 6, 7
9.	i	$(\neg \rightarrow)$ 8
10.	$\neg @_i(\phi \rightarrow @_j\phi)$	$(\neg \rightarrow)$ 8
11.	i	GoTo
12.	$\neg(\phi \rightarrow @_j\phi)$	$(\neg @)$ 10, 11
13.	ϕ	$(\neg \rightarrow)$ 12
14.	$\neg @_j\phi$	$(\neg \rightarrow)$ 12
15.	j	GoTo
16.	$\neg\phi$	$(\neg @)$ 14, 15

Figure 3.6: Getting stuck with R1+R5 and $(\neg \rightarrow)$.

Omitting the R1 restriction would make us able to close the branch with the original Nom rule, but, as Figure 3.7 shows, so would unfolding the abbreviation $\phi \rightarrow \psi \equiv \neg\phi \vee \psi$. The $(\neg\vee)$ rule negates its extension, so lines 5 and 9 now contain $\neg\neg j$ and $\neg\neg i$ instead of j and i. In a sense, the double negations delay the knowledge that the two nominals denote the same world and this allows us to wait until the last moment to reveal this information, making Nom applicable.

1.	a	
2.	$\neg @_i(\neg j \vee @_j(\neg i \vee @_i(\neg \phi \vee @_j \phi)))$	
3.	i	GoTo
4.	$\neg(\neg j \vee @_j(\neg i \vee @_i(\neg \phi \vee @_j \phi)))$	$(\neg @)$ 2, 3
5.	$\neg\neg j$	$(\neg\vee)$ 4
6.	$\neg @_j(\neg i \vee @_i(\neg \phi \vee @_j \phi))$	$(\neg\vee)$ 4
7.	j	GoTo
8.	$\neg(\neg i \vee @_i(\neg \phi \vee @_j \phi))$	$(\neg @)$ 6, 7
9.	$\neg\neg i$	$(\neg\vee)$ 8
10.	$\neg @_i(\neg \phi \vee @_j \phi)$	$(\neg\vee)$ 8
11.	i	GoTo
12.	$\neg(\neg \phi \vee @_j \phi)$	$(\neg @)$ 10, 11
13.	$\neg\neg \phi$	$(\neg\vee)$ 12
14.	$\neg @_j \phi$	$(\neg\vee)$ 12
15.	j	GoTo
16.	$\neg \phi$	$(\neg @)$ 14, 15
17.	i	$(\neg\neg)$ 9
18.	$\neg\neg \phi$	Nom 11, 13, 17
	$\times$	

Figure 3.7: Being saved from R1 by double negations.

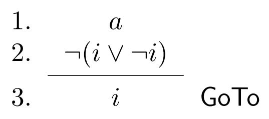
3.3 Restricted GoTo Considered Harmful

The subdivision of a branch into blocks corresponding to different nominals provides us with a local perspective where we can focus on one type of block at a time. It also means that perspective shifts become explicit through the GoTo rule and imposing restrictions on it has a significant impact on the proof system.

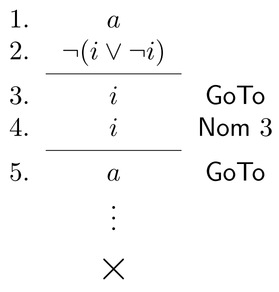
3.3.1 Ramifications

The R4 restriction as stated reintroduces the ability to take a wrong turn in the construction of a tableau and get stuck as a result. This is illustrated in Figure 3.8 on the facing page which also shows that if we ignore R1 we can play a cheap trick with the Nom rule to make progress.

Let us ignore this issue for now and focus on another consequence: We may not be able to close a modified branch by simply following the rule applications from the original one. Consider the property of weakening a branch with an extra



(a) Need to backtrack.



(b) Saved by Nom.

Figure 3.8: Taking a wrong turn again.

formula on some block and still being able to close it. Figure 3.9 illustrates how legal extensions of a branch become scarce resources under R1 and R4. On the left, we could use the $(\neg\neg)$ rule on line 4 to justify the following GoTo. But after the weakening, ϕ is no longer new to a, so the $(\neg\neg)$ rule is not justified and by extension, neither is the GoTo. On the weakened branch, opening the second a-block is a mistake, so under R4, weakening requires possibly rewriting the rest of the branch to avoid detours. Note that if ϕ was the only legal extension at that point then there is no way to recover from this mistake besides backtracking which complicates rule induction.

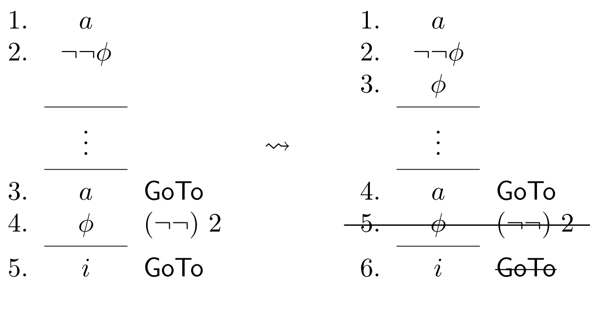


Figure 3.9: Unjustified GoTo after weakening.

Renaming the nominals on a branch may collapse formulas and thus invalidate rule applications in a similar way, causing the same issue. Instead of coupling the elimination of detours with each of these properties, we would like a terminating restriction on GoTo that works with the standard induction rule provided by Isabelle and can be dealt with separately.

3.3.2 Unique blocks

Consider first a radical alternative. The GoTo rule gives us the desirable property that the content of a branch never changes but is only extended with new blocks and formulas. This is an advantage for the branching ($\vee$) rule since it allows us to work on the two branches separately without any risk of interference between them. However, if we were willing to give up that property we could obtain a system with a much simpler GoTo rule: You are only allowed to open a new i-block if there is no i-block already. This formulation of the rule is terminating if we assume that only a finite number of nominals can be generated on a branch (which the remaining restrictions seek to guarantee).

For the rest of the rules, we would drop the notion of a current block and instead fill in whatever existing block matches the rule we are applying. Figure 3.10 illustrates how such a tableau might look like a table, and how it could still be viewed as a macro that expands into a labeled system.

$$\begin{array}{cccc} i & j & k & \cdots \\ \hline \phi_1 & \psi_1 & \rho_1 & \\ \phi_2 & \psi_2 & \rho_2 & \\ & \psi_3 & & \\ \vdots & \vdots & \vdots & \end{array} \quad \rightsquigarrow \quad \begin{array}{c} @_i\phi_1 \\ @_i\phi_2 \\ \vdots \\ @_j\psi_1 \\ @_j\psi_2 \\ @_j\psi_3 \\ \vdots \\ @_k\rho_1 \\ @_k\rho_2 \\ \vdots \end{array}$$

Figure 3.10: At most one block per nominal.

The main disadvantage of this system, as mentioned, is that whenever we apply a branching rule we have to copy the entire table to both branches to avoid interference between them. However, this is more of a problem when drawing it on paper than if implemented on a computer. A different critique is that without the explicit perspective shift, this is nothing more than different notation for a labeled system.

3.3.3 Making progress

Instead, consider again the need to impose a restriction on GoTo. Without some form of restriction, we could create an infinite branch by repeatedly applying it. In, say, an internalised labeled system, every rule contains its own perspective shift since we can work on $@_i$-prefixed formulas one moment and $@_j$-prefixed formulas the next without an explicit rule application in-between: The GoTo rule is coupled with every other rule. This coupling provides an important property for termination: Every perspective change also makes progress. There is no rule to extend the branch with an empty satisfaction statement like GoTo adds an empty block. Blackburn et al. make a similar remark when relating the restrictions to restrictions on a labeled deduction system [BBBJ17]. And notably, Bolander and Blackburn show that you can get a terminating system for the labeled system without introducing the ability to get stuck [BB07].

3.3.4 Money in the bank

I will use the notion of having *money in the bank* to obtain a similar coupling between applications of GoTo and the rest of the rules. Every regular rule application adds one coin to our savings and we pay for every GoTo with one coin. Additionally, we are allowed to start with some number of coins in the bank. Intuitively, this restriction prevents an infinite number of GoTos in a row since our savings at any point in time are finite and decrease with each application.

The initial saving is what makes properties like weakening and substitution easier to prove without pruning detours. In case a rule application is invalidated on the modified branch, causing two GoTos in a row, we can justify this by assuming a larger initial saving and spending some of that. A closed branch is still finite, so the required initial saving is too. We can then show separately, as it is done in Section 4.1 on page 25, that if a branch can be closed then it can be closed starting from just a single coin. Note that there are example formulas, like $@_i(\phi \vee \neg\phi)$, where an initial coin *is* required to close the branch since it must start with a GoTo. The reformulated R4 restriction becomes:

R4 The GoTo rule consumes one coin from the bank. (The remaining rules add one coin.)

If we want to, taking a wrong turn can be resolved by “borrowing money” from the bank. If a branch Θ can be closed starting from n coins I write $n \vdash \Theta$. If n is unknown or irrelevant, I typically write $\vdash \Theta$.

3.4 Formalization

The formalization is based on lists which are ordered sequences. Formally a block is a list of formulas paired with an opening nominal and a branch is a list of blocks. A formula is *on* a block if it is either contained in the list of formulas or equals the opening nominal.

Figure 3.12 on page 22 shows the definition of ST in Isabelle as a set of inductively defined rules. The premises of the rules are separated by the meta-implication, $\Longrightarrow$, with the conclusion at the end. The # symbol separates the head and tail of a list and the constructor *Suc* adds one to a natural number.

It is instructive to look at a verified tableau that uses these rules; Figure 3.11 proves the validity of $@_i i$. For exposition I have divided the proof into four sections delimited by empty lines.

```
lemma
  fixes i
  defines ‹p ≡ ¬ (@ i (Nom i))›
  shows ‹Suc n ⊢ [([p], a)]›
proof -

  have ‹i ∈ branch-nominals [([p], a)]›
    unfolding p-def branch-nominals-def by simp
  then have ?thesis if ‹n ⊢ [([], i), ([p], a)]›
    using that GoTo by fast

  moreover have ‹new (¬ Nom i) i [([], i), ([p], a)]›
    unfolding p-def new-def by auto
  moreover have ‹(¬ (@ i (Nom i))) at a in [([], i), ([p], a)]›
    unfolding p-def by fastforce
  ultimately have ?thesis if ‹Suc n ⊢ [([¬ Nom i], i), ([p], a)]›
    using that SatN by fast

  then show ?thesis
    by (meson Close list.set-intros(1) on.simps)
qed
```

Figure 3.11: Closing tableau for the formula $@_i i$ verified in Isabelle/HOL.

The first section begins the proof by defining an abbreviation for the root formula and stating the goal of the lemma after **shows**. Note from the *Suc* that we are assuming at least one coin to spend on a GoTo. The hyphen after **proof** means that we are doing a direct proof, as opposed to, say, an induction proof.

The first two lines of the second section prove the premise of the GoTo rule: That i already occurs on the branch. The term after **have** is the local goal and the commands on the following line are the justification. The next two lines prove that if the branch extended with an empty i-block can be closed (using one less coin) then so can the original branch. Here, *?thesis* is an abbreviation for the goal of the lemma as defined in the first section. The command **then** is used to make the previously shown fact about i available to the justification of the second **have**. And **using** does the same for named facts like the GoTo rule.

The third section first proves the two premises of the ($\neg$@) rule: That the output is new and that the input occurs on the branch. These facts are chained together with the one above using the **moreover** command so that the three of them together are available on lines 5 and 6 due to **ultimately**. Thus, we see that $\neg i$ is a legal extension of the branch.

Finally, the fourth and last section shows the thesis using the closing condition, *Close*, and terminates the lemma with the **qed** command.

