



Formalizing a Seligman-Style Tableau System for Hybrid Logic

From, Asta Halkjær

Published in:
Archive of Formal Proofs

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

[Link back to DTU Orbit](#)

Citation (APA):
From, A. H. (2021). Formalizing a Seligman-Style Tableau System for Hybrid Logic. *Archive of Formal Proofs*, 2021.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Formalizing a Seligman-Style Tableau System for Hybrid Logic

Asta Halkjær From

December 14, 2021

Abstract

This work is a formalization of soundness and completeness proofs for a Seligman-style tableau system for hybrid logic. The completeness result is obtained via a synthetic approach using maximally consistent sets of tableau blocks. The formalization differs from previous work [1, 2] in a few ways. First, to avoid the need to backtrack in the construction of a tableau, the formalized system has no unnamed initial segment, and therefore no Name rule. Second, I show that the full Bridge rule is admissible in the system. Third, I start from rules restricted to only extend the branch with new formulas, including only witnessing diamonds that are not already witnessed, and show that the unrestricted rules are admissible. Similarly, I start from simpler versions of the @-rules and show that these are sufficient. The GoTo rule is restricted using a notion of potential such that each application consumes potential and potential is earned through applications of the remaining rules. I show that if a branch can be closed then it can be closed starting from a single unit. Finally, Nom is restricted by a fixed set of allowed nominals. The resulting system should be terminating.

Preamble

The formalization was part of the author's MSc thesis in Computer Science and Engineering at the Technical University of Denmark (DTU).

Supervisors:

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

Contents

1	Syntax	3
2	Semantics	4
2.1	Examples	4
3	Tableau	5
4	Soundness	7
5	No Detours	8
5.1	Free GoTo	9
6	Indexed Mapping	9
6.1	Indexing	9
6.2	Mapping	10
7	Duplicate Formulas	12
7.1	Removable indices	12
7.2	Omitting formulas	13
7.3	Induction	16
7.4	Unrestricted rules	16
8	Substitution	17
8.1	Unrestricted (\diamond) rule	20
9	Structural Properties	20
10	Bridge	22
10.1	Replacing	22
10.2	Descendants	23
10.3	Induction	25
10.4	Derivation	25
11	Completeness	25
11.1	Hintikka	25
11.1.1	Named model	27
11.2	Lindenbaum-Henkin	29
11.2.1	Consistency	30
11.2.2	Maximality	32
11.2.3	Saturation	32
11.3	Smullyan-Fitting	33
11.4	Result	33
	References	34

theory *Hybrid-Logic* imports *HOL-Library.Countable* begin

1 Syntax

datatype $\langle 'a, 'b \rangle$ *fm*
 = *Pro* $'a$
 | *Nom* $'b$
 | *Neg* $\langle 'a, 'b \rangle$ *fm* $\langle \neg \rightarrow [40] 40 \rangle$
 | *Dis* $\langle 'a, 'b \rangle$ *fm* $\langle 'a, 'b \rangle$ (**infixr** $\langle \vee \rangle$ 30)
 | *Dia* $\langle 'a, 'b \rangle$ *fm* $\langle \diamond \rightarrow 10 \rangle$
 | *Sat* $'b$ $\langle 'a, 'b \rangle$ *fm* $\langle @ \rightarrow 10 \rangle$

We can give other connectives as abbreviations.

abbreviation *Top* $\langle \top \rangle$ **where**
 $\langle \top \equiv (\text{undefined} \vee \neg \text{undefined}) \rangle$

abbreviation *Con* (**infixr** $\langle \wedge \rangle$ 35) **where**
 $\langle p \wedge q \equiv \neg (\neg p \vee \neg q) \rangle$

abbreviation *Imp* (**infixr** $\langle \longrightarrow \rangle$ 25) **where**
 $\langle p \longrightarrow q \equiv \neg (p \wedge \neg q) \rangle$

abbreviation *Box* $\langle \square \rightarrow 10 \rangle$ **where**
 $\langle \square p \equiv \neg (\diamond \neg p) \rangle$

primrec *nominals* :: $\langle 'a, 'b \rangle$ *fm* \Rightarrow $'b$ *set* **where**
 $\langle \text{nominals } (\text{Pro } x) = \{ \} \rangle$
 | $\langle \text{nominals } (\text{Nom } i) = \{ i \} \rangle$
 | $\langle \text{nominals } (\neg p) = \text{nominals } p \rangle$
 | $\langle \text{nominals } (p \vee q) = \text{nominals } p \cup \text{nominals } q \rangle$
 | $\langle \text{nominals } (\diamond p) = \text{nominals } p \rangle$
 | $\langle \text{nominals } (@ i p) = \{ i \} \cup \text{nominals } p \rangle$

primrec *sub* :: $\langle 'b \Rightarrow 'c \rangle \Rightarrow \langle 'a, 'b \rangle$ *fm* $\Rightarrow \langle 'a, 'c \rangle$ *fm* **where**
 $\langle \text{sub } - (\text{Pro } x) = \text{Pro } x \rangle$
 | $\langle \text{sub } f (\text{Nom } i) = \text{Nom } (f i) \rangle$
 | $\langle \text{sub } f (\neg p) = (\neg \text{sub } f p) \rangle$
 | $\langle \text{sub } f (p \vee q) = (\text{sub } f p \vee \text{sub } f q) \rangle$
 | $\langle \text{sub } f (\diamond p) = (\diamond \text{sub } f p) \rangle$
 | $\langle \text{sub } f (@ i p) = (@ (f i) (\text{sub } f p)) \rangle$

lemma *sub-nominals*: $\langle \text{nominals } (\text{sub } f p) = f \text{ ' nominals } p \rangle$
 $\langle \text{proof} \rangle$

lemma *sub-id*: $\langle \text{sub } id p = p \rangle$
 $\langle \text{proof} \rangle$

lemma *sub-upd-fresh*: $\langle i \notin \text{nominals } p \Longrightarrow \text{sub } (f(i := j)) p = \text{sub } f p \rangle$
 $\langle \text{proof} \rangle$

2 Semantics

Type variable $'w$ stands for the set of worlds and $'a$ for the set of propositional symbols. The accessibility relation is given by R and the valuation by V . The mapping from nominals to worlds is an extra argument g to the semantics.

datatype $\langle 'w, 'a \rangle \text{ model} =$
 $\text{Model } (R: \langle 'w \Rightarrow 'w \text{ set} \rangle) (V: \langle 'w \Rightarrow 'a \Rightarrow \text{bool} \rangle)$

primrec semantics

$:: \langle ('w, 'a) \text{ model} \Rightarrow ('b \Rightarrow 'w) \Rightarrow 'w \Rightarrow ('a, 'b) \text{ fm} \Rightarrow \text{bool} \rangle$
 $\langle \langle -, -, - \models \rightarrow [50, 50, 50] 50 \rangle \text{ where}$
 $\langle (M, -, w \models \text{Pro } x) = V M w x \rangle$
 $| \langle (-, g, w \models \text{Nom } i) = (w = g i) \rangle$
 $| \langle (M, g, w \models \neg p) = (\neg M, g, w \models p) \rangle$
 $| \langle (M, g, w \models (p \vee q)) = ((M, g, w \models p) \vee (M, g, w \models q)) \rangle$
 $| \langle (M, g, w \models \diamond p) = (\exists v \in R M w. M, g, v \models p) \rangle$
 $| \langle (M, g, - \models @ i p) = (M, g, g i \models p) \rangle$

lemma $\langle M, g, w \models \top \rangle$
 $\langle \text{proof} \rangle$

lemma semantics-fresh:

$\langle i \notin \text{nominals } p \implies (M, g, w \models p) = (M, g(i := v), w \models p) \rangle$
 $\langle \text{proof} \rangle$

2.1 Examples

abbreviation is-named $:: \langle ('w, 'b) \text{ model} \Rightarrow \text{bool} \rangle \text{ where}$
 $\langle \text{is-named } M \equiv \forall w. \exists a. V M a = w \rangle$

abbreviation reflexive $:: \langle ('w, 'b) \text{ model} \Rightarrow \text{bool} \rangle \text{ where}$
 $\langle \text{reflexive } M \equiv \forall w. w \in R M w \rangle$

abbreviation irreflexive $:: \langle ('w, 'b) \text{ model} \Rightarrow \text{bool} \rangle \text{ where}$
 $\langle \text{irreflexive } M \equiv \forall w. w \notin R M w \rangle$

abbreviation symmetric $:: \langle ('w, 'b) \text{ model} \Rightarrow \text{bool} \rangle \text{ where}$
 $\langle \text{symmetric } M \equiv \forall v w. w \in R M v \longleftrightarrow v \in R M w \rangle$

abbreviation asymmetric $:: \langle ('w, 'b) \text{ model} \Rightarrow \text{bool} \rangle \text{ where}$
 $\langle \text{asymmetric } M \equiv \forall v w. \neg (w \in R M v \wedge v \in R M w) \rangle$

abbreviation transitive $:: \langle ('w, 'b) \text{ model} \Rightarrow \text{bool} \rangle \text{ where}$
 $\langle \text{transitive } M \equiv \forall v w x. w \in R M v \wedge x \in R M w \longrightarrow x \in R M v \rangle$

abbreviation universal $:: \langle ('w, 'b) \text{ model} \Rightarrow \text{bool} \rangle \text{ where}$
 $\langle \text{universal } M \equiv \forall v w. v \in R M w \rangle$

lemma $\langle \text{irreflexive } M \implies M, g, w \models @ i \neg (\Diamond \text{Nom } i) \rangle$
 $\langle \text{proof} \rangle$

We can automatically show some characterizations of frames by pure axioms.

lemma $\langle \text{irreflexive } M = (\forall g w. M, g, w \models @ i \neg (\Diamond \text{Nom } i)) \rangle$
 $\langle \text{proof} \rangle$

lemma $\langle \text{asymmetric } M = (\forall g w. M, g, w \models @ i (\Box \neg (\Diamond \text{Nom } i))) \rangle$
 $\langle \text{proof} \rangle$

lemma $\langle \text{universal } M = (\forall g w. M, g, w \models \Diamond \text{Nom } i) \rangle$
 $\langle \text{proof} \rangle$

3 Tableau

A block is defined as a list of formulas paired with an opening nominal. The opening nominal is not necessarily in the list. A branch is a list of blocks.

type-synonym $\langle 'a, 'b \text{ block} = \langle 'a, 'b \text{ fm list} \times 'b \rangle$

type-synonym $\langle 'a, 'b \text{ branch} = \langle 'a, 'b \text{ block list} \rangle$

abbreviation $\text{member-list} :: \langle 'a \Rightarrow 'a \text{ list} \Rightarrow \text{bool} \rangle (\langle - \in. - \rangle [51, 51] 50)$ **where**
 $\langle x \in. xs \equiv x \in \text{set } xs \rangle$

The predicate *on* presents the opening nominal as appearing on the block.

primrec $\text{on} :: \langle 'a, 'b \text{ fm} \Rightarrow 'a, 'b \text{ block} \Rightarrow \text{bool} \rangle (\langle - \text{ on} - \rangle [51, 51] 50)$ **where**
 $\langle p \text{ on } (ps, i) = (p \in. ps \vee p = \text{Nom } i) \rangle$

syntax

$\text{-Ballon} :: \langle \text{pttrn} \Rightarrow 'a \text{ set} \Rightarrow \text{bool} \Rightarrow \text{bool} \rangle (\langle (\exists \forall (-/\text{on-}) ./ -) \rangle [0, 0, 10] 10)$

$\text{-Bexon} :: \langle \text{pttrn} \Rightarrow 'a \text{ set} \Rightarrow \text{bool} \Rightarrow \text{bool} \rangle (\langle (\exists \exists (-/\text{on-}) ./ -) \rangle [0, 0, 10] 10)$

translations

$\forall p \text{ on } A. P \rightarrow \forall p. p \text{ on } A \rightarrow P$

$\exists p \text{ on } A. P \rightarrow \exists p. p \text{ on } A \wedge P$

abbreviation $\text{list-nominals} :: \langle 'a, 'b \text{ fm list} \Rightarrow 'b \text{ set} \rangle$ **where**

$\langle \text{list-nominals } ps \equiv (\bigcup p \in \text{set } ps. \text{nominals } p) \rangle$

primrec $\text{block-nominals} :: \langle 'a, 'b \text{ block} \Rightarrow 'b \text{ set} \rangle$ **where**

$\langle \text{block-nominals } (ps, i) = \{i\} \cup \text{list-nominals } ps \rangle$

definition $\text{branch-nominals} :: \langle 'a, 'b \text{ branch} \Rightarrow 'b \text{ set} \rangle$ **where**

$\langle \text{branch-nominals } \text{branch} \equiv (\bigcup \text{block} \in \text{set } \text{branch}. \text{block-nominals } \text{block}) \rangle$

abbreviation $\text{at-in-branch} :: \langle 'a, 'b \text{ fm} \Rightarrow 'b \Rightarrow 'a, 'b \text{ branch} \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{at-in-branch } p \text{ a branch} \equiv \exists ps. (ps, a) \in. \text{branch} \wedge p \text{ on } (ps, a) \rangle$

notation *at-in-branch* ($\langle - \text{ at } - \text{ in } - \rangle$ [51, 51, 51] 50)

definition *new* :: $\langle ('a, 'b) \text{ fm} \Rightarrow 'b \Rightarrow ('a, 'b) \text{ branch} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{new } p \text{ a branch} \equiv \neg p \text{ at } a \text{ in branch} \rangle$

definition *witnessed* :: $\langle ('a, 'b) \text{ fm} \Rightarrow 'b \Rightarrow ('a, 'b) \text{ branch} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{witnessed } p \text{ a branch} \equiv \exists i. (\text{@ } i \text{ p}) \text{ at } a \text{ in branch} \wedge (\diamond \text{ Nom } i) \text{ at } a \text{ in branch} \rangle$

A branch has a closing tableau iff it is contained in the following inductively defined set. In that case I call the branch closeable. The first argument on the left of the turnstile, A , is a fixed set of nominals restricting Nom . This set rules out the copying of nominals and accessibility formulas introduced by DiaP . The second argument is "potential", used to restrict the GoTo rule.

inductive *STA* :: $\langle 'b \text{ set} \Rightarrow \text{nat} \Rightarrow ('a, 'b) \text{ branch} \Rightarrow \text{bool} \rangle$ ($\langle -, - \vdash - \rangle$ [50, 50, 50] 50)

for A :: $\langle 'b \text{ set} \rangle$ **where**

Close:

$\langle p \text{ at } i \text{ in branch} \Longrightarrow (\neg p) \text{ at } i \text{ in branch} \Longrightarrow$
 $A, n \vdash \text{branch} \rangle$

| *Neg:*

$\langle (\neg \neg p) \text{ at } a \text{ in } (ps, a) \# \text{branch} \Longrightarrow$
 $\text{new } p \text{ a } ((ps, a) \# \text{branch}) \Longrightarrow$
 $A, \text{Suc } n \vdash (p \# ps, a) \# \text{branch} \Longrightarrow$
 $A, n \vdash (ps, a) \# \text{branch} \rangle$

| *DisP:*

$\langle (p \vee q) \text{ at } a \text{ in } (ps, a) \# \text{branch} \Longrightarrow$
 $\text{new } p \text{ a } ((ps, a) \# \text{branch}) \Longrightarrow \text{new } q \text{ a } ((ps, a) \# \text{branch}) \Longrightarrow$
 $A, \text{Suc } n \vdash (p \# ps, a) \# \text{branch} \Longrightarrow A, \text{Suc } n \vdash (q \# ps, a) \# \text{branch} \Longrightarrow$
 $A, n \vdash (ps, a) \# \text{branch} \rangle$

| *DisN:*

$\langle (\neg (p \vee q)) \text{ at } a \text{ in } (ps, a) \# \text{branch} \Longrightarrow$
 $\text{new } (\neg p) \text{ a } ((ps, a) \# \text{branch}) \vee \text{new } (\neg q) \text{ a } ((ps, a) \# \text{branch}) \Longrightarrow$
 $A, \text{Suc } n \vdash ((\neg q) \# (\neg p) \# ps, a) \# \text{branch} \Longrightarrow$
 $A, n \vdash (ps, a) \# \text{branch} \rangle$

| *DiaP:*

$\langle (\diamond p) \text{ at } a \text{ in } (ps, a) \# \text{branch} \Longrightarrow$
 $i \notin A \cup \text{branch-nominals } ((ps, a) \# \text{branch}) \Longrightarrow$
 $\nexists a. p = \text{Nom } a \Longrightarrow \neg \text{witnessed } p \text{ a } ((ps, a) \# \text{branch}) \Longrightarrow$
 $A, \text{Suc } n \vdash ((\text{@ } i \text{ p}) \# (\diamond \text{ Nom } i) \# ps, a) \# \text{branch} \Longrightarrow$
 $A, n \vdash (ps, a) \# \text{branch} \rangle$

| *DiaN:*

$\langle (\neg (\diamond p)) \text{ at } a \text{ in } (ps, a) \# \text{branch} \Longrightarrow$
 $(\diamond \text{ Nom } i) \text{ at } a \text{ in } (ps, a) \# \text{branch} \Longrightarrow$
 $\text{new } (\neg (\text{@ } i \text{ p})) \text{ a } ((ps, a) \# \text{branch}) \Longrightarrow$
 $A, \text{Suc } n \vdash ((\neg (\text{@ } i \text{ p})) \# ps, a) \# \text{branch} \Longrightarrow$
 $A, n \vdash (ps, a) \# \text{branch} \rangle$

| *SatP:*

$\langle (\text{@ } a \text{ p}) \text{ at } b \text{ in } (ps, a) \# \text{branch} \Longrightarrow$
 $\text{new } p \text{ a } ((ps, a) \# \text{branch}) \Longrightarrow$

$A, \text{Suc } n \vdash (p \# ps, a) \# \text{branch} \implies$
 $A, n \vdash (ps, a) \# \text{branch}$
| *SatN*:
 $\langle \neg (@ a p) \text{ at } b \text{ in } (ps, a) \# \text{branch} \implies$
 $\text{new } (\neg p) a ((ps, a) \# \text{branch}) \implies$
 $A, \text{Suc } n \vdash ((\neg p) \# ps, a) \# \text{branch} \implies$
 $A, n \vdash (ps, a) \# \text{branch}$
| *GoTo*:
 $\langle i \in \text{branch-nominals } \text{branch} \implies$
 $A, n \vdash ([], i) \# \text{branch} \implies$
 $A, \text{Suc } n \vdash \text{branch}$
| *Nom*:
 $\langle p \text{ at } b \text{ in } (ps, a) \# \text{branch} \implies \text{Nom } a \text{ at } b \text{ in } (ps, a) \# \text{branch} \implies$
 $\forall i. p = \text{Nom } i \vee p = (\Diamond \text{Nom } i) \longrightarrow i \in A \implies$
 $\text{new } p a ((ps, a) \# \text{branch}) \implies$
 $A, \text{Suc } n \vdash (p \# ps, a) \# \text{branch} \implies$
 $A, n \vdash (ps, a) \# \text{branch}$

abbreviation *STA-ex-potential* :: $\langle 'b \text{ set} \Rightarrow ('a, 'b) \text{ branch} \Rightarrow \text{bool} \rangle$ ($\langle - \vdash - \rangle$ [50, 50] 50) **where**
 $\langle A \vdash \text{branch} \equiv \exists n. A, n \vdash \text{branch} \rangle$

lemma *STA-Suc*: $\langle A, n \vdash \text{branch} \implies A, \text{Suc } n \vdash \text{branch} \rangle$
 $\langle \text{proof} \rangle$

A verified derivation in the calculus.

lemma
fixes i
defines $\langle p \equiv \neg (@ i (\text{Nom } i)) \rangle$
shows $\langle A, \text{Suc } n \vdash [[p], a] \rangle$
 $\langle \text{proof} \rangle$

4 Soundness

An i -block is satisfied by a model M and assignment g if all formulas on the block are true under M at the world $g i$. A branch is satisfied by a model and assignment if all blocks on it are.

primrec *block-sat* :: $\langle ('w, 'a) \text{ model} \Rightarrow ('b \Rightarrow 'w) \Rightarrow ('a, 'b) \text{ block} \Rightarrow \text{bool} \rangle$
 $\langle \langle -, - \models_B - \rangle$ [50, 50] 50) **where**
 $\langle (M, g \models_B (ps, i)) = (\forall p \text{ on } (ps, i). M, g, g i \models p) \rangle$

abbreviation *branch-sat* ::
 $\langle ('w, 'a) \text{ model} \Rightarrow ('b \Rightarrow 'w) \Rightarrow ('a, 'b) \text{ branch} \Rightarrow \text{bool} \rangle$
 $\langle \langle -, - \models_\Theta - \rangle$ [50, 50] 50) **where**
 $\langle M, g \models_\Theta \text{branch} \equiv \forall (ps, i) \in \text{set } \text{branch}. M, g \models_B (ps, i) \rangle$

lemma *block-nominals*:
 $\langle p \text{ on } \text{block} \implies i \in \text{nominals } p \implies i \in \text{block-nominals } \text{block} \rangle$

$\langle \text{proof} \rangle$

lemma *block-sat-fresh*:

assumes $\langle M, g \models_B \text{block} \rangle \langle i \notin \text{block-nominals block} \rangle$

shows $\langle M, g(i := v) \models_B \text{block} \rangle$

$\langle \text{proof} \rangle$

lemma *branch-sat-fresh*:

assumes $\langle M, g \models_{\Theta} \text{branch} \rangle \langle i \notin \text{branch-nominals branch} \rangle$

shows $\langle M, g(i := v) \models_{\Theta} \text{branch} \rangle$

$\langle \text{proof} \rangle$

If a branch has a derivation then it cannot be satisfied.

lemma *soundness'*: $\langle A, n \vdash \text{branch} \implies M, g \models_{\Theta} \text{branch} \implies \text{False} \rangle$

$\langle \text{proof} \rangle$

lemma *block-sat*: $\langle \forall p \text{ on block. } M, g, w \models p \implies M, g \models_B \text{block} \rangle$

$\langle \text{proof} \rangle$

lemma *branch-sat*:

assumes $\langle \forall (ps, i) \in \text{set branch. } \forall p \text{ on } (ps, i). M, g, w \models p \rangle$

shows $\langle M, g \models_{\Theta} \text{branch} \rangle$

$\langle \text{proof} \rangle$

lemma *soundness*:

assumes $\langle A, n \vdash \text{branch} \rangle$

shows $\langle \exists \text{block} \in \text{set branch. } \exists p \text{ on block. } \neg M, g, w \models p \rangle$

$\langle \text{proof} \rangle$

corollary $\langle \neg A, n \vdash [] \rangle$

$\langle \text{proof} \rangle$

theorem *soundness-fresh*:

assumes $\langle A, n \vdash [(\neg p), i] \rangle \langle i \notin \text{nominals } p \rangle$

shows $\langle M, g, w \models p \rangle$

$\langle \text{proof} \rangle$

5 No Detours

We only need to spend initial potential when we apply GoTo twice in a row. Otherwise another rule will have been applied in-between that justifies the GoTo. Therefore, by filtering out detours we can close any closeable branch starting from a single unit of potential.

primrec *nonempty* :: $\langle ('a, 'b) \text{block} \Rightarrow \text{bool} \rangle$ **where**

$\langle \text{nonempty } (ps, i) = (ps \neq []) \rangle$

lemma *nonempty-Suc*:

assumes

$\langle A, n \vdash (ps, a) \# \text{filter nonempty left @ right} \rangle$

$\langle q \text{ at } a \text{ in } (ps, a) \# \text{filter nonempty left @ right} \rangle \langle q \neq \text{Nom } a \rangle$

shows $\langle A, \text{Suc } n \vdash \text{filter nonempty } ((ps, a) \# \text{left}) @ \text{right} \rangle$

$\langle \text{proof} \rangle$

lemma *STA-nonempty*:

$\langle A, n \vdash \text{left @ right} \implies A, \text{Suc } m \vdash \text{filter nonempty left @ right} \rangle$

$\langle \text{proof} \rangle$

theorem *STA-potential*: $\langle A, n \vdash \text{branch} \implies A, \text{Suc } m \vdash \text{branch} \rangle$

$\langle \text{proof} \rangle$

corollary *STA-one*: $\langle A, n \vdash \text{branch} \implies A, 1 \vdash \text{branch} \rangle$

$\langle \text{proof} \rangle$

5.1 Free GoTo

The above result allows us to prove a version of GoTo that works "for free."

lemma *GoTo'*:

assumes $\langle A, \text{Suc } n \vdash ([], i) \# \text{branch} \rangle \langle i \in \text{branch-nominals branch} \rangle$

shows $\langle A, \text{Suc } n \vdash \text{branch} \rangle$

$\langle \text{proof} \rangle$

6 Indexed Mapping

This section contains some machinery for showing admissible rules.