3.4.1 Soundness

The formalized soundness proof follows the one by Blackburn et al. [BBBJ17]. Since all blocks in the present system are named by an opening nominal, I can use a stricter definition of block satisfiability that does not existentially quantify over a world.

Definition 3.1 (Satisfiability) *Let M be a model and g an assignment. A block B with opening nominal i is satisfied by M and g, written $M, g \models_B B$, if for all ϕ on B, $M, g, g(i) \models \phi$. A branch Θ is satisfied by M and g, written $M, g \models_\Theta \Theta$ if it is satisfied block-wise, that is for all blocks B in Θ, $M, g \models_B B$.*

If a branch is closeable then it is not satisfied by any model and assignment.

Lemma 3.2 (Branch Unsatisfiability) *If $\vdash \Theta$ then for all models M and assignments g, $M, g \not\models_\Theta \Theta$.*

Proof. By rule induction for an arbitrary assignment. In each case we assume Θ to be satisfied by an arbitrary M and g and derive a contradiction.

Closing If the branch closes directly then we have both ϕ and $\neg\phi$ at i in Θ for some ϕ and i. But since Θ is satisfied by M and g, we then have $M, g, g(i) \models \phi$ and $M, g, g(i) \models \neg\phi$ which is a contradiction.

```
inductive ST :: ‹nat ⇒ ('a, 'b) branch ⇒ bool› (‹- ⊢ -› [50, 50] 50) where
  Close: ‹p at i in branch ⟹ (¬ p) at i in branch ⟹ n ⊢ branch›
| Neg:
  ‹(¬ ¬ p) at a in (ps, a) # branch ⟹
   new p a ((ps, a) # branch) ⟹
   Suc n ⊢ (p # ps, a) # branch ⟹
   n ⊢ (ps, a) # branch›
| DisP:
  ‹(p ∨ q) at a in (ps, a) # branch ⟹
   new p a ((ps, a) # branch) ⟹ new q a ((ps, a) # branch) ⟹
   Suc n ⊢ (p # ps, a) # branch ⟹ Suc n ⊢ (q # ps, a) # branch ⟹
   n ⊢ (ps, a) # branch›
| DisN:
  ‹(¬ (p ∨ q)) at a in (ps, a) # branch ⟹
   new (¬ p) a ((ps, a) # branch) ∨ new (¬ q) a ((ps, a) # branch) ⟹
   Suc n ⊢ ((¬ q) # (¬ p) # ps, a) # branch ⟹
   n ⊢ (ps, a) # branch›
| DiaP:
  ‹(◇ p) at a in (ps, a) # branch ⟹
   i ∉ branch-nominals ((ps, a) # branch) ⟹
   ∄ a. p = Nom a ⟹ ¬ witnessed p a ((ps, a) # branch) ⟹
   Suc n ⊢ ((@ i p) # (◇ Nom i) # ps, a) # branch ⟹
   n ⊢ (ps, a) # branch›
| DiaN:
  ‹(¬ (◇ p)) at a in (ps, a) # branch ⟹
   (◇ Nom i) at a in (ps, a) # branch ⟹
   new (¬ (@ i p)) a ((ps, a) # branch) ⟹
   Suc n ⊢ ((¬ (@ i p)) # ps, a) # branch ⟹
   n ⊢ (ps, a) # branch›
| SatP:
  ‹(@ a p) at b in (ps, a) # branch ⟹
   new p a ((ps, a) # branch) ⟹
   Suc n ⊢ (p # ps, a) # branch ⟹
   n ⊢ (ps, a) # branch›
| SatN:
  ‹(¬ (@ a p)) at b in (ps, a) # branch ⟹
   new (¬ p) a ((ps, a) # branch) ⟹
   Suc n ⊢ ((¬ p) # ps, a) # branch ⟹
   n ⊢ (ps, a) # branch›
| GoTo:
  ‹i ∈ branch-nominals branch ⟹
   n ⊢ ([], i) # branch ⟹
   Suc n ⊢ branch›
| Nom:
  ‹p at b in (ps, a) # branch ⟹ Nom i at b in (ps, a) # branch ⟹
   Nom i at a in (ps, a) # branch ⟹ new p a ((ps, a) # branch) ⟹
   Suc n ⊢ (p # ps, a) # branch ⟹
   n ⊢ (ps, a) # branch›
```

Figure 3.12: Definition of ST in Isabelle/HOL.

Case ($\neg\neg$) By assumption we have $\neg\neg\phi$ at a in Θ which is satisfied by M and g, so $M, g, g(a) \models \neg\neg\phi$, so $M, g, g(a) \models \phi$. The current block is an a-block so $M, g \models_\Theta \phi -_a \Theta$ and this contradicts the induction hypothesis.

Case ($\Diamond$) We have $\Diamond\phi$ at a in Θ, so $M, g, g(a) \models \Diamond\phi$. Let v be the world accessible from $g(a)$ s.t. $M, g, v \models \phi$. By the induction hypothesis, i is fresh in Θ, so we have $M, g(i := v), v \models \phi$ where $g(i := v)$ is the assignment that maps i to v and every other nominal j to $g(j)$. That means we know $M, g(i := v), g(a) \models @_i\phi$. Moreover, since v is accessible from $g(a)$, we have $M, g(i := v), g(a) \models \Diamond i$. In combination, Θ extended by $@_i\phi$ and $\Diamond i$ is satisfied by M and $g(i := v)$: $M, g(i := v) \models_\Theta @_i p -_a \Diamond i -_a \Theta$. This contradicts the induction hypothesis.

The remaining cases are similar to ($\neg\neg$). □

Corollary 3.3 (Formula Unsatisfiability) *If $\vdash \Theta$ then for any M, g and w, there is a ϕ on a block in Θ such that $M, g, w \not\models \phi$.*

Proof. Assume that for a given M and g no such ϕ exists, then Θ is satisfied by M and g which contradicts Lemma 3.2 on page 21.

Theorem 3.4 (Soundness) *If $\vdash \Theta$ where Θ consists of just $\neg\phi$ on an i-block and i does not occur in ϕ, then ϕ is valid.*

Proof. Fix an arbitrary model M, assignment g and world w. There are only two formulas on Θ so by Corollary 3.3, for any g' either $M, g', w \not\models \neg\phi$ or $M, g', w \not\models i$. That is, either $M, g', w \models \phi$ or $M, g', w \not\models i$.

That means that $M, g', g'(i) \models \phi$, since $M, g', g'(i) \models i$ by definition. In particular $M, g(i := w), (g(i := w))(i) \models \phi$ since g' was arbitrary. This reduces to $M, g(i := w), w \models \phi$ and since i is assumed fresh we can drop its reassignment and show $M, g, w \models \phi$. Since M, g and w were chosen arbitrarily, this proves the validity of ϕ. □

Chapter 4

Lifting Restrictions

The synthetic completeness proof by Jørgensen et al. was made for the unrestricted version of ST [JBBB16]. Before formalizing the completeness result it will therefore be useful to show that the restricted rules presented in the former chapter do not affect which branches can be closed. In fact for restrictions R1, R2 and R5 we can show the unrestricted versions of the rules to be admissible, which I will do in this chapter. First however, I will show that one initial coin suffices to close any closeable branch.

4.1 No Detours

The proof technique will be as follows: Given a branch that can be closed by some number of rule applications we are going to cut it in half. The blocks above the cut are untouched while below the cut we will filter away empty blocks, i.e. detours. All extensions are made below the cut and I will show that the cut branch can be closed starting from a single coin. In the end, we can choose to cut the branch so that all root blocks are preserved, that is, only detours in the extension are filtered away, and thus obtain our result.

I will use comma liberally to denote the extension of a branch Θ with a block B, as in $\vdash B, \Theta$, or to append two branches, as in $\vdash \Theta, \Theta'$. Given a branch Θ I am going to write $\lfloor \Theta \rfloor$ for Θ with all empty blocks removed, where an empty block consists only of its opening nominal. First, a remark and a technical lemma.

Remark 4.1 (More coins) *If $n \vdash \Theta$ then $n + m \vdash \Theta$.*

Lemma 4.2 (Filter block extension) *Let B be a block whose opening nominal occurs in $\lfloor \Theta_l \rfloor, \Theta_r$. Including B in the filtered section of the branch may require assuming another initial coin. That is, if $n \vdash B, \lfloor \Theta_l \rfloor, \Theta_r$ then $n + 1 \vdash \lfloor B, \Theta_l \rfloor, \Theta_r$.*

Proof. There are two cases. If B is empty, we need to apply GoTo to go from $n \vdash B, \lfloor \Theta_l \rfloor, \Theta_r$ to $n + 1 \vdash \lfloor \Theta_l \rfloor, \Theta_r$ since $\lfloor B, \Theta_l \rfloor = \lfloor \Theta_l \rfloor$. If B is nonempty, $\lfloor B, \Theta_l \rfloor = B, \lfloor \Theta_l \rfloor$ and the case follows from Remark 4.1. □

We are now in a position to prove the main lemma which is more general than we need so it can be proved by induction.

Lemma 4.3 (Filtering detours) *If a branch can be closed starting from n coins, then any filtering cut of the branch can be closed from $m+1$ coins. That is, if $n \vdash \Theta_l, \Theta_r$ then $m + 1 \vdash \lfloor \Theta_l \rfloor, \Theta_r$.*

Proof. Proof by induction over the construction of the closing tableau, for arbitrary Θ_l, Θ_r.

Closing By assumption we have ϕ and $\neg\phi$ at i in Θ_l, Θ_r and since filtering only removes empty blocks, we still have them in $\lfloor \Theta_l \rfloor, \Theta_r$. If $\phi = k$ for some nominal k then the filtering may remove a block whose opening nominal is k, but if that was our given assumption then $i = k$ and so i occurs on the same block as $\neg\phi$ which is preserved by definition.

Case $(\neg\neg)$ We have $\neg\neg\phi$ at a in Θ_l, Θ_r and ϕ new to a. There are two cases.

If Θ_l is empty then we have $m + 1 \vdash \phi -_a \Theta_r$ by the induction hypothesis, By $(\neg\neg)$ we know $m \vdash \Theta_r$ and so by Remark 4.1 we have $m + 1 \vdash \Theta_r$ which is what needs to be shown.

If Θ_l is nonempty, then split it into the current block B and the rest Θ_l'. The induction hypothesis tells us that $m + 1 \vdash \lfloor \phi -_a B, \Theta_l' \rfloor, \Theta_r$. The extension makes the current block nonempty, so this reduces to $m + 1 \vdash \phi -_a B, \lfloor \Theta_l' \rfloor, \Theta_r$. The a-block with $\neg\neg\phi$ is preserved by the filtering and so is the newness of ϕ

so by $(\neg\neg)$ we have $m \vdash B, \lfloor \Theta_l' \rfloor, \Theta_r$. And now from Lemma 4.2 on the facing page we have $m + 1 \vdash \lfloor B, \Theta_l' \rfloor, \Theta_r \equiv m + 1 \vdash \lfloor \Theta_l \rfloor, \Theta_r$ which is the thesis.

Note that filtering cannot witness an unwitnessed a diamond, so except for GoTo, the rest of the cases follow similarly.

GoTo The induction hypothesis gives us $m+1 \vdash \lfloor B, \Theta_l \rfloor, \Theta_r$ where B is empty. This reduces to what needs to be shown: $m + 1 \vdash \lfloor \Theta_l \rfloor, \Theta_r$. □

Theorem 4.4 (Positive coins) *If $n \vdash \Theta$ then $m + 1 \vdash \Theta$.*

Proof. By Lemma 4.3 on the preceding page for empty Θ_l and $\Theta_r = \Theta$. □

Corollary 4.5 (A single coin) *If $n \vdash \Theta$ then $1 \vdash \Theta$.*

Proof. By Theorem 4.4 for $m = 0$. □

4.1.1 Free GoTo

From Theorem 4.4 we can obtain a version of GoTo that does not spend any coins, but does require at least one coin to be available.

Theorem 4.6 (Free GoTo)
If $n + 1 \vdash B, \Theta$ where B is an empty block whose opening nominal occurs in Θ, then $n + 1 \vdash \Theta$.

Proof. By applying GoTo we have $n + 2 \vdash \Theta$ and then Theorem 4.4 gives us $n + 1 \vdash \Theta$ as wanted. □

The requirement for a single coin to be available essentially comes from the case where we need to immediately open a new block to close the branch. Given that we can strengthen any derivation to only make use of one coin, we may wonder whether we actually need the ability to have more coins in the bank or if we could make do with a Boolean flag telling us whether GoTo is allowed or not. That is, could we have a GoTo rule like this: $0 \vdash B, \Theta \Longrightarrow 1 \vdash \Theta$. But note that Theorems 4.4 and 4.6 do not reduce the number of coins to m but to $m + 1$ which would map to 1 in the Boolean scheme and not match the antecedent of the proposed GoTo, preventing further applications. For the proof, we need the interim flexibility of having more than one coin available.

4.2 As Good as New

In this section I will show that any formula that is not new to a branch can be omitted. Or viewed from the point of constructing a branch, that we can extend the current block with a formula that already occurs on a block of the same type. This allows us to obtain versions of the ST rules without the R1 restriction. Figure 4.1 illustrates the proof idea: Given a branch, we are going to mark a lasting occurrence of ϕ on an i-block ($\rightarrow$) along with a number of other occurrences that are to be omitted on the new branch (†). I will then show that every time one of the omitted occurrences is used as rule input, we could instead have used the lasting occurrence. To omit the R1 restriction, we consider a branch with an extension that is legal for some rule but maybe not under R1. If the extension includes something new then the original rule applies and if not then it can be omitted as it occurs elsewhere.

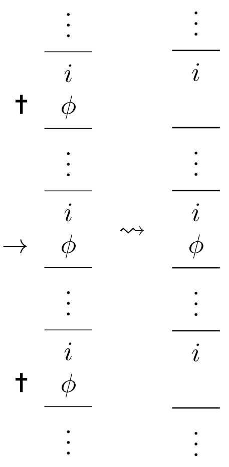


Figure 4.1: Strengthening.

To mark occurrences I will make use of a set of indices into the branch which is introduced in the next section. Since the proof is not very difficult, I will focus on its formalization.

4.2.1 Indexing

Indices into a branch are pairs of natural numbers where the first component specifies the block and the second component specifies the formula on that block. The notation $\Theta(v)$ stands for the v'th block in Θ and $\Theta(v)(v') \equiv \Theta(v, v')$ stands for the v''th formula on that block. Figure 4.2 illustrates the scheme. The indices start from the top such that they do not need to be updated if we extend the branch with a new block or the current block with additional formulas. If Δ is an index set, let $\Delta(v) = \{v' \mid (v, v') \in \Delta\}$ be its v-projection.

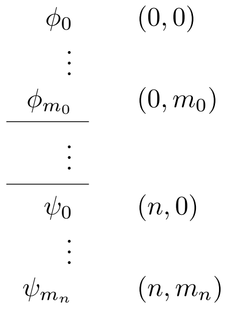


Figure 4.2: Indexing.

4.2.2 Duplicates

The following definition is called *Dup p i branch* in the formalization.

Definition 4.7 (Duplicate set) *A set of indices Δ is a duplicate set for ϕ at i in Θ iff for every index $(v, v') \in \Delta$, $\Theta(v)$ is an i-block, $\Theta(v, v') = \phi$ and there is an index $(w, w') \notin \Delta$ such that $\Theta(w)$ is an i-block and $\Theta(w, w') = \phi$.*

Remark 4.8 (Duplicate set projection) *If Δ is a duplicate set for ϕ at i in Θ and $\Theta(v)$ is some block, then for all $v' \in \Delta(v)$, $\Theta(v, v') = \phi$.*

4.2.3 Omitting

The omitting of formulas on a branch is decomposed into omitting formulas on its blocks using the index projection. Recall that blocks formally consist of a list of formulas paired with an opening nominal. The following operation, *omit*, prunes a list of formulas, *ps*, based on a set of block indices, *xs*:

```
primrec omit :: ‹nat set ⇒ ('a, 'b) fm list ⇒ ('a, 'b) fm list› where
  ‹omit xs [] = []›
| ‹omit xs (p # ps) = (if length ps ∈ xs then omit xs ps else p # omit xs ps)›
```

The first line specifies the type of the function and the next line the operation on an empty list. The final line checks whether to include the formula *p* in the result by looking up its index, *length ps*, in the given index set.

We lift this to work on the branch using the following indexed mapping function which applies a function *f* to every element of the list but also passes as argument to *f* the index of the element, *length xs*:

```
primrec mapi :: ‹(nat ⇒ 'a ⇒ 'b) ⇒ 'a list ⇒ 'b list› where
  ‹mapi f [] = []›
| ‹mapi f (x # xs) = f (length xs) x # mapi f xs›
```

```
definition omit-branch :: ‹(nat × nat) set ⇒ ('a, 'b) branch ⇒ ('a, 'b) branch›
where
  ‹omit-branch xs branch ≡ mapi (λv. omit-block (proj xs v)) branch›
```

In the above, the function *omit-block* is *omit* lifted to work on blocks and *proj xs v* is exactly the projection $\Delta(v)$ where *xs* is how I write Δ in the formalization.

4.2.4 Induction

The main lemma *ST-Dup* is shown by rule induction and the full proof is omitted. Note that strengthening does not change the number of coins in the bank, since any omitted formula cannot have come about through a rule application because that would violate R1:

lemma *ST-Dup*:
  assumes ⟨*n* ⊢ *branch*⟩ ⟨*Dup q i branch xs*⟩
  shows ⟨*n* ⊢ *omit-branch xs branch*⟩

Most cases follow straight-forwardly from the following lemmas: If a formula occurs at *a* before the strengthening then it does so afterwards and vice versa:

lemma *omit-branch-mem*:
  assumes ⟨*Dup p i branch xs*⟩ ⟨*q at a in branch*⟩
  shows ⟨*q at a in omit-branch xs branch*⟩

lemma *omit-branch-mem-dual*:
  assumes ⟨*p at i in omit-branch xs branch*⟩
  shows ⟨*p at i in branch*⟩

The latter result is used to show, contrapositively, that diamonds remain unwitnessed on the strengthened branch and that formulas in general remain new; a requirement since we have yet to lift R1 and R2.

Finally for GoTo and ($\Diamond$) it is necessary to show that after strengthening the branch contains the same set of nominals.

Note that the result does not hold if any of the rules requires its input to occur on a specific block, like the original Nom rule requires the shared nominal to occur on the current block, since we are allowed to omit any formula in favor of another and R1 prevents us from making any copies.

4.2.5 Lifting R1

With the above result we can obtain the following rule for extending the branch with an "old" formula:

theorem *Dup*:
  assumes ‹$n \vdash (p \ \# \ ps, \ a) \ \# \ branch$› ‹$\neg \ new \ p \ a \ ((ps, \ a) \ \# \ branch)$›
  shows ‹$n \vdash (ps, \ a) \ \# \ branch$›

In the formalization, opening nominals cannot be indexed as they do not occur in the list with the other formulas but as the second component of a pair. This complicates the proof of *Dup* but I think it is a reasonable trade-off over making the indexing more complex or losing the ability to easily discern the opening nominal of a block.

Using *Dup*, the R1-unrestricted rules can now be obtained automatically and I will use these from now on. Take for instance the (¬¬) rule:

lemma *Neg′*:
  assumes
    ‹$(\neg \ \neg \ p) \ at \ a \ in \ (ps, \ a) \ \# \ branch$›
    ‹$n \vdash (p \ \# \ ps, \ a) \ \# \ branch$›
  shows ‹$n \vdash (ps, \ a) \ \# \ branch$›

Note again that the number of coins is unchanged, since the extension may have been justified by strengthening instead of a rule application. One could use Theorem 4.4 on page 27 to write 1 instead of n in the goal, but this would throw away information in the case where n is zero, so I prefer this variant.

Strengthening is the only property presented in this chapter that always preserves the ability to close a branch starting from zero coins so for brevity the remaining lemmas will be shown for an existentially quantified number of coins. Therefore it is useful to prove the following generalization of the (∨) rule where the number of coins in each branch does not need to match:

lemma *DisP″*:
  assumes
    ‹$(p \lor q) \ at \ a \ in \ (ps, \ a) \ \# \ branch$›
    ‹$n \vdash (p \ \# \ ps, \ a) \ \# \ branch$› ‹$m \vdash (q \ \# \ ps, \ a) \ \# \ branch$›
  shows ‹$max \ n \ m \vdash (ps, \ a) \ \# \ branch$›

4.3 Too Many Witnesses

In this section I will first show a substitution lemma and then use it to lift the R2 restriction on the (◇) rule.

4.3.1 Substitution

Jørgensen et al. give a substitution lemma for the unrestricted ST based on substituting a nominal for another, one at a time [JBBB16, Lemma 3.2]. Here I show a similar substitution lemma, but for the restricted, present ST and based on a substituting function applied to all nominals that occur on the branch. The substitution on formulas is encoded as follows in Isabelle:

```
primrec sub :: ‹('b ⇒ 'c) ⇒ ('a, 'b) fm ⇒ ('a, 'c) fm› where
  ‹sub - (Pro x) = Pro x›
| ‹sub f (Nom i) = Nom (f i)›
| ‹sub f (¬ p) = (¬ sub f p)›
| ‹sub f (p ∨ q) = (sub f p ∨ sub f q)›
| ‹sub f (◇ p) = (◇ sub f p)›
| ‹sub f (@ i p) = (@ (f i) (sub f p))›
```

Note that the substitution may change the type of the nominals, say from integers to strings. If θ is a substitution function and ϕ is a formula, $\phi\theta$ denotes the formula obtained by applying the substitution. Similarly, Figure 4.3 illustrates how substitution is lifted to blocks and branches and $B\theta$ and $\Theta\theta$ denote each, respectively. For brevity, I am going to assume that the type of the substitution function always matches the formula, block or branch it is applied to.

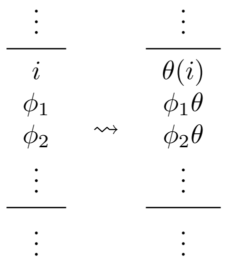


Figure 4.3: Substitution.

I will show that if a closing tableau exists for a branch Θ, then for any substitution θ either with same domain and co-domain or an infinite co-domain, a closing tableau exists for $\Theta\theta$. The requirement on θ is needed for the ($\Diamond$) case.

First, note that since I am not requiring the substitution to be injective, two distinct formulas may be *collapsed* into the same formula by the substitution. As such, a previously R1-legal extension, because the formula was new to the branch, may no longer be one. Lifting R1 is a prerequisite for this lemma.

Remark 4.9 (Substituted occurrence) *Let θ be a substitution function. If ϕ occurs at i in Θ then $\phi\theta$ occurs at $\theta(i)$ in $\Theta\theta$ (Figure 4.3).*

Theorem 4.10 (Substitution) *Let θ be a substitution function. Assume that for all finite sets A, if there exists a nominal not in A then there exists a nominal not in the image of A under f. If $\vdash \Theta$ then $\vdash \Theta\theta$.*

Proof. Shown by rule induction over the construction of Θ for an arbitrary θ.

Closing By assumption we have ϕ and $\neg\phi$ at i in Θ, so by Remark 4.9 on the facing page we have $\phi\theta$ and $(\neg\phi)\theta \equiv \neg(\phi\theta)$ at $\theta(i)$ in $\Theta\theta$. It closes immediately.