6.1 Indexing

We use pairs of natural numbers to index into the branch. The first component specifies the block and the second specifies the formula on that block. We index from the back to ensure that indices are stable under the addition of new formulas and blocks.

primrec *rev-nth* :: $\langle 'a \text{ list} \Rightarrow \text{nat} \Rightarrow 'a \text{ option} \rangle$ (**infixl** $\langle !. \rangle$ 100) **where**

$\langle [] !. v = \text{None} \rangle$

$| \langle (x \# xs) !. v = (\text{if length } xs = v \text{ then } \text{Some } x \text{ else } xs !. v) \rangle$

lemma *rev-nth-last*: $\langle xs !. 0 = \text{Some } x \implies \text{last } xs = x \rangle$

$\langle \text{proof} \rangle$

lemma *rev-nth-zero*: $\langle (xs @ [x]) !. 0 = \text{Some } x \rangle$

$\langle \text{proof} \rangle$

lemma *rev-nth-snoc*: $\langle (xs @ [x]) !. \text{Suc } v = \text{Some } y \implies xs !. v = \text{Some } y \rangle$

$\langle \text{proof} \rangle$

lemma *rev-nth-Suc*: $\langle (xs @ [x]) !. Suc\ v = xs !. v \rangle$
 $\langle proof \rangle$

lemma *rev-nth-bounded*: $\langle v < length\ xs \implies \exists x. xs !. v = Some\ x \rangle$
 $\langle proof \rangle$

lemma *rev-nth-Cons*: $\langle xs !. v = Some\ y \implies (x \# xs) !. v = Some\ y \rangle$
 $\langle proof \rangle$

lemma *rev-nth-append*: $\langle xs !. v = Some\ y \implies (ys @ xs) !. v = Some\ y \rangle$
 $\langle proof \rangle$

lemma *rev-nth-mem*: $\langle block \in. branch \longleftrightarrow (\exists v. branch !. v = Some\ block) \rangle$
 $\langle proof \rangle$

lemma *rev-nth-on*: $\langle p\ on\ (ps, i) \longleftrightarrow (\exists v. ps !. v = Some\ p) \vee p = Nom\ i \rangle$
 $\langle proof \rangle$

lemma *rev-nth-Some*: $\langle xs !. v = Some\ y \implies v < length\ xs \rangle$
 $\langle proof \rangle$

lemma *index-Cons*:

assumes $\langle ((ps, a) \# branch) !. v = Some\ (qs, b) \rangle \langle qs !. v' = Some\ q \rangle$
shows $\langle \exists qs'. ((p \# ps, a) \# branch) !. v = Some\ (qs', b) \wedge qs' !. v' = Some\ q \rangle$
 $\langle proof \rangle$

6.2 Mapping

primrec *mapi* :: $\langle (nat \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a\ list \Rightarrow 'b\ list \rangle$ **where**
 $\langle mapi\ f\ [] = [] \rangle$
 $| \langle mapi\ f\ (x \# xs) = f\ (length\ xs)\ x \# mapi\ f\ xs \rangle$

primrec *mapi-block* ::
 $\langle (nat \Rightarrow ('a, 'b)\ fm \Rightarrow ('a, 'b)\ fm) \Rightarrow (('a, 'b)\ block \Rightarrow ('a, 'b)\ block) \rangle$ **where**
 $\langle mapi-block\ f\ (ps, i) = (mapi\ f\ ps, i) \rangle$

definition *mapi-branch* ::
 $\langle (nat \Rightarrow nat \Rightarrow ('a, 'b)\ fm \Rightarrow ('a, 'b)\ fm) \Rightarrow (('a, 'b)\ branch \Rightarrow ('a, 'b)\ branch) \rangle$
where
 $\langle mapi-branch\ f\ branch \equiv mapi\ (\lambda v. mapi-block\ (f\ v))\ branch \rangle$

abbreviation *mapper* ::
 $\langle (('a, 'b)\ fm \Rightarrow ('a, 'b)\ fm) \Rightarrow$
 $(nat \times nat)\ set \Rightarrow nat \Rightarrow nat \Rightarrow ('a, 'b)\ fm \Rightarrow ('a, 'b)\ fm \rangle$ **where**
 $\langle mapper\ f\ xs\ v\ v'\ p \equiv (if\ (v, v') \in xs\ then\ f\ p\ else\ p) \rangle$

lemma *mapi-block-add-oob*:
assumes $\langle length\ ps \leq v' \rangle$

shows

$\langle \text{mapi-block } (\text{mapper } f \ (\{(v, v')\} \cup xs) \ v) \ (ps, i) =$
 $\text{mapi-block } (\text{mapper } f \ xs \ v) \ (ps, i) \rangle$
 $\langle \text{proof} \rangle$

lemma *mapi-branch-add-oob*:

assumes $\langle \text{length } \text{branch} \leq v \rangle$

shows

$\langle \text{mapi-branch } (\text{mapper } f \ (\{(v, v')\} \cup xs)) \ \text{branch} =$
 $\text{mapi-branch } (\text{mapper } f \ xs) \ \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *mapi-branch-head-add-oob*:

$\langle \text{mapi-branch } (\text{mapper } f \ (\{(\text{length } \text{branch}, \text{length } ps)\} \cup xs)) \ ((ps, a) \# \text{branch})$
 $=$
 $\text{mapi-branch } (\text{mapper } f \ xs) \ ((ps, a) \# \text{branch}) \rangle$
 $\langle \text{proof} \rangle$

lemma *mapi-branch-mem*:

assumes $\langle (ps, i) \in. \text{branch} \rangle$

shows $\langle \exists v. (\text{mapi } (f \ v) \ ps, i) \in. \text{mapi-branch } f \ \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *rev-nth-mapi-branch*:

assumes $\langle \text{branch} \ !. \ v = \text{Some } (ps, a) \rangle$

shows $\langle (\text{mapi } (f \ v) \ ps, a) \in. \text{mapi-branch } f \ \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *rev-nth-mapi-block*:

assumes $\langle ps \ !. \ v' = \text{Some } p \rangle$

shows $\langle f \ v' \ p \ \text{on } (\text{mapi } f \ ps, a) \rangle$
 $\langle \text{proof} \rangle$

lemma *mapi-append*:

$\langle \text{mapi } f \ (xs \ @ \ ys) = \text{mapi } (\lambda v. f \ (v + \text{length } ys)) \ xs \ @ \ \text{mapi } f \ ys \rangle$
 $\langle \text{proof} \rangle$

lemma *mapi-block-id*: $\langle \text{mapi-block } (\text{mapper } f \ \{\} \ v) \ (ps, i) = (ps, i) \rangle$

$\langle \text{proof} \rangle$

lemma *mapi-branch-id*: $\langle \text{mapi-branch } (\text{mapper } f \ \{\}) \ \text{branch} = \text{branch} \rangle$

$\langle \text{proof} \rangle$

lemma *length-mapi*: $\langle \text{length } (\text{mapi } f \ xs) = \text{length } xs \rangle$

$\langle \text{proof} \rangle$

lemma *mapi-rev-nth*:

assumes $\langle xs \ !. \ v = \text{Some } x \rangle$

shows $\langle \text{mapi } f \ xs \ !. \ v = \text{Some } (f \ v \ x) \rangle$

⟨proof⟩

7 Duplicate Formulas

7.1 Removable indices

abbreviation $\langle \text{proj} \equiv \text{Equiv-Relations.proj} \rangle$

definition $\text{all-is} :: \langle ('a, 'b) \text{ fm} \Rightarrow ('a, 'b) \text{ fm list} \Rightarrow \text{nat set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{all-is } p \text{ ps } xs \equiv \forall v \in xs. \text{ps} !. v = \text{Some } p \rangle$

definition $\text{is-at} :: \langle ('a, 'b) \text{ fm} \Rightarrow 'b \Rightarrow ('a, 'b) \text{ branch} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{is-at } p \text{ i branch } v \text{ v}' \equiv \exists \text{ps. branch} !. v = \text{Some } (\text{ps}, i) \wedge \text{ps} !. v' = \text{Some } p \rangle$

This definition is slightly complicated by the inability to index the opening nominal.

definition $\text{is-elsewhere} :: \langle ('a, 'b) \text{ fm} \Rightarrow 'b \Rightarrow ('a, 'b) \text{ branch} \Rightarrow (\text{nat} \times \text{nat}) \text{ set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{is-elsewhere } p \text{ i branch } xs \equiv \exists w \text{ w}' \text{ ps. } (w, w') \notin xs \wedge \text{branch} !. w = \text{Some } (\text{ps}, i) \wedge (p = \text{Nom } i \vee \text{ps} !. w' = \text{Some } p) \rangle$

definition $\text{Dup} :: \langle ('a, 'b) \text{ fm} \Rightarrow 'b \Rightarrow ('a, 'b) \text{ branch} \Rightarrow (\text{nat} \times \text{nat}) \text{ set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{Dup } p \text{ i branch } xs \equiv \forall (v, v') \in xs. \text{is-at } p \text{ i branch } v \text{ v}' \wedge \text{is-elsewhere } p \text{ i branch } xs \rangle$

lemma Dup-all-is :

assumes $\langle \text{Dup } p \text{ i branch } xs \rangle \langle \text{branch} !. v = \text{Some } (\text{ps}, a) \rangle$

shows $\langle \text{all-is } p \text{ ps } (\text{proj } xs \text{ } v) \rangle$

⟨proof⟩

lemma Dup-branch :

$\langle \text{Dup } p \text{ i branch } xs \implies \text{Dup } p \text{ i } (\text{extra } @ \text{ branch}) \text{ } xs \rangle$

⟨proof⟩

lemma Dup-block :

assumes $\langle \text{Dup } p \text{ i } ((\text{ps}, a) \# \text{ branch}) \text{ } xs \rangle$

shows $\langle \text{Dup } p \text{ i } ((\text{ps}' @ \text{ps}, a) \# \text{ branch}) \text{ } xs \rangle$

⟨proof⟩

definition $\text{only-touches} :: \langle 'b \Rightarrow ('a, 'b) \text{ branch} \Rightarrow (\text{nat} \times \text{nat}) \text{ set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{only-touches } i \text{ branch } xs \equiv \forall (v, v') \in xs. \forall \text{ps } a. \text{branch} !. v = \text{Some } (\text{ps}, a) \longrightarrow i = a \rangle$

lemma Dup-touches : $\langle \text{Dup } p \text{ i branch } xs \implies \text{only-touches } i \text{ branch } xs \rangle$
⟨proof⟩

lemma $\text{only-touches-opening}$:

assumes $\langle \text{only-touches } i \text{ branch } xs \rangle \langle (v, v') \in xs \rangle \langle \text{branch} !. v = \text{Some } (\text{ps}, a) \rangle$

shows $\langle i = a \rangle$
 $\langle \text{proof} \rangle$

lemma *Dup-head*:

$\langle \text{Dup } p \ i \ ((ps, a) \# \text{branch}) \ xs \implies \text{Dup } p \ i \ ((q \# ps, a) \# \text{branch}) \ xs \rangle$
 $\langle \text{proof} \rangle$

lemma *Dup-head-oob'*:

assumes $\langle \text{Dup } p \ i \ ((ps, a) \# \text{branch}) \ xs \rangle$
shows $\langle (\text{length } \text{branch}, k + \text{length } ps) \notin xs \rangle$
 $\langle \text{proof} \rangle$

lemma *Dup-head-oob*:

assumes $\langle \text{Dup } p \ i \ ((ps, a) \# \text{branch}) \ xs \rangle$
shows $\langle (\text{length } \text{branch}, \text{length } ps) \notin xs \rangle$
 $\langle \text{proof} \rangle$

7.2 Omitting formulas

primrec *omit* :: $\langle \text{nat set} \Rightarrow ('a, 'b) \text{ fm list} \Rightarrow ('a, 'b) \text{ fm list} \rangle$ **where**

$\langle \text{omit } xs \ [] = [] \rangle$
 $| \langle \text{omit } xs \ (p \# ps) = (\text{if } \text{length } ps \in xs \text{ then } \text{omit } xs \ ps \text{ else } p \# \text{omit } xs \ ps) \rangle$

primrec *omit-block* :: $\langle \text{nat set} \Rightarrow ('a, 'b) \text{ block} \Rightarrow ('a, 'b) \text{ block} \rangle$ **where**

$\langle \text{omit-block } xs \ (ps, a) = (\text{omit } xs \ ps, a) \rangle$

definition *omit-branch* :: $\langle (\text{nat} \times \text{nat}) \text{ set} \Rightarrow ('a, 'b) \text{ branch} \Rightarrow ('a, 'b) \text{ branch} \rangle$

where

$\langle \text{omit-branch } xs \ \text{branch} \equiv \text{mapi } (\lambda v. \text{omit-block } (\text{proj } xs \ v)) \ \text{branch} \rangle$

lemma *omit-mem*: $\langle ps \ !. \ v = \text{Some } p \implies v \notin xs \implies p \in. \text{omit } xs \ ps \rangle$

$\langle \text{proof} \rangle$

lemma *omit-id*: $\langle \text{omit } \{ \} \ ps = ps \rangle$

$\langle \text{proof} \rangle$

lemma *omit-block-id*: $\langle \text{omit-block } \{ \} \ \text{block} = \text{block} \rangle$

$\langle \text{proof} \rangle$

lemma *omit-branch-id*: $\langle \text{omit-branch } \{ \} \ \text{branch} = \text{branch} \rangle$

$\langle \text{proof} \rangle$

lemma *omit-branch-mem-diff-opening*:

assumes $\langle \text{only-touches } i \ \text{branch } xs \rangle \langle (ps, a) \in. \text{branch} \rangle \langle i \neq a \rangle$

shows $\langle (ps, a) \in. \text{omit-branch } xs \ \text{branch} \rangle$

$\langle \text{proof} \rangle$

lemma *Dup-omit-branch-mem-same-opening*:

assumes $\langle \text{Dup } p \ i \ \text{branch } xs \rangle \langle p \ \text{at } i \ \text{in } \text{branch} \rangle$

shows $\langle p \text{ at } i \text{ in omit-branch } xs \text{ branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *omit-del*:

assumes $\langle p \in. ps \rangle \langle p \notin \text{set } (\text{omit } xs \ ps) \rangle$
shows $\langle \exists v. ps \ !. v = \text{Some } p \wedge v \in xs \rangle$
 $\langle \text{proof} \rangle$

lemma *omit-all-is*:

assumes $\langle \text{all-is } p \ ps \ xs \rangle \langle q \in. ps \rangle \langle q \notin \text{set } (\text{omit } xs \ ps) \rangle$
shows $\langle q = p \rangle$
 $\langle \text{proof} \rangle$

definition *all-is-branch* :: $\langle ('a, 'b) \text{ fm} \Rightarrow 'b \Rightarrow ('a, 'b) \text{ branch} \Rightarrow (\text{nat} \times \text{nat}) \text{ set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{all-is-branch } p \ i \ \text{branch } xs \equiv \forall (v, v') \in xs. v < \text{length } \text{branch} \longrightarrow \text{is-at } p \ i \ \text{branch } v \ v' \rangle$

lemma *all-is-branch*:

$\langle \text{all-is-branch } p \ i \ \text{branch } xs \Longrightarrow \text{branch} \ !. v = \text{Some } (ps, a) \Longrightarrow \text{all-is } p \ ps \ (\text{proj } xs \ v) \rangle$
 $\langle \text{proof} \rangle$

lemma *Dup-all-is-branch*: $\langle \text{Dup } p \ i \ \text{branch } xs \Longrightarrow \text{all-is-branch } p \ i \ \text{branch } xs \rangle$

$\langle \text{proof} \rangle$

lemma *omit-branch-mem-diff-formula*:

assumes $\langle \text{all-is-branch } p \ i \ \text{branch } xs \rangle \langle q \text{ at } i \ \text{in } \text{branch} \rangle \langle p \neq q \rangle$
shows $\langle q \text{ at } i \ \text{in } \text{omit-branch } xs \ \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *Dup-omit-branch-mem*:

assumes $\langle \text{Dup } p \ i \ \text{branch } xs \rangle \langle q \text{ at } a \ \text{in } \text{branch} \rangle$
shows $\langle q \text{ at } a \ \text{in } \text{omit-branch } xs \ \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *omit-set*: $\langle \text{set } (\text{omit } xs \ ps) \subseteq \text{set } ps \rangle$

$\langle \text{proof} \rangle$

lemma *on-omit*: $\langle p \ \text{on } (\text{omit } xs \ ps, i) \Longrightarrow p \ \text{on } (ps, i) \rangle$

$\langle \text{proof} \rangle$

lemma *all-is-set*:

assumes $\langle \text{all-is } p \ ps \ xs \rangle$
shows $\langle \{p\} \cup \text{set } (\text{omit } xs \ ps) = \{p\} \cup \text{set } ps \rangle$
 $\langle \text{proof} \rangle$

lemma *all-is-list-nominals*:

assumes $\langle \text{all-is } p \ ps \ xs \rangle$

shows $\langle \text{nominals } p \cup \text{list-nominals (omit } xs \text{ } ps) = \text{nominals } p \cup \text{list-nominals } ps \rangle$
 $\langle \text{proof} \rangle$

lemma *all-is-block-nominals*:

assumes $\langle \text{all-is } p \text{ } ps \text{ } xs \rangle$
shows $\langle \text{nominals } p \cup \text{block-nominals (omit } xs \text{ } ps, i) = \text{nominals } p \cup \text{block-nominals } (ps, i) \rangle$
 $\langle \text{proof} \rangle$

lemma *all-is-branch-nominals'*:

assumes $\langle \text{all-is-branch } p \text{ } i \text{ } \text{branch } xs \rangle$
shows
 $\langle \text{nominals } p \cup \text{branch-nominals (omit-branch } xs \text{ } \text{branch}) = \text{nominals } p \cup \text{branch-nominals } \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *Dup-branch-nominals*:

assumes $\langle \text{Dup } p \text{ } i \text{ } \text{branch } xs \rangle$
shows $\langle \text{branch-nominals (omit-branch } xs \text{ } \text{branch}) = \text{branch-nominals } \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *omit-branch-mem-dual*:

assumes $\langle p \text{ at } i \text{ in omit-branch } xs \text{ } \text{branch} \rangle$
shows $\langle p \text{ at } i \text{ in } \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *witnessed-omit-branch*:

assumes $\langle \text{witnessed } p \text{ } a \text{ (omit-branch } xs \text{ } \text{branch}) \rangle$
shows $\langle \text{witnessed } p \text{ } a \text{ } \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *new-omit-branch*:

assumes $\langle \text{new } p \text{ } a \text{ } \text{branch} \rangle$
shows $\langle \text{new } p \text{ } a \text{ (omit-branch } xs \text{ } \text{branch}) \rangle$
 $\langle \text{proof} \rangle$

lemma *omit-oob*:

assumes $\langle \text{length } ps \leq v \rangle$
shows $\langle \text{omit } (\{v\} \cup xs) \text{ } ps = \text{omit } xs \text{ } ps \rangle$
 $\langle \text{proof} \rangle$

lemma *omit-branch-oob*:

assumes $\langle \text{length } \text{branch} \leq v \rangle$
shows $\langle \text{omit-branch } (\{(v, v')\} \cup xs) \text{ } \text{branch} = \text{omit-branch } xs \text{ } \text{branch} \rangle$
 $\langle \text{proof} \rangle$

7.3 Induction

lemma *STA-Dup*:

assumes $\langle A, n \vdash \text{branch} \rangle \langle \text{Dup } q \text{ i branch } xs \rangle$
shows $\langle A, n \vdash \text{omit-branch } xs \text{ branch} \rangle$
\langle proof \rangle

theorem *Dup*:

assumes $\langle A, n \vdash (p \# ps, a) \# \text{branch} \rangle \langle \neg \text{new } p \text{ a } ((ps, a) \# \text{branch}) \rangle$
shows $\langle A, n \vdash (ps, a) \# \text{branch} \rangle$
\langle proof \rangle

7.4 Unrestricted rules

lemma *STA-add*: $\langle A, n \vdash \text{branch} \implies A, m + n \vdash \text{branch} \rangle$
\langle proof \rangle

lemma *STA-le*: $\langle A, n \vdash \text{branch} \implies n \leq m \implies A, m \vdash \text{branch} \rangle$
\langle proof \rangle

lemma *Neg'*:

assumes
 $\langle (\neg \neg p) \text{ at } a \text{ in } (ps, a) \# \text{branch} \rangle$
 $\langle A, n \vdash (p \# ps, a) \# \text{branch} \rangle$
shows $\langle A, n \vdash (ps, a) \# \text{branch} \rangle$
\langle proof \rangle

lemma *DisP'*:

assumes
 $\langle (p \vee q) \text{ at } a \text{ in } (ps, a) \# \text{branch} \rangle$
 $\langle A, n \vdash (p \# ps, a) \# \text{branch} \rangle \langle A, n \vdash (q \# ps, a) \# \text{branch} \rangle$
shows $\langle A, n \vdash (ps, a) \# \text{branch} \rangle$
\langle proof \rangle

lemma *DisP''*:

assumes
 $\langle (p \vee q) \text{ at } a \text{ in } (ps, a) \# \text{branch} \rangle$
 $\langle A, n \vdash (p \# ps, a) \# \text{branch} \rangle \langle A, m \vdash (q \# ps, a) \# \text{branch} \rangle$
shows $\langle A, \max n m \vdash (ps, a) \# \text{branch} \rangle$
\langle proof \rangle

lemma *DisN'*:

assumes
 $\langle (\neg (p \vee q)) \text{ at } a \text{ in } (ps, a) \# \text{branch} \rangle$
 $\langle A, n \vdash ((\neg q) \# (\neg p) \# ps, a) \# \text{branch} \rangle$
shows $\langle A, n \vdash (ps, a) \# \text{branch} \rangle$
\langle proof \rangle

lemma *DiaP'*:

assumes

$\langle (\diamond p) \text{ at } a \text{ in } (ps, a) \# \text{ branch} \rangle$
 $\langle i \notin A \cup \text{branch-nominals } ((ps, a) \# \text{ branch}) \rangle$
 $\langle \nexists a. p = \text{Nom } a \rangle$
 $\langle \neg \text{witnessed } p \ a \ ((ps, a) \# \text{ branch}) \rangle$
 $\langle A, n \vdash ((@ i p) \# (\diamond \text{Nom } i) \# ps, a) \# \text{ branch} \rangle$
shows $\langle A, n \vdash (ps, a) \# \text{ branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *DiaN'*:

assumes
 $\langle (\neg (\diamond p)) \text{ at } a \text{ in } (ps, a) \# \text{ branch} \rangle$
 $\langle (\diamond \text{Nom } i) \text{ at } a \text{ in } (ps, a) \# \text{ branch} \rangle$
 $\langle A, n \vdash ((\neg (@ i p)) \# ps, a) \# \text{ branch} \rangle$
shows $\langle A, n \vdash (ps, a) \# \text{ branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *SatP'*:

assumes
 $\langle (@ a p) \text{ at } b \text{ in } (ps, a) \# \text{ branch} \rangle$
 $\langle A, n \vdash (p \# ps, a) \# \text{ branch} \rangle$
shows $\langle A, n \vdash (ps, a) \# \text{ branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *SatN'*:

assumes
 $\langle (\neg (@ a p)) \text{ at } b \text{ in } (ps, a) \# \text{ branch} \rangle$
 $\langle A, n \vdash ((\neg p) \# ps, a) \# \text{ branch} \rangle$
shows $\langle A, n \vdash (ps, a) \# \text{ branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *Nom'*:

assumes
 $\langle p \text{ at } b \text{ in } (ps, a) \# \text{ branch} \rangle$
 $\langle \text{Nom } a \text{ at } b \text{ in } (ps, a) \# \text{ branch} \rangle$
 $\langle \forall i. p = \text{Nom } i \vee p = (\diamond \text{Nom } i) \longrightarrow i \in A \rangle$
 $\langle A, n \vdash (p \# ps, a) \# \text{ branch} \rangle$
shows $\langle A, n \vdash (ps, a) \# \text{ branch} \rangle$
 $\langle \text{proof} \rangle$

8 Substitution

lemma *finite-nominals*: $\langle \text{finite } (\text{nominals } p) \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-block-nominals*: $\langle \text{finite } (\text{block-nominals } \text{block}) \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-branch-nominals*: $\langle \text{finite } (\text{branch-nominals } \text{branch}) \rangle$
 $\langle \text{proof} \rangle$

abbreviation $sub\text{-}list :: \langle ('b \Rightarrow 'c) \Rightarrow ('a, 'b) \text{ fm list} \Rightarrow ('a, 'c) \text{ fm list} \rangle$ **where**
 $\langle sub\text{-}list f ps \equiv map (sub f) ps \rangle$

primrec $sub\text{-}block :: \langle ('b \Rightarrow 'c) \Rightarrow ('a, 'b) \text{ block} \Rightarrow ('a, 'c) \text{ block} \rangle$ **where**
 $\langle sub\text{-}block f (ps, i) = (sub\text{-}list f ps, f i) \rangle$

definition $sub\text{-}branch :: \langle ('b \Rightarrow 'c) \Rightarrow ('a, 'b) \text{ branch} \Rightarrow ('a, 'c) \text{ branch} \rangle$ **where**
 $\langle sub\text{-}branch f blocks \equiv map (sub\text{-}block f) blocks \rangle$

lemma $sub\text{-}block\text{-}mem$: $\langle p \text{ on } block \implies sub f p \text{ on } sub\text{-}block f block \rangle$
 $\langle proof \rangle$

lemma $sub\text{-}branch\text{-}mem$:
assumes $\langle (ps, i) \in . branch \rangle$
shows $\langle (sub\text{-}list f ps, f i) \in . sub\text{-}branch f branch \rangle$
 $\langle proof \rangle$

lemma $sub\text{-}block\text{-}nominals$: $\langle block\text{-}nominals (sub\text{-}block f block) = f ` block\text{-}nominals block \rangle$
 $\langle proof \rangle$

lemma $sub\text{-}branch\text{-}nominals$:
 $\langle branch\text{-}nominals (sub\text{-}branch f branch) = f ` branch\text{-}nominals branch \rangle$
 $\langle proof \rangle$

lemma $sub\text{-}list\text{-}id$: $\langle sub\text{-}list id ps = ps \rangle$
 $\langle proof \rangle$

lemma $sub\text{-}block\text{-}id$: $\langle sub\text{-}block id block = block \rangle$
 $\langle proof \rangle$

lemma $sub\text{-}branch\text{-}id$: $\langle sub\text{-}branch id branch = branch \rangle$
 $\langle proof \rangle$

lemma $sub\text{-}block\text{-}upd\text{-}fresh$:
assumes $\langle i \notin block\text{-}nominals block \rangle$
shows $\langle sub\text{-}block (f(i := j)) block = sub\text{-}block f block \rangle$
 $\langle proof \rangle$

lemma $sub\text{-}branch\text{-}upd\text{-}fresh$:
assumes $\langle i \notin branch\text{-}nominals branch \rangle$
shows $\langle sub\text{-}branch (f(i := j)) branch = sub\text{-}branch f branch \rangle$
 $\langle proof \rangle$

lemma $sub\text{-}comp$: $\langle sub f (sub g p) = sub (f o g) p \rangle$
 $\langle proof \rangle$

lemma $sub\text{-}list\text{-}comp$: $\langle sub\text{-}list f (sub\text{-}list g ps) = sub\text{-}list (f o g) ps \rangle$

$\langle \text{proof} \rangle$

lemma *sub-block-comp*: $\langle \text{sub-block } f \text{ (sub-block } g \text{ block)} = \text{sub-block } (f \circ g) \text{ block} \rangle$
 $\langle \text{proof} \rangle$

lemma *sub-branch-comp*:
 $\langle \text{sub-branch } f \text{ (sub-branch } g \text{ branch)} = \text{sub-branch } (f \circ g) \text{ branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *swap-id*: $\langle (\text{id}(i := j, j := i)) \circ (\text{id}(i := j, j := i)) = \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *at-in-sub-branch*:
assumes $\langle p \text{ at } i \text{ in } (ps, a) \# \text{branch} \rangle$
shows $\langle \text{sub } f \text{ } p \text{ at } f \text{ } i \text{ in } (\text{sub-list } f \text{ } ps, f \text{ } a) \# \text{sub-branch } f \text{ branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *sub-still-allowed*:
assumes $\langle \forall i. p = \text{Nom } i \vee p = (\diamond \text{Nom } i) \longrightarrow i \in A \rangle$
shows $\langle \text{sub } f \text{ } p = \text{Nom } i \vee \text{sub } f \text{ } p = (\diamond \text{Nom } i) \longrightarrow i \in f \text{ } A \rangle$
 $\langle \text{proof} \rangle$

If a branch has a closing tableau then so does any branch obtained by renaming nominals as long as the substitution leaves some nominals free. This is always the case for substitutions that do not change the type of nominals. Since some formulas on the renamed branch may no longer be new, they do not contribute any potential and so we existentially quantify over the potential needed to close the new branch. We assume that the set of allowed nominals A is finite such that we can obtain a free nominal.

lemma *STA-sub'*:
fixes $f :: \langle 'b \Rightarrow 'c \rangle$
assumes $\langle \bigwedge (f :: 'b \Rightarrow 'c) \ i \ A. \text{finite } A \Longrightarrow i \notin A \Longrightarrow \exists j. j \notin f \text{ } A \rangle$
 $\langle \text{finite } A \rangle \langle A, n \vdash \text{branch} \rangle$
shows $\langle f \text{ } A \vdash \text{sub-branch } f \text{ branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *ex-fresh-gt*:
fixes $f :: \langle 'b \Rightarrow 'c \rangle$
assumes $\langle \exists g :: 'c \Rightarrow 'b. \text{surj } g \rangle \langle \text{finite } A \rangle \langle i \notin A \rangle$
shows $\langle \exists j. j \notin f \text{ } A \rangle$
 $\langle \text{proof} \rangle$

corollary *STA-sub-gt*:
fixes $f :: \langle 'b \Rightarrow 'c \rangle$
assumes $\langle \exists g :: 'c \Rightarrow 'b. \text{surj } g \rangle \langle A \vdash \text{branch} \rangle$
 $\langle \text{finite } A \rangle \langle \forall i \in \text{branch-nominals } \text{branch}. f \text{ } i \in f \text{ } A \longrightarrow i \in A \rangle$
shows $\langle f \text{ } A \vdash \text{sub-branch } f \text{ branch} \rangle$
 $\langle \text{proof} \rangle$

corollary *STA-sub-inf*:

fixes $f :: \langle 'b \Rightarrow 'c \rangle$
assumes $\langle \text{infinite } (UNIV :: 'c \text{ set}) \rangle \langle A \vdash \text{branch} \rangle$
 $\langle \text{finite } A \rangle \langle \forall i \in \text{branch-nominals } \text{branch}. f \ i \in f \ 'A \longrightarrow i \in A \rangle$
shows $\langle f \ 'A \vdash \text{sub-branch } f \ \text{branch} \rangle$
 $\langle \text{proof} \rangle$

corollary *STA-sub*:

fixes $f :: \langle 'b \Rightarrow 'b \rangle$
assumes $\langle A \vdash \text{branch} \rangle \langle \text{finite } A \rangle$
shows $\langle f \ 'A \vdash \text{sub-branch } f \ \text{branch} \rangle$
 $\langle \text{proof} \rangle$

8.1 Unrestricted (\diamond) rule

lemma *DiaP''*:

assumes
 $\langle (\diamond p) \text{ at } a \text{ in } (ps, a) \# \text{branch} \rangle$
 $\langle i \notin A \cup \text{branch-nominals } ((ps, a) \# \text{branch}) \rangle \langle \nexists a. p = \text{Nom } a \rangle$
 $\langle \text{finite } A \rangle$
 $\langle A \vdash ((@ i p) \# (\diamond \text{Nom } i) \# ps, a) \# \text{branch} \rangle$
shows $\langle A \vdash (ps, a) \# \text{branch} \rangle$
 $\langle \text{proof} \rangle$

9 Structural Properties

lemma *block-nominals-branch*:

assumes $\langle \text{block} \in. \text{branch} \rangle$
shows $\langle \text{block-nominals } \text{block} \subseteq \text{branch-nominals } \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *sub-block-fresh*:

assumes $\langle i \notin \text{branch-nominals } \text{branch} \rangle \langle \text{block} \in. \text{branch} \rangle$
shows $\langle \text{sub-block } (f(i := j)) \ \text{block} = \text{sub-block } f \ \text{block} \rangle$
 $\langle \text{proof} \rangle$

lemma *list-down-induct* [*consumes 1, case-names Start Cons*]:

assumes $\langle \forall y \in \text{set } ys. Q \ y \rangle \langle P \ (ys \ @ \ xs) \rangle$
 $\langle \bigwedge y \ xs. Q \ y \Longrightarrow P \ (y \ # \ xs) \Longrightarrow P \ xs \rangle$
shows $\langle P \ xs \rangle$
 $\langle \text{proof} \rangle$

If the last block on a branch has opening nominal a and the last formulas on that block occur on another block alongside nominal a , then we can drop those formulas.

lemma *STA-drop-prefix*:

assumes $\langle \text{set } ps \subseteq \text{set } qs \rangle \langle (qs, a) \in. \text{branch} \rangle \langle A, n \vdash (ps \ @ \ ps', a) \# \text{branch} \rangle$

shows $\langle A, n \vdash (ps', a) \# \text{branch} \rangle$
 $\langle \text{proof} \rangle$

We can drop a block if it is subsumed by another block.