Case $(\neg\neg)$ By assumption we have $\neg\neg\phi$ at a in Θ and the current block is an a-block. By the induction hypothesis we know that the substituted branch extended by $\phi\theta$ has a closing tableau: $\vdash \phi\theta -_{\theta(a)} \Theta\theta$. By Remark 4.9 we have $\neg\neg(\phi\theta)$ at $\theta(a)$ in $\Theta\theta$, so by the unrestricted $(\neg\neg)$ rule we have $\vdash \Theta\theta$ as desired.

The cases for $(\vee)$, $(\neg\vee)$, $(\neg\Diamond)$ $(@)$, $(\neg@)$ and Nom are similar. GoTo is trivial.

Case $(\Diamond)$ By assumption we have $\Diamond\phi$ at a in Θ, the nominal i is fresh in Θ and by the induction hypothesis $\vdash (@_i\phi)\theta' -_{\theta'(a)} (\Diamond i)\theta' -_{\theta'(a)} \Theta\theta'$ for any θ'. The $\Diamond\phi$ is unwitnessed at a in Θ but since the substitution may collapse formulas, $\Diamond\phi\theta$ may be witnessed at $\theta(a)$ in $\Theta\theta$. Thus there are two cases:

If $\Diamond\phi\theta$ is witnessed at $\theta(a)$ in $\Theta\theta$ then let i' be the witnessing nominal, such that $@_{i'}(\phi\theta)$ and $\Diamond i'$ both occur at $\theta(a)$ in $\Theta\theta$. Apply the induction hypothesis at $\theta(i := i')$ to obtain $\vdash @_{i'}(\phi\theta) -_{\theta(a)} \Diamond i' -_{\theta(a)} \Theta\theta$, where the added assignment has been reduced away in the places where i is fresh. Both formulas in the extension are justified by the Nom rule so we obtain $\vdash \Theta\theta$ as needed.

Otherwise the formula is unwitnessed. To apply the $(\Diamond)$ rule, we need the witnessing nominal to be fresh in $\Theta\theta$ but since θ is not necessarily injective, this may not be the case for $\theta(i)$. But since Θ is finite, we have by assumption a nominal j that is fresh to $\Theta\theta$. Apply the induction hypothesis at $\theta(i := j)$ to learn $\vdash @_j(\phi\theta) -_{\theta(a)} \Diamond j -_{\theta(a)} \Theta\theta$ where, again, I have reduced the term using the fact that i is fresh in Θ. The $(\Diamond)$ rule now applies: $\Diamond\phi\theta$ is unwitnessed at $\theta(a)$ in $\Theta\theta$ and we have ensured that j is fresh. Thus we can conclude $\vdash \Theta\theta$. $\square$

Corollary 4.11 (Substitution into same type) *Let θ be a substitution function whose domain and codomain coincide. If $\vdash \Theta$ then $\vdash \Theta\theta$.*

Proof. For any finite set A and substitution θ', the cardinality of the image of A under θ is at most that of A since θ' can only collapse distinct nominals, not separate identical ones. Thus the existence of a fresh nominal is preserved and the thesis follows from Theorem 4.10 on the preceding page.

Corollary 4.12 (Substitution with infinite codomain) *Let θ be a substitution function with an infinite codomain. If $\vdash \Theta$ then $\vdash \Theta\theta$.*

Proof. For any finite set A and substitution θ', the image of A under θ' is also finite. Since the codomain of θ' is infinite, a fresh nominal exists and the thesis follows from Theorem 4.10 on page 32.

4.3.2 Lifting R2

To lift R2 we can employ the same trick as in the $(\Diamond)$ case in the proof of Theorem 4.10 on page 32.

Theorem 4.13 (Unrestricted $(\Diamond)$) *If $\vdash @_i\phi -_a \Diamond i -_a \Theta$, i is fresh in Θ and ϕ is not a nominal, then $\vdash \Theta$.*

Proof. If $\Diamond\phi$ is unwitnessed at a in Θ then the restricted $(\Diamond)$ rule applies directly. Otherwise let i' be the witnessing nominal, obtain $\vdash @_i'\phi -_a \Diamond i' -_a \Theta$ using Corollary 4.11 on the preceding page and the fact that i is fresh, and justify the extension using Nom (or strengthening). $\square$

4.4 General Satisfaction

To obtain the unrestricted versions of $(@)$ and $(\neg @)$ as depicted in Figure 3.5 on page 14 we will need to show a structural lemma that allows us to rearrange, add and contract blocks on a branch.

Figure 4.4 illustrates the structural property. Given a closeable branch consisting of blocks $B_1, B_2, \ldots, B_n$ in that order, I will show that a closing branch exists for any sequence of blocks $B'_1, B'_2, \ldots, B'_m$ that are a superset of the original: $\{B_1, B_2, \ldots, B_n\} \subseteq \{B'_1, B'_2, \ldots, B'_m\}$. This lemma is slightly stronger than Lemma 3.3(i) given by Jørgensen et al. which states that a closed tableau exists for any superset of blocks but leaves the order of the blocks unspecified [JBBB16]. As shown here, we get to pick the order.

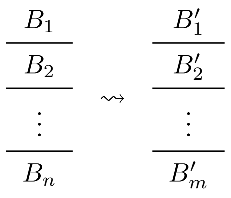


Figure 4.4: Rearranging.

4.4.1 Rearranging blocks

Denote the formulas on a block B by Formulas(B).

Lemma 4.14 (Dropping current block) *If* $\vdash B_n, B_{n-1}, \ldots, B_1$*, and there is an* i *such that* $1 \leq i \leq n-1$, Formulas(B_n) $\subseteq$ Formulas(B_i) *and* B_i, B_n *have the same opening nominal then* $\vdash B_{n-1}, \ldots, B_1$*.*

Proof. Figure 4.5 illustrates the applications of Nom and GoTo. □

Since the number of formulas to be dropped by Nom can vary, the proof is formalized using induction. However, the standard induction principle works in the wrong direction: Assuming a property holds for a list, it asks us to show that it still holds when *adding* an extra element; here we want to show that the branch remains closeable when *removing* a formula from the current block. Therefore, I use the following custom induction principle instead:

```
lemma list-down-induct
  [consumes 1, case-names Start Cons]:
  assumes ‹∀ y ∈ set ys. Q y› ‹P (ys @ xs)›
    ‹⋀y xs. Q y ⟹ P (y # xs) ⟹ P xs›
  shows ‹P xs› using assms by (induct ys) auto
```

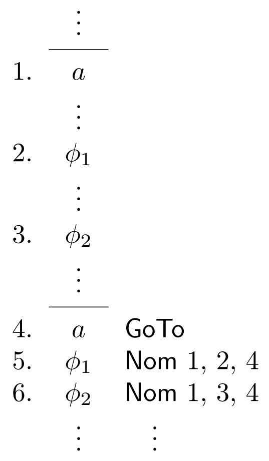


Figure 4.5: Dropping a block.

The principle allows us to drop a prefix of a list if we know that a property, Q, holds for all elements in that prefix, here that the formula appears earlier on the same type of block. There are two cases dubbed *Start* and *Cons*. For the former we need to show that the property P holds for the full list, *P (ys @ xs)*, and for the latter that if we know the property holds for a list, *xs*, with some head *y*, *P (y # xs)*, and we know *Q y*, then we we can show *P xs*.

The following notation is useful for the next theorem: Given a branch Θ consisting of blocks $B_1, B_2, \ldots, B_n$, let Blocks(Θ) be the set of blocks $\{B_1, B_2, \ldots, B_n\}$.

Theorem 4.15 (Branch structure) *Assume the universe of nominals is infinite. If* $\vdash \Theta$ *and* Blocks(Θ) $\subseteq$ Blocks(Θ') *then* $\vdash \Theta'$*.*

Proof. By induction over the construction Θ's closing tableau, for arbitrary Θ'. In the following, let $\Theta \equiv B_n, B_{n-1}, \ldots, B_1$ and $\Theta' \equiv B'_m, B'_{m-1}, \ldots, B'_1$.

Closing By assumption we have both ϕ on some i-block in Θ and $\neg\phi$ on some i-block in Θ. Since all blocks in Θ also occur in Θ', Θ' closes immediately.

Case $(\neg\neg)$ By assumption we have $\neg\neg\phi$ at a in Θ and the current block, B_n, is an a-block. The induction hypothesis says that Blocks(Θ) $\subseteq$ Blocks(Θ') and that for any Θ'' whose blocks are a superset of Blocks($\phi -_a B_n, B_{n-1}, \ldots, B_1$), we know $\vdash \Theta''$.

Since the current block of Θ' is not necessarily B_n, we cannot apply the induction hypothesis directly at $\phi -_a \Theta'$. Instead, we add the entire block $\phi -_a B_n$ to the front of Θ' and learn that $\vdash \phi -_a B_n, \Theta'$.

From the assumption, we have $\neg\neg\phi$ at a in B_n, Θ' so the $(\neg\neg)$ rule applies and we obtain $\vdash B_n, \Theta'$. We need to show $\vdash \Theta'$, but we know by the induction hypothesis that B_n occurs in Θ' so the case closes by Lemma 4.14 on the preceding page.

The cases for $(\vee)$, $(\neg\vee)$, $(\neg\Diamond)$, $(@)$, $(\neg@)$, GoTo and Nom are similar.

Case $(\Diamond)$ In this case we have by assumption $\Diamond\phi$ at a in Θ, the current block is an a-block, ϕ is not a nominal and the nominal i is fresh to Θ. The induction hypothesis says that Blocks(Θ) $\subseteq$ Blocks(Θ') and that for any Θ'' whose blocks are a superset of Blocks($@_i\phi -_a \Diamond i -_a B_n, B_{n-1}, \ldots, B_1$), we know $\vdash \Theta''$.

We cannot follow the same strategy as for $(\neg\neg)$ since i may not be fresh to Θ' and so the $(\Diamond)$ rule would not apply to justify the extension. Instead, let j be a nominal that is fresh to Θ' (and thus also Θ). The existence of j is justified by the fact that Θ' contains a finite number of nominals and the universe of nominals is assumed to be infinite.

Let θ be the substitution that simultaneously maps i to j and j to i and leaves all other nominals as is. Note that for any block $B \in$ Blocks(Θ), $B\theta = B$ since both i and j are fresh to Θ. Therefore, Blocks(Θ) $\subseteq$ Blocks($\Theta'\theta$); only any new blocks in Θ' are affected by the substitution. And so, the previous trick applies: Blocks($@_i\phi -_a \Diamond i -_a \Theta$) $\subseteq$ Blocks($@_i\phi -_a \Diamond i -_a B_n, \Theta'\theta$) and the induction hypothesis gives us $\vdash @_i\phi -_a \Diamond i -_a B_n, \Theta'\theta$.

The witnessing nominal i is fresh to B_n by assumption and to $\Theta'\theta$ due to θ, so rule $(\Diamond)$ applies and we learn $\vdash B_n, \Theta'\theta$. We have $B_n\theta = B_n \in$ Blocks($\Theta'\theta$) so by Lemma 4.14 on the previous page we know $\vdash \Theta'\theta$. Finally, we use Corollary 4.11 on page 33 to obtain $\vdash \Theta'\theta\theta$ which reduces to $\vdash \Theta'$ by the nature of θ. $\square$

Theorem 4.16 (Block structure) *Assume the universe of nominals is infinite. If* $\vdash B_n, \Theta$, $\text{Formulas}(B_n) \subseteq \text{Formulas}(B'_n)$ *and* B_n, B'_n *have the same opening nominal then* $\vdash B'_n, \Theta$.

Proof. Similar to that of Theorem 4.15 on page 35. The GoTo case follows from that theorem and Lemma 4.14 on page 35, without appeal to the induction hypothesis by first weakening with B'_n and then dropping B_n. □

4.4.2 Lifting R5

Recall the general rules for the satisfaction statement as given in Figure 3.5 on page 14. E.g. (@): If in Θ we have $@_i\phi$ at b and i at a and the current block is an a-block then ϕ is a sound extension. Or if we think in the other direction, then the branch remains closeable without ϕ.

Figure 4.6 shows how weakening is used to obtain this version of the rule from its restricted variant. Each squiggly arrow, ($\rightsquigarrow$), represents an application of Theorem 4.15 on page 35. We start on the left from the branch that we assume has a closing tableau. Through weakening so does the middle branch, where the removal of ϕ is justified by Nom. By swapping the bottom two blocks, we arrive at the right branch and from the (@), Nom and GoTo rules we know that it can be closed without the extension.

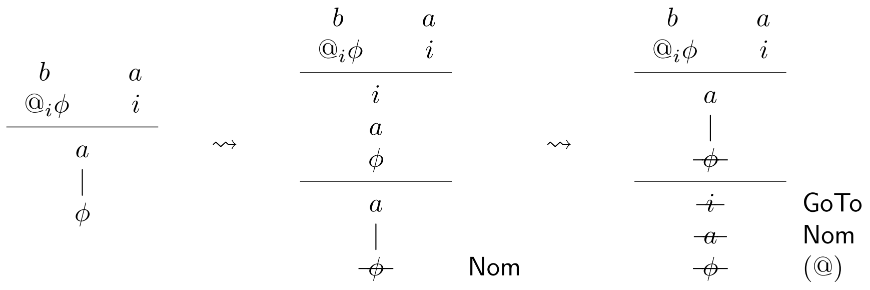


Figure 4.6: Proving the unrestricted (@) rule.

The lifting of R5 for (¬@) is similar and omitted. Note that this lifting makes use of Theorem 4.15 on page 35 and thereby relies on the universe of nominals to be infinite. Therefore, I will assume this to be the case from now on. Since we are not weakening with any fresh nominals it should be possible to prove the lifting without this assumption but for brevity this is not done here.

As Figure 4.6 illustrates with the application of Nom, Theorem 4.15 shows that forward referencing is admissible in the system as long as it is acyclic. We are justifying ϕ by an extension that appears after ϕ but does not depend on it, so it might as well have appeared before and using it to justify ϕ is allowed.

Chapter 5

Bridge

5.1 Overview

Figure 5.1 shows the Bridge rule: If j is reachable from i and j and k are equivalent, then k is reachable from i. This rule will be of use for the completeness proof but is also interesting in its own right. Jørgensen et al. note that the rule can be seen as a restricted form of cut for nominals [JBBB16]. They show that ST can do something similar to Bridge in terms of the two blocks that appear in the completeness proof but do not show admissibility of the general rule.

$$\begin{array}{ccc} i & a & a \\ \Diamond j & j & k \\ \hline \multicolumn{3}{c}{i} \\ \multicolumn{3}{c}{|} \\ \multicolumn{3}{c}{\Diamond k} \end{array}$$

Figure 5.1: The Bridge rule.

In this chapter I will show that Bridge *is* admissible in ST. The idea is as follows: Assuming $\vdash \Diamond k -_i \Theta$, I will show first $\vdash \Diamond j -_i \Theta$ and then use the Nom rule to justify the $\Diamond j$ extension. However, closing the branch may have relied on using $\Diamond k$ as input to some rule. This motivates the notion of a *descendant* which will be formally defined below. Intuitively, I will show every descendant of the $\Diamond k$ can be replaced in a suitable way such that the branch can still be closed.

The proof follows the one by Jørgensen et al. [JBBB16] with two important generalizations. First, their notion of a descendant is tied directly to rule applications whereas mine only depends on the conditions for a rule to be applied. In other words, where they ask whether a given formula is certainly a descendant, I ask whether it could be considered one. This means I only have to look at the formulas on the branch, not the applied rules. Second, where Jørgensen et al. replace every descendant on a branch, I replace only occurrences specified by a given set of indices into the branch. This makes it easier to employ standard rule induction and thus eases formalization. In the end, I take the set to contain only the index of the single $\Diamond k$ that I want to be replaced.

5.2 Descendants

This section defines what can be considered a descendant of $\Diamond k$ at i in Θ. If we inspect the rules of ST then we see that $\Diamond k$ can only be used as input to two rules, $(\neg\Diamond)$ and Nom. Based on this information, we inductively built a set of indices that satisfies a $D_{\Theta,i,k}$-relation. If a set is $D_{\Theta,i,k}$ then all indices in the set point at formulas that can be considered descendants of $\Diamond k$ on an i-block in Θ, including the initial $\Diamond k$. There are three cases, the first starts off the set, the second mimics the $(\neg\Diamond)$ rule and the third mimics the Nom rule:

Definition 5.1 (Descendant set)

Initial *If $\Theta(v)$ is an i-block and $\Theta(v, v') = \Diamond k$, then $\{(v, v')\}$ is $D_{\Theta,i,k}$.*

Derived *If Δ is $D_{\Theta,i,k}$, $(w, w') \in \Delta$, $\Theta(w)$ is an a-block, $\Theta(w, w') = \Diamond k$, $\Theta(v)$ is an a-block and $\Theta(v, v') = \neg @_k \phi$ for some ϕ then $\{(v, v')\} \cup \Delta$ is $D_{\Theta,i,k}$.*

Copied *If Δ is $D_{\Theta,i,k}$, $(w, w') \in \Delta$, $\Theta(w)$ is a b-block, $\Theta(w, w') = \phi$, there is a nominal j that occurs both at a and b in Θ, $\Theta(v)$ is an a-block and $\Theta(v, v') = \phi$ then $\{(v, v')\} \cup \Delta$ is $D_{\Theta,i,k}$.*

Lemma 5.2 (Descendant types.) *If Δ is $D_{\Theta,i,k}$ and $(v, v') \in \Delta$ then the formula $\Theta(v, v')$ is either $\Diamond k$ or $\neg @_k \phi$ for some ϕ.*

Proof. By inspection of the cases for constructing Δ. □

Lemma 5.2 is an analogue of Lemma 4.1 by Jørgensen et al. [JBBB16].

Note that due to the Nom rule we cannot say anything about the opening nominal of the block that the descendant appears on.

Lemma 5.3 (Descendants of extensions.) *If Δ is $D_{\Theta,i,k}$ and Θ' is an extension of Θ then Δ is $D_{\Theta',i,k}$.*

Proof. By construction all indices in Δ are bound in Θ. Since indices are stable under extensions, Δ is $D_{\Theta',i,k}$. □

5.3 Replacements

Having considered $\Diamond k$ as input to a rule, let us now consider the corresponding output. For the $(\neg\Diamond)$ rule, the output is $\neg @_k \phi$ where ϕ is also part of the rule input. If we replace the $\Diamond k$ with a $\Diamond j$, we need to replace the rule output with $\neg @_j \phi$ for it to be justified. For the Nom rule we output the input formula, so we need to replace $\Diamond k$ with $\Diamond j$ in both places.

Following Jørgensen et al. [JBBB16], I call $\neg @_j \phi$ the j-replacement of $\neg @_k \phi$ and $\Diamond j$ the j-replacement of $\Diamond k$. The j-replacement of any other formula is itself. I also use p^j to denote the j-replacement of p. The branch obtained by replacing every index from Δ in Θ by its j-replacement is denoted Θ^Δ.

Remark 5.4 (Out of bounds indices.) *If none of the indices in Δ' are bound in Θ then $\Theta^{\Delta \cup \Delta'} = \Theta^\Delta$.*

Lemma 5.5 (Θ^Δ nominals) *Disregarding j and k, the same nominals occur on Θ and Θ^Δ.*

Proof. The only difference between Θ and Θ^Δ is that some occurrences of k may have been replaced by j. □

5.4 Induction

Lemma 5.6 (Replacing descendants.) *If $\vdash \Theta$, j and k both appear at c in Θ and the set of indices Δ is $D_{\Theta,i,k}$, then $\vdash \Theta^\Delta$.*

Proof. By rule induction for an arbitrary Δ.

Closing By assumption we have both ϕ and $\neg\phi$ at a in Θ.

In Θ^Δ, these occurrences may or may not have been replaced depending on their shape and on Δ. By the definition of a j-replacement, at most one of them can be replaced at a time. Assuming one of them *is* replaced, there are three cases as seen in Table 5.1 on the facing page. These three cases each have a closing extension as seen in Figure 5.2 on the next page. In the fourth case where none of ϕ and $\neg\phi$ are replaced, Θ^Δ closes immediately.

Case $(\neg\neg)$ We have $\neg\neg\phi$ at a in Θ, the current block is an a-block and Δ is $D_{\Theta,i,k}$. To apply the induction hypothesis we need a set of indices which is $D_{\phi -_a \Theta,i,k}$, that is, relates to the extended branch. By Lemma 5.3 on the preceding page, the given set Δ is applicable. Thus we have $\vdash \phi -_a \Theta^\Delta$, where ϕ is unchanged since its index cannot occur in Δ. Moreover we have $\neg\neg\phi$ at a in Θ^Δ since $(\neg\neg\phi)^j = \neg\neg\phi$. Thus $(\neg\neg)$ applies and we conclude $\vdash \Theta^\Delta$.

The cases for $(\vee)$, $(\neg\vee)$ and $(@)$ are similar.

By Lemma 5.5 on the previous page and the fact that both j and k occur directly on blocks in Θ, the nominals in Θ and Θ^Δ are exactly the same. Thus, $(\Diamond)$ and GoTo also follow in a similar fashion.

Case $(\neg\Diamond)$ We have both $\neg\Diamond\phi$ and $\Diamond i'$ at a in Θ, the current block is an a-block and we know that Δ is $D_{\Theta,i,k}$. Let (w, w') be the index of the given $\Diamond i'$. There are two cases.

If $(w, w') \notin \Delta$ then $\Diamond i'$ is also at a in Θ^Δ and the case follows similarly to $(\neg\neg)$: The rule input is unchanged so we do not have to replace the output.

Otherwise, $(w, w') \in \Delta$ so by Lemma 5.2 on page 40, $i' = k$ and $(\Diamond i')^j = \Diamond j$. To account for this, we need to extend Δ to include the index of the output, $\neg @_k \phi$, to make sure it becomes $\neg @_j \phi$ such that the $(\neg\Diamond)$ rule justifies it. Let (v, v') be the index of the output. By Lemma 5.3 on the previous page, Δ is $D_{\neg @_k \phi -_a \Theta,i,k}$ and by the **Derived** case, so is $\{(v, v')\} \cup \Delta$.

Thus we apply the induction hypothesis at the extended index set, $\{(v, v')\} \cup \Delta$, and learn $\vdash (\neg @_k \phi)^j -_a \Theta^{j\{(v,v')\}\cup\Delta}$. By Remark 5.4 on the preceding page we have $\vdash \neg @_j \phi -_a \Theta^\Delta$. We can now apply the $(\neg\Diamond)$ rule and conclude the case.

ϕ	ϕ^j	$(\neg\phi)^j$
$\Diamond k$	$\Diamond j$	$\neg(\Diamond k)$
$\neg@_k\psi$	$\neg@_j\psi$	$\neg\neg@_k\psi$
$@_k\psi$	$@_k\psi$	$\neg@_j\psi$

Table 5.1: The j-replacements for complementary pairs.