lemma *STA-drop-block*:

assumes
 $\langle \text{set } ps \subseteq \text{set } ps' \rangle \langle (ps', a) \in. \text{branch} \rangle$
 $\langle A, n \vdash (ps, a) \# \text{branch} \rangle$
shows $\langle A, \text{Suc } n \vdash \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *STA-drop-block'*:

assumes $\langle A, n \vdash (ps, a) \# \text{branch} \rangle \langle (ps, a) \in. \text{branch} \rangle$
shows $\langle A, \text{Suc } n \vdash \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *sub-branch-image*: $\langle \text{set } (\text{sub-branch } f \text{ branch}) = \text{sub-block } f \text{ ' set branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *sub-block-repl*:

assumes $\langle j \notin \text{block-nominals } \text{block} \rangle$
shows $\langle i \notin \text{block-nominals } (\text{sub-block } (\text{id}(i := j, j := i)) \text{ block}) \rangle$
 $\langle \text{proof} \rangle$

lemma *sub-branch-repl*:

assumes $\langle j \notin \text{branch-nominals } \text{branch} \rangle$
shows $\langle i \notin \text{branch-nominals } (\text{sub-branch } (\text{id}(i := j, j := i)) \text{ branch}) \rangle$
 $\langle \text{proof} \rangle$

If a finite set of blocks has a closing tableau then so does any finite superset.

lemma *STA-struct*:

fixes $\text{branch} :: \langle ('a, 'b) \text{ branch} \rangle$
assumes
 $\text{inf}: \langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$ **and** $\text{fin}: \langle \text{finite } A \rangle$ **and**
 $\langle A, n \vdash \text{branch} \rangle \langle \text{set } \text{branch} \subseteq \text{set } \text{branch}' \rangle$
shows $\langle A \vdash \text{branch}' \rangle$
 $\langle \text{proof} \rangle$

If a branch has a closing tableau then we can replace the formulas of the last block on that branch with any finite superset and still obtain a closing tableau.

lemma *STA-struct-block*:

fixes $\text{branch} :: \langle ('a, 'b) \text{ branch} \rangle$
assumes
 $\text{inf}: \langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$ **and** $\text{fin}: \langle \text{finite } A \rangle$ **and**
 $\langle A, n \vdash (ps, a) \# \text{branch} \rangle \langle \text{set } ps \subseteq \text{set } ps' \rangle$
shows $\langle A \vdash (ps', a) \# \text{branch} \rangle$
 $\langle \text{proof} \rangle$

10 Bridge

We define a *descendants k i branch* relation on sets of indices. The sets are built on the index of a $\diamond \text{Nom } k$ on an *i*-block in *branch* and can be extended by indices of formula occurrences that can be thought of as descending from that $\diamond \text{Nom } k$ by application of either the $(\neg \diamond)$ or *Nom* rule.

We show that if we have nominals *j* and *k* on the same block in a closeable branch, then the branch obtained by the following transformation is also closeable: For every index *v*, if the formula at *v* is $\diamond \text{Nom } k$, replace it by $\diamond \text{Nom } j$ and if it is $\neg (@ k p)$ replace it by $\neg (@ j p)$. There are no other cases.

From this transformation we can show admissibility of the Bridge rule under the assumption that *j* is an allowed nominal.

10.1 Replacing

abbreviation *bridge'* :: $\langle 'b \Rightarrow 'b \Rightarrow ('a, 'b) \text{ fm} \Rightarrow ('a, 'b) \text{ fm} \rangle$ **where**
 $\langle \text{bridge}' k j p \equiv \text{case } p \text{ of}$
 $\quad (\diamond \text{Nom } k') \Rightarrow (\text{if } k = k' \text{ then } (\diamond \text{Nom } j) \text{ else } (\diamond \text{Nom } k'))$
 $\quad | (\neg (@ k' q)) \Rightarrow (\text{if } k = k' \text{ then } (\neg (@ j q)) \text{ else } (\neg (@ k' q)))$
 $\quad | p \Rightarrow p \rangle$

abbreviation *bridge* ::
 $\langle 'b \Rightarrow 'b \Rightarrow (\text{nat} \times \text{nat}) \text{ set} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow ('a, 'b) \text{ fm} \Rightarrow ('a, 'b) \text{ fm} \rangle$ **where**
 $\langle \text{bridge } k j \equiv \text{mapper } (\text{bridge}' k j) \rangle$

lemma *bridge-on-Nom*:
 $\langle \text{Nom } i \text{ on } (ps, a) \Longrightarrow \text{Nom } i \text{ on } (\text{mapi } (\text{bridge } k j \text{ xs } v) ps, a) \rangle$
 $\langle \text{proof} \rangle$

lemma *bridge'-nominals*:
 $\langle \text{nominals } (\text{bridge}' k j p) \cup \{k, j\} = \text{nominals } p \cup \{k, j\} \rangle$
 $\langle \text{proof} \rangle$

lemma *bridge-nominals*:
 $\langle \text{nominals } (\text{bridge } k j \text{ xs } v v' p) \cup \{k, j\} = \text{nominals } p \cup \{k, j\} \rangle$
 $\langle \text{proof} \rangle$

lemma *bridge-block-nominals*:
 $\langle \text{block-nominals } (\text{mapi-block } (\text{bridge } k j \text{ xs } v) (ps, a)) \cup \{k, j\} =$
 $\quad \text{block-nominals } (ps, a) \cup \{k, j\} \rangle$
 $\langle \text{proof} \rangle$

lemma *bridge-branch-nominals*:
 $\langle \text{branch-nominals } (\text{mapi-branch } (\text{bridge } k j \text{ xs}) \text{ branch}) \cup \{k, j\} =$
 $\quad \text{branch-nominals } \text{branch} \cup \{k, j\} \rangle$
 $\langle \text{proof} \rangle$

lemma *at-in-mapi-branch*:

assumes $\langle p \text{ at } a \text{ in branch} \rangle \langle p \neq \text{Nom } a \rangle$
shows $\langle \exists v v'. f v v' p \text{ at } a \text{ in mapi-branch } f \text{ branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *nom-at-in-bridge*:

fixes $k j xs$
defines $\langle f \equiv \text{bridge } k j xs \rangle$
assumes $\langle \text{Nom } i \text{ at } a \text{ in branch} \rangle$
shows $\langle \text{Nom } i \text{ at } a \text{ in mapi-branch } f \text{ branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *nominals-mapi-branch-bridge*:

assumes $\langle \text{Nom } k \text{ at } j \text{ in branch} \rangle$
shows $\langle \text{branch-nominals } (\text{mapi-branch } (\text{bridge } k j xs) \text{ branch}) = \text{branch-nominals } \text{branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *bridge-proper-dia*:

assumes $\langle \# a. p = \text{Nom } a \rangle$
shows $\langle \text{bridge } k j xs v v' (\diamond p) = (\diamond p) \rangle$
 $\langle \text{proof} \rangle$

lemma *bridge-compl-cases*:

fixes $k j xs v v' w w' p$
defines $\langle q \equiv \text{bridge } k j xs v v' p \rangle$ **and** $\langle q' \equiv \text{bridge } k j xs w w' (\neg p) \rangle$
shows
 $\langle (q = (\diamond \text{Nom } j) \wedge q' = (\neg (\diamond \text{Nom } k))) \vee$
 $(\exists r. q = (\neg (@ j r)) \wedge q' = (\neg \neg (@ k r))) \vee$
 $(\exists r. q = (@ k r) \wedge q' = (\neg (@ j r))) \vee$
 $(q = p \wedge q' = (\neg p)) \rangle$
 $\langle \text{proof} \rangle$

10.2 Descendants

inductive *descendants* :: $\langle 'b \Rightarrow 'b \Rightarrow ('a, 'b) \text{ branch} \Rightarrow (\text{nat} \times \text{nat}) \text{ set} \Rightarrow \text{bool} \rangle$
where

Initial:

$\langle \text{branch } !. v = \text{Some } (qs, i) \Longrightarrow qs !. v' = \text{Some } (\diamond \text{Nom } k) \Longrightarrow$
 $\text{descendants } k i \text{ branch } \{(v, v')\} \rangle$

| *Derived:*

$\langle \text{branch } !. v = \text{Some } (qs, a) \Longrightarrow qs !. v' = \text{Some } (\neg (@ k p)) \Longrightarrow$
 $\text{descendants } k i \text{ branch } xs \Longrightarrow (w, w') \in xs \Longrightarrow$
 $\text{branch } !. w = \text{Some } (rs, a) \Longrightarrow rs !. w' = \text{Some } (\diamond \text{Nom } k) \Longrightarrow$
 $\text{descendants } k i \text{ branch } (\{(v, v')\} \cup xs) \rangle$

| *Copied:*

$\langle \text{branch } !. v = \text{Some } (qs, a) \Longrightarrow qs !. v' = \text{Some } p \Longrightarrow$
 $\text{descendants } k i \text{ branch } xs \Longrightarrow (w, w') \in xs \Longrightarrow$

$\text{branch} \!. w = \text{Some } (rs, b) \implies rs \!. w' = \text{Some } p \implies$
 $\text{Nom } a \text{ at } b \text{ in branch} \implies$
 $\text{descendants } k \text{ i branch } (\{(v, v')\} \cup xs)$

lemma *descendants-initial:*

assumes $\langle \text{descendants } k \text{ i branch } xs \rangle$
shows $\langle \exists (v, v') \in xs. \exists ps.$
 $\text{branch} \!. v = \text{Some } (ps, i) \wedge ps \!. v' = \text{Some } (\diamond \text{Nom } k) \rangle$
 $\langle \text{proof} \rangle$

lemma *descendants-bounds-fst:*

assumes $\langle \text{descendants } k \text{ i branch } xs \rangle \langle (v, v') \in xs \rangle$
shows $\langle v < \text{length branch} \rangle$
 $\langle \text{proof} \rangle$

lemma *descendants-bounds-snd:*

assumes $\langle \text{descendants } k \text{ i branch } xs \rangle \langle (v, v') \in xs \rangle \langle \text{branch} \!. v = \text{Some } (ps, a) \rangle$
shows $\langle v' < \text{length } ps \rangle$
 $\langle \text{proof} \rangle$

lemma *descendants-branch:*

$\langle \text{descendants } k \text{ i branch } xs \implies \text{descendants } k \text{ i } (\text{extra } @ \text{ branch}) xs \rangle$
 $\langle \text{proof} \rangle$

lemma *descendants-block:*

assumes $\langle \text{descendants } k \text{ i } ((ps, a) \# \text{branch}) xs \rangle$
shows $\langle \text{descendants } k \text{ i } ((ps' @ ps, a) \# \text{branch}) xs \rangle$
 $\langle \text{proof} \rangle$

lemma *descendants-no-head:*

assumes $\langle \text{descendants } k \text{ i } ((ps, a) \# \text{branch}) xs \rangle$
shows $\langle \text{descendants } k \text{ i } ((p \# ps, a) \# \text{branch}) xs \rangle$
 $\langle \text{proof} \rangle$

lemma *descendants-types:*

assumes
 $\langle \text{descendants } k \text{ i branch } xs \rangle \langle (v, v') \in xs \rangle$
 $\langle \text{branch} \!. v = \text{Some } (ps, a) \rangle \langle ps \!. v' = \text{Some } p \rangle$
shows $\langle p = (\diamond \text{Nom } k) \vee (\exists q. p = (\neg (@ k q))) \rangle$
 $\langle \text{proof} \rangle$

lemma *descendants-oob-head':*

assumes $\langle \text{descendants } k \text{ i } ((ps, a) \# \text{branch}) xs \rangle$
shows $\langle (\text{length branch}, m + \text{length } ps) \notin xs \rangle$
 $\langle \text{proof} \rangle$

lemma *descendants-oob-head:*

assumes $\langle \text{descendants } k \text{ i } ((ps, a) \# \text{branch}) xs \rangle$
shows $\langle (\text{length branch}, \text{length } ps) \notin xs \rangle$