1.	c	
2.	j	
3.	c	
4.	k	
5.	a	
6.	$\Diamond j$	ϕ^j
7.	$\neg\Diamond k$	$(\neg\phi)^j$
	$\vdots$	
8.	a	GoTo
9.	$\neg@_j k$	$(\neg\Diamond)$ 6, 7
10.	j	GoTo
11.	k	Nom 2, 4, 10
12.	$\neg k$	$(\neg@)$ 9, 10
	$\times$	

1.	c	
2.	j	
3.	c	
4.	k	
5.	a	
6.	$\neg@_j\psi$	ϕ^j
7.	$\neg\neg@_k\psi$	$(\neg\phi)^j$
	$\vdots$	
8.	a	GoTo
9.	$@_k\psi$	$(\neg\neg)$ 7
10.	j	GoTo
11.	k	Nom 2, 4, 10
12.	ψ	$(@)$ 9, 11
13.	$\neg\psi$	$(\neg@)$ 6, 10
	$\times$	

1.	c	
2.	j	
3.	c	
4.	k	
5.	a	
6.	$@_k\psi$	ϕ^j
7.	$\neg(@_j\psi)$	$(\neg\phi)^j$
	$\vdots$	
8.	j	GoTo
9.	k	Nom 2, 4, 8
10.	ψ	$(@)$ 6, 9
11.	$\neg\psi$	$(\neg@)$ 7, 8
	$\times$	

Figure 5.2: Closing extensions for the three interesting base cases.

Case ($\neg@$) We have $\neg@_a\phi$ at b in Θ, the current block is an a-block and we know that Δ is $D_{\Theta,i,k}$. Let (w, w') be the index of $\neg@_a\phi$ in Θ. There are two cases.
If $(w, w') \notin \Delta$ then, like before, the case follows similarly to ($\neg\neg$).
Otherwise, $(w, w') \in \Delta$ so by Lemma 5.2 on page 40, $a = k$ and $(\neg@_a\phi)^j = \neg@_j\phi$. By the induction hypothesis we know $\vdash \neg\phi -_a \Theta^\Delta$. We cannot apply the ($\neg@$) rule just yet since we have $\neg@_j\phi$ at b in Θ^Δ and the current block is a k-block, but by Theorem 4.16 on page 37 we get $\vdash \neg\phi -_k j -_k \Theta^\Delta$. The ($\neg@$) and Nom rules justify the extension as illustrated in Figure 5.3.

1.	c	
2.	j	
3.	c	
4.	k	
5.	b	
6.	$\neg@_j\phi$	
	$\vdots$	
7.	k	$(a = k)$
	$\vert$	
8.	j	Nom, 2, 4, 7
9.	$\neg\phi$	($\neg@$), 6, 8
	$\times$	

Figure 5.3: Closing ($\neg@$).

Case Nom We have both ϕ and i' at b in Θ and i' at a in Θ. The current block is an a-block and we know that Δ is $D_{\Theta,i,k}$. Let (w, w') be the index of the given ϕ. If $(w, w') \notin \Delta$ then the case follows similarly to ($\neg\neg$).

Otherwise, $(w, w') \in \Delta$ so the input to the rule gets replaced and we need to replace the output too. By Lemma 5.3 on page 41, Δ is $D_{\phi -_a \Theta,i,k}$ and by the **Copied** case, so is $\{(v, v')\} \cup \Delta$.

Thus we know from the induction hypothesis and Lemma 5.4 on page 41 that $\vdash \phi^j -_a \Theta^\Delta$. The remaining rule input stays the same on Θ^Δ, so the Nom rule applies and concludes the case. $\square$

5.5 Derivation

Theorem 5.7 (Bridge is admissible in ST) *If $\Diamond j$ is at i in Θ, j and k are both at a in Θ, the current block is an i-block and $\vdash \Diamond k -_i \Theta$, then $\vdash \Theta$.*

Proof. Let (v, v') be the index of the final $\Diamond k$ and let Δ be the set containing just that index. By the **Initial** case, Δ is $D_{\Diamond k -_i \Theta,i,k}$. Thus $\vdash (\Diamond k)^j -_i \Theta^\Delta$ by Lemma 5.6 on page 41. Then $\vdash \Diamond j -_i \Theta$ by Lemma 5.4 on page 41 and finally, due to the Nom rule, $\vdash \Theta$. $\square$

Chapter 6

Completeness

By now we have lifted all of the imposed termination restrictions and proved the Bridge rule to be admissible. This means that the synthetic completeness proof by Jørgensen et al. [JBBB16] applies to the calculus without modifications. In this chapter I will focus on its formalization and point out a small error in their definition of Hintikka sets.

The overall idea of the synthetic approach is as follows. First, we will classify certain sets of blocks as *Hintikka* sets and show that we can construct a model, called the *named model*, for any formula on a block in such a set. Next we will see how to extend any consistent set of blocks into a maximally consistent, saturated set using a *Lindenbaum-Henkin* construction. Finally we will see that such a set is a Hintikka set. Completeness then follows by contradiction: If ϕ is valid but $\neg\phi$ does not have a closing tableau then the set consisting of a single block with $\neg\phi$ on it is consistent and can be extended into a Hintikka set. Thus, a model for $\neg\phi$ can be obtained, contradicting the assumption that ϕ is valid [JBBB16].

6.1 Hintikka sets

Jørgensen et al. give a 13-part definition of when a set of blocks is a Hintikka set and argue that we should think of such a set H as an abstract version of an exhausted open tableau branch [JBBB16]. For example, requirement **(x)** says that if there is an i-block with $\neg\neg\phi$ on it in H then it should also have an i-block with ϕ on it. The corrected, formalized definition can be seen in Figure 6.1 on page 48. The correction is explained in section 6.1.2 on the next page.

Note that the blocks used by Jørgensen et al. are sets of formulas and so they often require these sets to be finite. I continue to work with lists of formulas which are always finite. Furthermore my blocks are named by construction so where they refer to a "finite named block" I will simply write "block."

Let H be a Hintikka set of blocks. The *names* of H are the nominals that appear as opening nominals in H. To build the named model over H we first need to consider equivalence classes of nominals. Nominals i and j are equivalent in H, written $i \sim_H j$, if there is an i-block in H with j on it. Or in Isabelle syntax:

definition *hequiv* :: ⟨*('a, 'b) block set* ⇒ *'b* ⇒ *'b* ⇒ *bool*⟩ **where**
  ⟨*hequiv H i j* ≡ *Nom j at i in' H*⟩

It follows from the Hintikka definition that this relation is an equivalence relation on the names in H:

abbreviation *hequiv-rel* :: ⟨*('a, 'b) block set* ⇒ *('b × 'b) set*⟩ **where**
  ⟨*hequiv-rel H* ≡ {(*i*, *j*) |*i j*. *hequiv H i j*}⟩

lemma *hequiv-rel*: ⟨*hintikka H* ⟹ *equiv* (*names H*) (*hequiv-rel H*)⟩
  using *hequiv-refl-rel hequiv-sym-rel hequiv-trans-rel* **by** (*rule equivI*)

I will denote by $|i|$ the set of nominals equivalent to i. The underlying Hintikka set is implicit. As shorthand, I will say that a formula occurs at $|i|$ in H if it occurs on some i'-block in H with $i' \in |i|$.

6.1.1 Named model

We are now ready to construct the named model of H. The worlds of the model are sets of equivalent nominals. The assignment maps a nominal i to $|i|$:

definition *assign* :: ‹$('a, 'b)$ *block set* $\Rightarrow$ $'b \Rightarrow 'b$ *set*› **where**
‹*assign H i* $\equiv$ *proj* (*hequiv-rel H*) *i*›

From world $|i|$ we can reach all and only worlds $|j|$ where $\Diamond j$ occurs at $|i|$ in H:

definition *reach* :: ‹$('a, 'b)$ *block set* $\Rightarrow$ $'b$ *set* $\Rightarrow$ $'b$ *set set*› **where**
‹*reach H is* $\equiv$ $\{$*assign H j* $|i\ j.\ i \in is \wedge (\Diamond$ *Nom j*$)$ *at i in' H*$\}$›

Finally, propositional symbol x is true at $|i|$ iff x occurs at $|i|$ in H:

definition *val* :: ‹$('a, 'b)$ *block set* $\Rightarrow$ $'b$ *set* $\Rightarrow$ $'a \Rightarrow$ *bool*› **where**
‹*val H is x* $\equiv \exists\, i \in is.$ *Pro x at i in' H*›

The *Hintikka block lemma* states if H is Hintikka and ϕ occurs on an i-block in H, then the named model models ϕ at $|i|$. For the induction we also need that $\neg\phi$ is *not* modeled. The lemma follows by induction over the complexity of the formula and the proof is omitted here [JBBB16, Lemma 3.1].

lemma *hintikka-model*:
assumes ‹*hintikka H*›
shows
‹*p at i in' H* $\Longrightarrow$ *Model* (*reach H*) (*val H*), *assign H*, *assign H i* $\models$ *p*›
‹$(\neg\ p)$ *at i in' H* $\Longrightarrow$ $\neg$ *Model* (*reach H*) (*val H*), *assign H*, *assign H i* $\models$ *p*›

6.1.2 Definition correction

Now that we understand the construction of the named model, let us consider the very first case of the definition by Jørgensen et al.:

(i) *If there is an i-block in H with an atomic formula a on it, then there is no i-block in H with* $\neg a$ *on it.*

Consider the following set of blocks where i and j are distinct nominals and x is a propositional symbol:

$$\{([i, x], j), ([j, \neg x], i)\} \tag{6.1}$$

```
definition hintikka :: ⟨('a, 'b) block set ⇒ bool⟩ where
  ⟨hintikka H ≡
    (∀ x i j. Nom j at i in' H ⟶ Pro x at j in' H ⟶
      ¬ (¬ Pro x) at i in' H)
  ∧
    (∀ a i. Nom a at i in' H ⟶ ¬ (¬ Nom a) at i in' H)
  ∧
    (∀ i j. (◇ Nom j) at i in' H ⟶ ¬ (¬ (◇ Nom j)) at i in' H)
  ∧
    (∀ p i. i ∈ nominals p ⟶ (∃ block ∈ H. p on block) ⟶
      (∃ ps. (ps, i) ∈ H))
  ∧
    (∀ i j. Nom j at i in' H ⟶ Nom i at j in' H)
  ∧
    (∀ i j k. Nom j at i in' H ⟶ Nom k at j in' H ⟶
      Nom k at i in' H)
  ∧
    (∀ i j k. (◇ Nom j) at i in' H ⟶ Nom k at j in' H ⟶
      (◇ Nom k) at i in' H)
  ∧
    (∀ i j k. (◇ Nom j) at i in' H ⟶ Nom k at i in' H ⟶
      (◇ Nom j) at k in' H)
  ∧
    (∀ p q i. (p ∨ q) at i in' H ⟶
      p at i in' H ∨ q at i in' H)
  ∧
    (∀ p q i. (¬ (p ∨ q)) at i in' H ⟶
      (¬ p) at i in' H ∧ (¬ q) at i in' H)
  ∧
    (∀ p i. (¬ ¬ p) at i in' H ⟶
      p at i in' H)
  ∧
    (∀ p i a. (@ i p) at a in' H ⟶
      p at i in' H)
  ∧
    (∀ p i a. (¬ (@ i p)) at a in' H ⟶
      (¬ p) at i in' H)
  ∧
    (∀ p i. (∄ a. p = Nom a) ⟶ (◇ p) at i in' H ⟶
      (∃ j. (◇ Nom j) at i in' H ∧ (@ j p) at i in' H))
  ∧
    (∀ p i j. (¬ (◇ p)) at i in' H ⟶ (◇ Nom j) at i in' H ⟶
      (¬ (@ j p)) at i in' H)⟩
```

Figure 6.1: Hintikka definition in Isabelle/HOL.