$\langle proof \rangle$

10.3 Induction

We induct over an arbitrary set of indices. That way, we can determine in each case whether the extension gets replaced or not by manipulating the set before applying the induction hypothesis.

lemma *STA-bridge'*:

fixes $a :: 'b$

assumes

$\langle inf: \langle infinite (UNIV :: 'b set) \rangle \text{ and } fin: \langle finite A \rangle \text{ and } \langle j \in A \rangle$

$\langle A, n \vdash (ps, a) \# branch \rangle$

$\langle descendants\ k\ i\ ((ps, a) \# branch)\ xs \rangle$

$\langle Nom\ k\ at\ j\ in\ branch \rangle$

shows $\langle A \vdash mapi\text{-}branch\ (bridge\ k\ j\ xs)\ ((ps, a) \# branch) \rangle$

$\langle proof \rangle$

lemma *STA-bridge*:

fixes $i :: 'b$

assumes

$\langle inf: \langle infinite (UNIV :: 'b set) \rangle \text{ and}$

$\langle A \vdash branch \rangle \langle descendants\ k\ i\ branch\ xs \rangle$

$\langle Nom\ k\ at\ j\ in\ branch \rangle$

$\langle finite\ A \rangle \langle j \in A \rangle$

shows $\langle A \vdash mapi\text{-}branch\ (bridge\ k\ j\ xs)\ branch \rangle$

$\langle proof \rangle$

10.4 Derivation

theorem *Bridge*:

fixes $i :: 'b$

assumes $\langle inf: \langle infinite (UNIV :: 'b set) \rangle \text{ and } fin: \langle finite A \rangle \text{ and } \langle j \in A \rangle$

$\langle Nom\ k\ at\ j\ in\ (ps, i) \# branch \rangle \langle (\Diamond\ Nom\ j)\ at\ i\ in\ (ps, i) \# branch \rangle$

$\langle A \vdash ((\Diamond\ Nom\ k) \# ps, i) \# branch \rangle$

shows $\langle A \vdash (ps, i) \# branch \rangle$

$\langle proof \rangle$

11 Completeness

11.1 Hintikka

abbreviation *at-in-set* $:: \langle ('a, 'b)\ fm \Rightarrow 'b \Rightarrow ('a, 'b)\ block\ set \Rightarrow bool \rangle$ **where**

$\langle at\text{-}in\text{-}set\ p\ a\ S \equiv \exists ps. (ps, a) \in S \wedge p\ on\ (ps, a) \rangle$

notation *at-in-set* $(\langle -\ at - in'' \rightarrow [51, 51, 51] 50)$

A set of blocks is Hintikka if it satisfies the following requirements. Intuitively, if it corresponds to an exhausted open branch with respect to the

fixed set of allowed nominals A . For example, we only require symmetry, "if j occurs at i then i occurs at j " if $i \in A$.

locale *Hintikka* =

fixes $A :: \langle 'b \text{ set} \rangle$ **and** $H :: \langle ('a, 'b) \text{ block set} \rangle$ **assumes**

ProP: $\langle \text{Nom } j \text{ at } i \text{ in}' H \implies \text{Pro } x \text{ at } j \text{ in}' H \implies \neg (\neg \text{Pro } x) \text{ at } i \text{ in}' H \rangle$ **and**

NomP: $\langle \text{Nom } a \text{ at } i \text{ in}' H \implies \neg (\neg \text{Nom } a) \text{ at } i \text{ in}' H \rangle$ **and**

NegN: $\langle (\neg \neg p) \text{ at } i \text{ in}' H \implies p \text{ at } i \text{ in}' H \rangle$ **and**

DisP: $\langle (p \vee q) \text{ at } i \text{ in}' H \implies p \text{ at } i \text{ in}' H \vee q \text{ at } i \text{ in}' H \rangle$ **and**

DisN: $\langle (\neg (p \vee q)) \text{ at } i \text{ in}' H \implies (\neg p) \text{ at } i \text{ in}' H \wedge (\neg q) \text{ at } i \text{ in}' H \rangle$ **and**

DiaP: $\langle \nexists a. p = \text{Nom } a \implies (\diamond p) \text{ at } i \text{ in}' H \implies$

$\exists j. (\diamond \text{Nom } j) \text{ at } i \text{ in}' H \wedge (@ j p) \text{ at } i \text{ in}' H \rangle$ **and**

DiaN: $\langle (\neg (\diamond p)) \text{ at } i \text{ in}' H \implies (\diamond \text{Nom } j) \text{ at } i \text{ in}' H \implies (\neg (@ j p)) \text{ at } i \text{ in}' H \rangle$ **and**

SatP: $\langle (@ i p) \text{ at } a \text{ in}' H \implies p \text{ at } i \text{ in}' H \rangle$ **and**

SatN: $\langle (\neg (@ i p)) \text{ at } a \text{ in}' H \implies (\neg p) \text{ at } i \text{ in}' H \rangle$ **and**

GoTo: $\langle i \in \text{nominals } p \implies \exists a. p \text{ at } a \text{ in}' H \implies \exists ps. (ps, i) \in H \rangle$ **and**

Nom: $\langle \forall a. p = \text{Nom } a \vee p = (\diamond \text{Nom } a) \longrightarrow a \in A \implies$

$p \text{ at } i \text{ in}' H \implies \text{Nom } j \text{ at } i \text{ in}' H \implies p \text{ at } j \text{ in}' H \rangle$

Two nominals i and j are equivalent in respect to a Hintikka set H if H contains an i -block with j on it. This is an equivalence relation on the names in H intersected with the allowed nominals A .

definition *hequiv* :: $\langle ('a, 'b) \text{ block set} \implies 'b \implies 'b \implies \text{bool} \rangle$ **where**

$\langle \text{hequiv } H \ i \ j \equiv \text{Nom } j \text{ at } i \text{ in}' H \rangle$

abbreviation *hequiv-rel* :: $\langle 'b \text{ set} \implies ('a, 'b) \text{ block set} \implies ('b \times 'b) \text{ set} \rangle$ **where**

$\langle \text{hequiv-rel } A \ H \equiv \{(i, j) \mid i \ j. \text{hequiv } H \ i \ j \wedge i \in A \wedge j \in A\} \rangle$

definition *names* :: $\langle ('a, 'b) \text{ block set} \implies 'b \text{ set} \rangle$ **where**

$\langle \text{names } H \equiv \{i \mid \exists ps \ i. (ps, i) \in H\} \rangle$

lemma *hequiv-refl*: $\langle i \in \text{names } H \implies \text{hequiv } H \ i \ i \rangle$

$\langle \text{proof} \rangle$

lemma *hequiv-refl'*: $\langle (ps, i) \in H \implies \text{hequiv } H \ i \ i \rangle$

$\langle \text{proof} \rangle$

lemma *hequiv-sym'*:

assumes $\langle \text{Hintikka } A \ H \rangle \langle i \in A \rangle \langle \text{hequiv } H \ i \ j \rangle$

shows $\langle \text{hequiv } H \ j \ i \rangle$

$\langle \text{proof} \rangle$

lemma *hequiv-sym*: $\langle \text{Hintikka } A \ H \implies i \in A \implies j \in A \implies \text{hequiv } H \ i \ j \longleftrightarrow$

$\text{hequiv } H \ j \ i \rangle$

$\langle \text{proof} \rangle$

lemma *hequiv-trans*:

assumes $\langle \text{Hintikka } A \ H \rangle \langle i \in A \rangle \langle k \in A \rangle \langle \text{hequiv } H \ i \ j \rangle \langle \text{hequiv } H \ j \ k \rangle$

shows $\langle \text{hequiv } H \ i \ k \rangle$

⟨proof⟩

lemma *hequiv-names*: ⟨ $hequiv\ H\ i\ j \implies i \in names\ H$ ⟩
⟨proof⟩

lemma *hequiv-names-rel*:
assumes ⟨*Hintikka* $A\ H$ ⟩
shows ⟨ $hequiv-rel\ A\ H \subseteq names\ H \times names\ H$ ⟩
⟨proof⟩

lemma *hequiv-refl-rel*:
assumes ⟨*Hintikka* $A\ H$ ⟩
shows ⟨*refl-on* ($names\ H \cap A$) (*hequiv-rel* $A\ H$)⟩
⟨proof⟩

lemma *hequiv-sym-rel*: ⟨*Hintikka* $A\ H \implies sym\ (hequiv-rel\ A\ H)$ ⟩
⟨proof⟩

lemma *hequiv-trans-rel*: ⟨*Hintikka* $B\ A \implies trans\ (hequiv-rel\ B\ A)$ ⟩
⟨proof⟩

lemma *hequiv-rel*: ⟨*Hintikka* $A\ H \implies equiv\ (names\ H \cap A)\ (hequiv-rel\ A\ H)$ ⟩
⟨proof⟩

lemma *nominal-in-names*:
assumes ⟨*Hintikka* $A\ H$ ⟩ ⟨ $\exists block \in H. i \in block-nominals\ block$ ⟩
shows ⟨ $i \in names\ H$ ⟩
⟨proof⟩

11.1.1 Named model

Given a Hintikka set H , a formula p on a block in H and a set of allowed nominals A which contains all "root-like" nominals in p we construct a model that satisfies p .

The worlds of our model are sets of equivalent nominals and nominals are assigned to the equivalence class of an equivalent allowed nominal. This definition resembles the "ur-father" notion.

From a world is , we can reach a world js iff there is an $i \in is$ and a $j \in js$ s.t. there is an i -block in H with $\diamond Nom\ j$ on it.

A propositional symbol p is true in a world is if there exists an $i \in is$ s.t. p occurs on an i -block in H .

definition *assign* :: ⟨ $'b\ set \Rightarrow ('a, 'b)\ block\ set \Rightarrow 'b \Rightarrow 'b\ set$ ⟩ **where**
⟨*assign* $A\ H\ i \equiv if\ \exists a. a \in A \wedge Nom\ a\ at\ i\ in'\ H$
 then *proj* (*hequiv-rel* $A\ H$) (*SOME* $a. a \in A \wedge Nom\ a\ at\ i\ in'\ H$)
 else $\{i\}$ ⟩

definition *reach* :: ⟨ $'b\ set \Rightarrow ('a, 'b)\ block\ set \Rightarrow 'b\ set \Rightarrow 'b\ set\ set$ ⟩ **where**

$\langle reach\ A\ H\ is \equiv \{ assign\ A\ H\ j \mid i\ j. i \in is \wedge (\Diamond\ Nom\ j)\ at\ i\ in'\ H \} \rangle$