We have i and x on a j-block and j and $\neg x$ on an i-block, so there is no violation of **(i)**. It is easy to verify that this set also satisfies the rest of the Hintikka requirements. However, i and j are equivalent, $i \sim j$, so $|i| = |j|$ and according to the valuation of the named model x is true at $|i|$ since it occurs at i. But according to the Hintikka block lemma, $\neg x$ is also true at $|i|$ since it occurs on a j-block, that is, also at $|i|$.

To remedy this contradiction, consider again the intuition that Hintikka sets should correspond to exhausted open tableau branches. The set in equation 6.1 on page 47 breaks this intuition since the Nom rule can be applied to extend either the j-block with $\neg x$ or the i-block with x. The branch is not exhausted (and in both cases the resulting blocks violate requirement **(i)**). To account for the Nom rule the requirement should be something like:

(i') If there is an i-block in H with j on it and a j-block in H with a propositional symbol x on it, then there is no i-block in H with $\neg x$ on it.

This version is what has been formalized. Note that the original definition is obtained by setting $i = j$. For nominals the original definition suffices.

The original requirement **(ix)** states that if there is an i-block in H with $\neg(\phi \vee \psi)$ on it then there is an i-block in H with both $\neg\phi$ and $\neg\psi$ on it [JBBB16]. I have relaxed this to say that both $\neg\phi$ and $\neg\psi$ must occur at i in H but not necessarily on the same block. Either variant suffices.

6.2 Lindenbaum-Henkin

In this section we will see how to extend a consistent set of blocks to be maximal and saturated. A set of blocks is consistent if no finite subset of it can be closed:

definition *consistent* :: ‹(*'a*, *'b*) *block set* ⇒ *bool*› **where**
  ‹*consistent S* ≡ ∄*S'*. *set S'* ⊆ *S* ∧ ⊢ *S'*›

6.2.1 Construction

Note that for countable universes of propositional symbols and nominals the set of formulas is countable. Thus, lists of formulas are countable and so are blocks.

This means that blocks over these universes can be enumerated. Isabelle can prove this countability automatically for defined datatypes:

```
instance fm :: (countable, countable) countable
  by countable-datatype
```

Starting from a consistent set of blocks we are going to enumerate every block and add it to our set if doing so preserves consistency. If ϕ is not a nominal then I will call $\Diamond\phi$ a proper diamond. Some of the added blocks may contain proper diamonds and we will also add blocks that witness these.

Definition 6.1 ($\Diamond$-Witness) *Let B be an a-block about to be added to our growing set S. The a-block B' is a $\Diamond$-witness to B if for all proper diamonds $\Diamond\phi$ on B, B' contains $@_i\phi$ and $\Diamond i$ for i fresh to B, S and the rest of B'.*

In the formalization, a function *witness* constructs the $\Diamond$-witness of a block given a set of already used nominals. The extension is then given in two steps. First as a primitive recursive function that extends the set with only the first n blocks of the enumeration:

```
primrec extend ::
  ‹('a, 'b) block set ⇒ (nat ⇒ ('a, 'b) block) ⇒ nat ⇒ ('a, 'b) block set› where
  ‹extend S f 0 = S›
| ‹extend S f (Suc n) =
    (if ¬ consistent ({f n} ∪ extend S f n)
     then extend S f n
     else
      let used = (⋃ block ∈ {f n} ∪ extend S f n. block-nominals block)
      in {f n, witness (f n) used} ∪ extend S f n)›
```

At 0 the given set is returned unmodified. Otherwise there are two cases for extending the recursively extended set. If block $f\ n$ causes inconsistency then it is not added but if consistency is preserved then it is added alongside a $\Diamond$-witness.

Jørgensen et al. give three cases by distinguishing whether the added block contains a $\Diamond$-formula or not [JBBB16]. This saves them from adding empty $\Diamond$-witness blocks when the enumerated block does not contain a $\Diamond$-formula, but since the empty blocks will be part of the full enumeration anyway, I have chosen the two-case definition.

To obtain the maximally consistent set, we take the union of all extended sets:

definition *Extend* ::
  ‹(′a, ′b) block set ⇒ (nat ⇒ (′a, ′b) block) ⇒ (′a, ′b) block set› **where**
  ‹$Extend\ S\ f \equiv (\bigcup n.\ extend\ S\ f\ n)$›

Jørgensen et al. call this the Lindenbaum-Henkin construction. In the following, I will call its output the *constructed set.*

6.2.2 Properties

For brevity I will not go into the proofs of each of these; see Jørgensen et al. for details [JBBB16, Lemma 3.4]. All the properties will assume that the starting set S to be extended contains a finite number of nominals. This is to ensure that fresh nominals are available for the ◇-witnesses.

Consistency The constructed set is consistent:

lemma *consistent-Extend*:
  fixes S :: ‹(′a, ′b) block set›
  assumes *inf*: ‹infinite (UNIV :: ′b set)› **and**
    ‹consistent S› ‹finite ($\bigcup$ (block-nominals ‘ S))›
  shows ‹consistent (Extend S f)›

Maximality A consistent set of blocks is *maximal* if any proper extension is inconsistent.

definition *maximal* :: ‹(′a, ′b) block set ⇒ bool› **where**
  ‹maximal S ≡ consistent S ∧
    ($\forall$ block. block $\notin S \longrightarrow \neg$ consistent ({block} ∪ S))›

The constructed set is maximal:

lemma *maximal-Extend*:
  fixes S :: ‹(′a, ′b) block set›
  assumes *inf*: ‹infinite (UNIV :: ′b set)› **and**
    ‹consistent S› ‹finite ($\bigcup$ (block-nominals ‘ S))› ‹surj f›
  shows ‹maximal (Extend S f)›

Saturation A set of blocks is *saturated* if every proper diamond is witnessed:

definition *saturated* :: ‹(*'a*, *'b*) *block set* ⇒ *bool*› **where**
‹*saturated* $S \equiv \forall p\ i.\ (\Diamond\ p)$ *at i in' S* $\longrightarrow (\nexists a.\ p =$ *Nom a*$) \longrightarrow$
$(\exists j.\ (@\ j\ p)$ *at i in' S* $\wedge\ (\Diamond$ *Nom j*$)$ *at i in' S*$)$›

The constructed set is saturated:

lemma *saturated-Extend*:
fixes *S* :: ‹(*'a*, *'b*) *block set*›
assumes *inf*: ‹*infinite* (*UNIV* :: *'b set*)› **and**
‹*consistent S*› ‹*finite* ($\bigcup$ (*block-nominals* ‘ *S*))› ‹*surj f*›
shows ‹*saturated* (*Extend S f*)›

6.3 Smullyan-Fitting

The above properties, allows us to show that the Lindenbaum-Henkin constructed set is Hintikka without worrying about the particular details of the construction. See Jørgensen et al. for proofs of some of the cases [JBBB16, Lemma 3.5].

lemma *hintikka-Extend*:
fixes *S* :: ‹(*'a*, *'b*) *block set*›
assumes *inf*: ‹*infinite* (*UNIV* :: *'b set*)› **and**
‹*maximal S*› ‹*consistent S*› ‹*saturated S*›
shows ‹*hintikka S*›

6.4 Result

Finally let us go through the formalized lemma of the final completeness result. The result applies to any formula p defined over a countable universe of propositional symbols and a countably infinite universe of nominals. If the formula is valid then the branch consisting of a single block with $\neg p$ on it can be closed starting from a single coin:

theorem *completeness*:
fixes *p* :: ‹(*'a* :: *countable*, *'b* :: *countable*) *fm*›

```
  assumes
    inf: ‹infinite (UNIV :: 'b set)› and
    valid: ‹∀(M :: ('b set, 'a) model) g w. M, g, w ⊨ p›
  shows ‹1 ⊢ [([¬ p], i)]›
```

We start by proving that a derivation from some number of coins exists and then reduce it to a single-coin derivation afterwards. The proof goes by contradiction:

```
proof –
  have ‹⊢ [([¬ p], i)]›
  proof (rule ccontr)
    assume ‹¬ ⊢ [([¬ p], i)]›
```

Since we assume the branch cannot be closed then it is consistent:

```
    then have *: ‹consistent {([¬ p], i)}›
      unfolding consistent-def using ST-struct inf
      by (metis empty-set list.simps(15))
```

First, define a syntactic abbreviation, *?S*, for the Lindenbaum-Henkin constructed set where *from-nat* is the function that enumerates the blocks. Then note that the original set contains a finite number of nominals:

```
    let ?S = ‹Extend {([¬ p], i)} from-nat›
    have ‹finite {([¬ p], i)}›
      by simp
    then have fin: ‹finite (⋃ (block-nominals ‘ {([¬ p], i)}))›
      using finite-nominals-set by blast
```

Thus the constructed set is consistent, maximal, saturated and hence Hintikka:

```
    have ‹consistent ?S›
      using consistent-Extend inf * fin by blast
    moreover have ‹maximal ?S›
      using maximal-Extend inf * fin by fastforce
    moreover have ‹saturated ?S›
      using saturated-Extend inf * fin by fastforce
    ultimately have ‹hintikka ?S›
      using hintikka-Extend inf by blast
```

Moreover, the original block is still contained in the constructed set and $\neg p$ occurs on it:

```
    moreover have ‹([¬ p], i) ∈ ?S›
      using Extend-mem by blast
    moreover have ‹(¬ p) on ([¬ p], i)›
      by simp
```

Thus we can obtain a countermodel for p but, as desired, this contradicts the assumption that it is valid:

```
    ultimately have ‹¬ Model (reach ?S) (val ?S), assign ?S, assign ?S i ⊨ p›
      using hintikka-model by fast
    then show False
      using valid by blast
  qed
```

Thus the branch closes and we can use Corollary 4.5 on page 27 to show the thesis, that it closes when starting from a single coin:

```
  then show ?thesis
    using ST-one by blast
qed
```

Chapter 7

Conclusion

This chapter concludes by first taking a look at how the Nom rule might be restricted to obtain a terminating system, then I discuss related and future work and finally I give a brief recap of the presented work.

7.1 On Termination

As mentioned in Chapter 3 it is sometimes possible to apply the presented rules infinitely due to the Nom rule. The problem is the interplay between ($\Diamond$) which, given a formula $\Diamond\phi$ on an a-block introduces a fresh nominal, i, ($\neg\Diamond$) which reveals information about accessible worlds and the rule Nom. The formula $\psi = \Box(\Diamond\top \land (a \land \top))$ at a along with an accessibility formula $\Diamond i$ effectively produces a fresh accessible world at i and reveals that i is equivalent to a. Thus we can GoTo i, apply ($\Diamond$), copy ψ to i with Nom, apply ($\neg\Diamond$) again and repeat ad infinitum [BBBJ17, Figure 8].

To prevent this issue, Blackburn et al. say that in this case i was *generated by a* and note that Nom should be restricted to not copy formulas to a block type that was generated by the source block type [BBBJ17]. They achieve this with a notion of quasi-root subformulas and a splitting of the Nom rule. Their definition of a quasi-root subformula assumes a single starting root formula and would have to be generalized to be formalized conveniently. I propose the following alternative approach that very directly treats generated nominals specially.

Associate a *tag* with each nominal on the branch: For a nominal a denote its tag by $\text{tag}(a)$. Two rules are modified. When a fresh nominal i is introduced by $(\Diamond)$ on an a-block, let its tag be one greater: $\text{tag}(i) = \text{tag}(a) + 1$. To copy a formula from a b-block to an a-block using Nom, require that $\text{tag}(a) \leq \text{tag}(b)$. Note that tags are never modified since new tags are only associated with fresh nominals.

If i is generated by j then $\text{tag}(i) > \text{tag}(j)$ and copying from a j-block to an i-block is disqualified but copying in the other direction is fine: We are forced to only copy in the "right direction" towards the older nominals. For an internalized perspective on tags, let the tag of a nominal match its order of appearance on the branch; we are only allowed to copy to nominals that appear earlier.

It remains to be shown that if a branch can be closed by copying in the "wrong direction" then it can also be closed by not doing so. That is, that the restriction preserves completeness. This work has not been formalized but I sketch a proof below. The overall idea is that any work done on j after copying "wrongly" from i to j could have been done on i instead.

First, note that we can still lift R1 in the presence of this restriction but that lifting collapsing substitution is no longer straight-forward. However, it is possible to show a version of the rearranging Theorem 4.15 on page 35 that does not introduce new formulas without lifting any restrictions, since formulas then remain new (R1) and unwitnessed (R2) on the rearranged branch.

Define an index set Δ_k over the branch Θ such that for all $(v, v') \in \Delta_k$, $\Theta(v)$ is some i-block, $\Theta(v, v')$ is some ϕ, ϕ occurs at k in Θ, $\text{tag}(i) > \text{tag}(k)$ and there exists a shared nominal that occurs at both i and k in Θ. Omitting these indices from a branch corresponds to a generalized Nom rule in the wrong direction. It is this operation that I will argue is admissible. Given a branch Θ and a set Δ_k defined over Θ, let Θ_{Δ_k} be Θ with all occurrences in Δ omitted. Moreover, allow a designated k-block in Θ_{Δ_k} to be extended by formulas that *already appear at an equivalent nominal on the branch*. Given a tableau for a branch Θ with a designated k-block and an index set Δ_k I argue by induction over the construction of Θ that Θ_{Δ_k} with an arbitrary weakening of the k-block has a closing tableau, given that the weakening satisfies the emphasized assumption.

For the closing condition, ϕ and $\neg\phi$ both appear at i in Θ and either ϕ or $\neg\phi$ may have been removed in Θ_{Δ_k}. If neither is removed the branch closes immediately and if both are removed then they both appear at k so the branch also closes. Therefore, assume that one of them is removed and call it ψ. Rearrange the branch to make the designated k-block the current block and extend it with ψ using Nom. By the definition of Δ_k, the tag of i is greater than that of k so this extension is legal. The obtained branch closes so Θ_{Δ_k} does too.

For the $(\neg\neg)$ case, $\neg\neg\phi$ occurs at a in Θ. Assume that it has been omitted in Θ_{Δ_k} as otherwise the case is trivial. Then apply the induction hypothesis at the designated k-block extended by ϕ and at Δ_k extended by the index of the ϕ-extension on the current block. The latter extension is legal exactly because the designated k-block is extended in tandem which corresponds to working on the k-block instead of the a-block. To get rid of ϕ on the k-block, make it the current block and apply $(\neg\neg)$. The thesis follows by re-rearranging the blocks.

In cases with multiple input to a rule where only some has been omitted, the designated k-block can be extended with the remaining input using Nom.

For the $(\Diamond)$ case, we know that ϕ is not witnessed at the current block type, but it may be witnessed at k. To prevent this, the notion of when a formula is *witnessed* needs to be strengthened to account for equivalent nominals such that ϕ is witnessed at a in Θ if there exists a block type in Θ equivalent to a that directly witnesses ϕ. Since the input $\Diamond\phi$ is only omitted if the current block type is equivalent to k, we know that ϕ is not witnessed at k either. Moreover, lifting this stronger restriction should still be possible using the presented technique.

In the Nom case we know that the b-block that is the source of the extension is not a k-block due to the restrictions on the tags, so the same technique as for $(\neg\neg)$ should apply.

The GoTo case is trivial since the nominals on Θ_{Δ_k} are the same as on Θ.

By taking the index set to only include a single formula extension and not weakening the designated k-block, a Nom rule in the wrong direction is obtained. Coupled with the built-in Nom rule we then have the unrestricted version.

Note that this proof sketch does not make any assumptions on the tags so any scheme for those will work.

7.2 Related Work

Blackburn et al. point to a number of proof systems for hybrid logic, including tableau, natural deduction, resolution and Hilbert systems, all based on labeling [BBBJ17]. The use of labeling gives a global proof style where formulas are nested below some prefix like a satisfaction operator. The work by Bolander and Blackburn on showing termination of an internalized labeled tableau system is notable [BB07].

The local proof style used by the presented calculus originates in Seligman's natural deduction and sequent calculus systems [Sel97, Sel01]. Instead of a copying Nom rule, Seligman allows a collapsing substitution of equivalent nominals and the equivalent of GoTo is obtained by packing every formula into a satisfaction statement, switching world and then unpacking the relevant satisfaction statements. Braüner avoids most of this packing and unpacking in his natural deduction system by using a Term rule where you explicitly state which satisfaction statements to make available in the subproof [Bra13]. The notion of blocks, and thus the Nom and GoTo rules used here, originates in the work on a Seligman-style tableau system by Blackburn et al. [BBBJ17].

Linker formalizes in Isabelle/HOL a semantic embedding of a spatio-temporal multi-modal logic designed for reasoning about motorway traffic which includes a hybrid logic-inspired *at*-operator [Lin17]. Linker and Hilscher give a sound labeled natural deduction proof system for a version of the logic without the hybrid extension [LH15]. To my knowledge, no other formalization of hybrid logic in Isabelle has been published. Doczkal and Smolka formalize hybrid logic with nominals but no special operators in constructive type theory using the proof assistant Coq. They do not define a proof system but give algorithmic proofs of small model theorems and computational decidability of satisfiability, validity, and equivalence of formulas [DS11]. In the proof assistant Agda, the closest thing to a formalization of hybrid logic is an embedding of the programming language ML5 by Licata and Harper. ML5 is is based on a Curry-Howard correspondence with the modal logic S5 and includes a modality that is a composition of quantification over worlds and the satisfaction operator [LH10]. The present work appears to be the first proof system for hybrid logic with a formalized completeness proof.

Formalizations of completeness proofs in Isabelle exist for, among others, a tableau system and a one-sided sequent calculus for first-order logic [Fro19], a natural deduction system for first-order logic [Ber07], a Hilbert system for epistemic logic [Fro18] and the first-order resolution calculus [Sch16]. Blanchette et al. give abstract proofs of soundness and completeness that can be instantiated for a range of Gentzen and tableau systems for various flavors of first-order logic [BPT17].

7.3 Future Work

Formalizing the above restriction of Nom, or an equivalent one, is obvious future work. I would then like to show that the system is terminating using a direct decreasing-length argument in the style of Bolander and Blackburn [BB07] instead of via translation as done by Blackburn et al. [BBBJ17]. Given a terminating system it should be possible to define an algorithm based on the calculus and verify soundness, completeness and termination of it. Isabelle supports code extraction to obtain an executable prover based on the verified algorithm.

Extending the system to handle quantification over worlds or the down-arrow binder [Bra17] are also options, but then the logic is no longer decidable and we need to give up on having a terminating system.

It would be interesting to come up with an internalized restriction on GoTo in place of the current coin system since the notion of coins does not belong to the object language.

7.4 Conclusion

I have presented a variant of an existing Seligman-style tableau system for basic hybrid logic and formalized it in the proof assistant Isabelle/HOL. The presented system is not terminating but does incorporate a number of restrictions to rule out sources of non-termination. Some of these restrictions have been reformulated to be more amenable to formalization and I have shown how to lift each of them by working within the proof system instead of via translation into a different system. I have also shown the full Bridge rule to be admissible using the notion of a descendant relation on a set of indices into the branch. Finally I have formalized completeness of the system via a synthetic approach that has previously only been applied to the unrestricted system.

Bibliography

[BB07] Thomas Bolander and Patrick Blackburn. Termination for Hybrid Tableaus. *Journal of Logic and Computation*, 17(3):517–554, 2007.

[BBBJ17] Patrick Blackburn, Thomas Bolander, Torben Braüner, and Klaus Frovin Jørgensen. Completeness and Termination for a Seligman-style Tableau System. *Journal of Logic and Computation*, 27(1):81–107, 2017.

[Ber07] Stefan Berghofer. First-Order Logic According to Fitting. *Archive of Formal Proofs*, August 2007. `http://isa-afp.org/entries/FOL-Fitting.html`, Formal proof development.

[Bla00] Patrick Blackburn. Representation, Reasoning, and Relational Structures: a Hybrid Logic Manifesto. *Logic Journal of the IGPL*, 8(3):339–365, 2000.

[BPT17] Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Soundness and Completeness Proofs by Coinductive Methods. *Journal of Automated Reasoning*, 58(1):149–179, 2017.

[Bra13] Torben Braüner. Hybrid-Logical Reasoning in False-Belief Tasks. In *Proceedings of Fourteenth Conference on Theoretical Aspects of Rationality and Knowledge (TARK)*, pages 186–195, India, 2013. Institute of Mathematical Sciences.

[Bra17] Torben Braüner. Hybrid Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2017 edition, 2017.

[DS11] Christian Doczkal and Gert Smolka. Constructive Formalization of Hybrid Logic with Eventualities. In *Certified Programs and Proofs - First International Conference, CPP 2011, Kenting, Taiwan, December 7-9, 2011. Proceedings*, pages 5–20, 2011.

[Fro18] Andreas Halkjær From. Epistemic Logic. *Archive of Formal Proofs*, October 2018. `http://isa-afp.org/entries/Epistemic_Logic.html`, Formal proof development.

[Fro19] Andreas Halkjær From. A Sequent Calculus for First-Order Logic. *Archive of Formal Proofs*, July 2019. `http://isa-afp.org/entries/FOL_Seq_Calc1.html`, Formal proof development.

[HGS07] Ian Horrocks, Birte Glimm, and Ulrike Sattler. Hybrid Logics and Ontology Languages. *Electronic Notes in Theoretical Computer Science*, 174(6):3–14, 2007.

[JBBB16] Klaus Frovin Jørgensen, Patrick Blackburn, Thomas Bolander, and Torben Braüner. Synthetic Completeness Proofs for Seligman-style Tableau Systems. In *Advances in Modal Logic 11, proceedings of the 11th conference on Advances in Modal Logic, held in Budapest, Hungary, August 30 - September 2, 2016*, pages 302–321, 2016.

[LH10] Daniel R. Licata and Robert Harper. A Monadic Formalization of ML5. In *Proceedings 5th International Workshop on Logical Frameworks and Meta-languages: Theory and Practice, LFMTP 2010, Edinburgh, UK, 14th July 2010*, pages 69–83, 2010.

[LH15] Sven Linker and Martin Hilscher. Proof Theory of a Multi-Lane Spatial Logic. *Logical Methods in Computer Science*, 11(3), 2015.

[Lin17] Sven Linker. Hybrid Multi-Lane Spatial Logic. *Archive of Formal Proofs*, November 2017. `http://isa-afp.org/entries/Hybrid_Multi_Lane_Spatial_Logic.html`, Formal proof development.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[Rou97] William C. Rounds. Feature Logics. In *Handbook of Logic and Language*, pages 475–533. 1997.

[Sch16] Anders Schlichtkrull. The Resolution Calculus for First-Order Logic. *Archive of Formal Proofs*, June 2016. `http://isa-afp.org/entries/Resolution_FOL.html`, Formal proof development.

[Sel97] Jerry Seligman. The Logic of Correct Description. In Marteen de Rijke, editor, *Advances in Intensional Logic*, volume 7 of *Applied Logic Series*, pages 107–135. Kluwer, 1997.

[Sel01] Jerry Seligman. Internalization: The Case of Hybrid Logics. *Journal of Logic and Computation*, 11(5):671–689, 2001.

[WN04] Martin Wildmoser and Tobias Nipkow. Certifying Machine Code Safety: Shallow versus Deep Embedding. In *Theorem Proving in Higher Order Logics, 17th International Conference, TPHOLs 2004, Park City, Utah, USA, September 14-17, 2004, Proceedings*, pages 305–320, 2004.