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

lemma *ex-assignment*:

assumes $\langle Hintikka\ A\ H \rangle$
shows $\langle assign\ A\ H\ i \neq \{ \} \rangle$
 $\langle proof \rangle$

lemma *ur-closure*:

assumes $\langle Hintikka\ A\ H \rangle \langle p\ at\ i\ in'\ H \rangle \langle \forall a. p = Nom\ a \vee p = (\Diamond\ Nom\ a) \longrightarrow a \in A \rangle$
shows $\langle \forall a \in assign\ A\ H\ i. p\ at\ a\ in'\ H \rangle$
 $\langle proof \rangle$

lemma *ur-closure'*:

assumes $\langle Hintikka\ A\ H \rangle \langle p\ at\ i\ in'\ H \rangle \langle \forall a. p = Nom\ a \vee p = (\Diamond\ Nom\ a) \longrightarrow a \in A \rangle$
shows $\langle \exists a \in assign\ A\ H\ i. p\ at\ a\ in'\ H \rangle$
 $\langle proof \rangle$

lemma *mem-hequiv-rel*: $\langle a \in proj\ (hequiv-rel\ A\ H)\ b \implies a \in A \rangle$
 $\langle proof \rangle$

lemma *hequiv-proj*:

assumes $\langle Hintikka\ A\ H \rangle$
 $\langle Nom\ a\ at\ i\ in'\ H \rangle \langle a \in A \rangle \langle Nom\ b\ at\ i\ in'\ H \rangle \langle b \in A \rangle$
shows $\langle proj\ (hequiv-rel\ A\ H)\ a = proj\ (hequiv-rel\ A\ H)\ b \rangle$
 $\langle proof \rangle$

lemma *hequiv-proj-opening*:

assumes $\langle Hintikka\ A\ H \rangle \langle Nom\ a\ at\ i\ in'\ H \rangle \langle a \in A \rangle \langle i \in A \rangle$
shows $\langle proj\ (hequiv-rel\ A\ H)\ a = proj\ (hequiv-rel\ A\ H)\ i \rangle$
 $\langle proof \rangle$

lemma *assign-proj-refl*:

assumes $\langle Hintikka\ A\ H \rangle \langle Nom\ i\ at\ i\ in'\ H \rangle \langle i \in A \rangle$
shows $\langle assign\ A\ H\ i = proj\ (hequiv-rel\ A\ H)\ i \rangle$
 $\langle proof \rangle$

lemma *assign-named*:

assumes $\langle Hintikka\ A\ H \rangle \langle i \in proj\ (hequiv-rel\ A\ H)\ a \rangle$
shows $\langle i \in names\ H \rangle$
 $\langle proof \rangle$

lemma *assign-unique*:

assumes $\langle Hintikka\ A\ H \rangle \langle a \in assign\ A\ H\ i \rangle$
shows $\langle assign\ A\ H\ a = assign\ A\ H\ i \rangle$

⟨proof⟩

lemma *assign-val*:

assumes

⟨Hintikka A H⟩ ⟨Pro x at a in' H⟩ ⟨(¬ Pro x) at i in' H⟩

⟨a ∈ assign A H i⟩ ⟨i ∈ names H⟩

shows False

⟨proof⟩

lemma *Hintikka-model*:

assumes ⟨Hintikka A H⟩

shows

⟨p at i in' H ⇒ nominals p ⊆ A ⇒

Model (reach A H) (val H), assign A H, assign A H i ⊨ p⟩

⟨(¬ p) at i in' H ⇒ nominals p ⊆ A ⇒

¬ Model (reach A H) (val H), assign A H, assign A H i ⊨ p⟩

⟨proof⟩

11.2 Lindenbaum-Henkin

A set of blocks is consistent if no finite subset can be derived. Given a consistent set of blocks we are going to extend it to be saturated and maximally consistent and show that is then Hintikka. All definitions are with respect to the set of allowed nominals.

definition *consistent* :: ⟨'b set ⇒ ('a, 'b) block set ⇒ bool⟩ **where**

⟨consistent A S ≡ ∄ S'. set S' ⊆ S ∧ A ⊢ S'⟩

instance *fm* :: (countable, countable) countable

⟨proof⟩

definition *proper-dia* :: ⟨('a, 'b) fm ⇒ ('a, 'b) fm option⟩ **where**

⟨proper-dia p ≡ case p of (◇ p) ⇒ (if ∄ a. p = Nom a then Some p else None) | - ⇒ None⟩

lemma *proper-dia*: ⟨proper-dia p = Some q ⇒ p = (◇ q) ∧ (∄ a. q = Nom a)⟩

⟨proof⟩

The following function witnesses each ◇ p in a fresh world.

primrec *witness-list* :: ⟨('a, 'b) fm list ⇒ 'b set ⇒ ('a, 'b) fm list⟩ **where**

⟨witness-list [] - = []⟩

| ⟨witness-list (p # ps) used =

(case proper-dia p of

None ⇒ witness-list ps used

| Some q ⇒

let i = SOME i. i ∉ used

in (@ i q) # (◇ Nom i) # witness-list ps ({i} ∪ used)⟩

primrec *witness* :: ⟨('a, 'b) block ⇒ 'b set ⇒ ('a, 'b) block⟩ **where**

$\langle \text{witness } (ps, a) \text{ used} = (\text{witness-list } ps \text{ used}, a) \rangle$

lemma *witness-list*:

$\langle \text{proper-dia } p = \text{Some } q \implies \text{witness-list } (p \# ps) \text{ used} =$
 $(\text{let } i = \text{SOME } i. i \notin \text{used}$
 $\text{in } (@ i q) \# (\diamond \text{Nom } i) \# \text{witness-list } ps (\{i\} \cup \text{used})) \rangle$
 $\langle \text{proof} \rangle$

primrec *extend* ::

$\langle 'b \text{ set} \implies ('a, 'b) \text{ block set} \implies (\text{nat} \implies ('a, 'b) \text{ block}) \implies \text{nat} \implies ('a, 'b) \text{ block set} \rangle$

where

$\langle \text{extend } A \text{ S } f \ 0 = S \rangle$
 $| \langle \text{extend } A \text{ S } f \ (\text{Suc } n) =$
 $(\text{if } \neg \text{consistent } A (\{f\ n\} \cup \text{extend } A \text{ S } f \ n)$
 $\text{then } \text{extend } A \text{ S } f \ n$
 else
 $\text{let } \text{used} = A \cup (\bigcup \text{block} \in \{f\ n\} \cup \text{extend } A \text{ S } f \ n. \text{block-nominals } \text{block})$
 $\text{in } \{f\ n, \text{witness } (f\ n) \text{ used}\} \cup \text{extend } A \text{ S } f \ n \rangle$

definition *Extend* ::

$\langle 'b \text{ set} \implies ('a, 'b) \text{ block set} \implies (\text{nat} \implies ('a, 'b) \text{ block}) \implies ('a, 'b) \text{ block set} \rangle$ **where**
 $\langle \text{Extend } A \text{ S } f \equiv (\bigcup n. \text{extend } A \text{ S } f \ n) \rangle$

lemma *extend-chain*: $\langle \text{extend } A \text{ S } f \ n \subseteq \text{extend } A \text{ S } f \ (\text{Suc } n) \rangle$
 $\langle \text{proof} \rangle$

lemma *extend-mem*: $\langle S \subseteq \text{extend } A \text{ S } f \ n \rangle$
 $\langle \text{proof} \rangle$

lemma *Extend-mem*: $\langle S \subseteq \text{Extend } A \text{ S } f \rangle$
 $\langle \text{proof} \rangle$

11.2.1 Consistency

lemma *split-list*:

$\langle \text{set } A \subseteq \{x\} \cup X \implies x \in A \implies \exists B. \text{set } (x \# B) = \text{set } A \wedge x \notin \text{set } B \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-drop-single*:

fixes $a :: 'b$
assumes
 $\text{inf}: \langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$ **and**
 $\text{fin}: \langle \text{finite } A \rangle$ **and**
 $\text{cons}: \langle \text{consistent } A (\{(p \# ps, a)\} \cup S) \rangle$
shows $\langle \text{consistent } A (\{(ps, a)\} \cup S) \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-drop-block*: $\langle \text{consistent } A (\{\text{block}\} \cup S) \implies \text{consistent } A \ S \rangle$
 $\langle \text{proof} \rangle$

lemma *inconsistent-weaken*: $\langle \neg \text{consistent } A \ S \implies S \subseteq S' \implies \neg \text{consistent } A \ S' \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-nominals-set*: $\langle \text{finite } S \implies \text{finite } (\bigcup \text{block} \in S. \text{block-nominals } \text{block}) \rangle$
 $\langle \text{proof} \rangle$

lemma *witness-list-used*:
fixes $i :: 'b$
assumes $\text{inf}: \langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$ **and** $\langle \text{finite } \text{used} \rangle$ $\langle i \notin \text{list-nominals } \text{ps} \rangle$
shows $\langle i \notin \text{list-nominals } (\text{witness-list } \text{ps } (\{i\} \cup \text{used})) \rangle$
 $\langle \text{proof} \rangle$

lemma *witness-used*:
fixes $i :: 'b$
assumes $\text{inf}: \langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$ **and**
 $\langle \text{finite } \text{used} \rangle$ $\langle i \notin \text{block-nominals } \text{block} \rangle$
shows $\langle i \notin \text{block-nominals } (\text{witness } \text{block } (\{i\} \cup \text{used})) \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-witness-list*:
fixes $a :: 'b$
assumes $\text{inf}: \langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$ **and** $\langle \text{consistent } A \ S \rangle$
 $\langle (ps, a) \in S \rangle$ $\langle \text{finite } \text{used} \rangle$ $\langle A \cup \bigcup (\text{block-nominals } 'S) \subseteq \text{used} \rangle$
shows $\langle \text{consistent } A \ (\{(\text{witness-list } \text{ps } \text{used}, a)\} \cup S) \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-witness*:
fixes $\text{block} :: \langle ('a, 'b) \text{ block} \rangle$
assumes $\langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$
 $\langle \text{consistent } A \ S \rangle$ $\langle \text{finite } (\bigcup (\text{block-nominals } 'S)) \rangle$ $\langle \text{block} \in S \rangle$ $\langle \text{finite } A \rangle$
shows $\langle \text{consistent } A \ (\{\text{witness } \text{block } (A \cup \bigcup (\text{block-nominals } 'S))\} \cup S) \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-extend*:
fixes $S :: \langle ('a, 'b) \text{ block set} \rangle$
assumes $\text{inf}: \langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$ **and** $\text{fin}: \langle \text{finite } A \rangle$ **and**
 $\langle \text{consistent } A \ (\text{extend } A \ S \ \text{f } n) \rangle$ $\langle \text{finite } (\bigcup (\text{block-nominals } ' \text{extend } A \ S \ \text{f } n)) \rangle$
shows $\langle \text{consistent } A \ (\text{extend } A \ S \ \text{f } (\text{Suc } n)) \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-nominals-extend*:
assumes $\langle \text{finite } (\bigcup (\text{block-nominals } 'S)) \rangle$
shows $\langle \text{finite } (\bigcup (\text{block-nominals } ' \text{extend } A \ S \ \text{f } n)) \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-extend'*:
fixes $S :: \langle ('a, 'b) \text{ block set} \rangle$
assumes $\langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$ $\langle \text{finite } A \rangle$ $\langle \text{consistent } A \ S \rangle$ $\langle \text{finite } (\bigcup$

$\langle \text{block-nominals } 'S \rangle$
shows $\langle \text{consistent } A \text{ (extend } A \text{ } S \text{ } f \text{ } n) \rangle$
 $\langle \text{proof} \rangle$

lemma *UN-finite-bound*:
assumes $\langle \text{finite } A \rangle \langle A \subseteq (\bigcup n. f \text{ } n) \rangle$
shows $\langle \exists m :: \text{nat. } A \subseteq (\bigcup n \leq m. f \text{ } n) \rangle$
 $\langle \text{proof} \rangle$

lemma *extend-bound*: $\langle (\bigcup n \leq m. \text{extend } A \text{ } S \text{ } f \text{ } n) = \text{extend } A \text{ } S \text{ } f \text{ } m \rangle$
 $\langle \text{proof} \rangle$

lemma *consistent-Extend*:
fixes $S :: \langle 'a, 'b \rangle \text{ block set} \rangle$
assumes *inf*: $\langle \text{infinite } (UNIV :: 'b \text{ set}) \rangle$ **and** $\langle \text{finite } A \rangle$
 $\langle \text{consistent } A \text{ } S \rangle \langle \text{finite } (\bigcup (\text{block-nominals } 'S)) \rangle$
shows $\langle \text{consistent } A \text{ (Extend } A \text{ } S \text{ } f) \rangle$
 $\langle \text{proof} \rangle$

11.2.2 Maximality

A set of blocks is maximally consistent if any proper extension makes it inconsistent.

definition *maximal* :: $\langle 'b \text{ set} \Rightarrow ('a, 'b) \text{ block set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{maximal } A \text{ } S \equiv \text{consistent } A \text{ } S \wedge (\forall \text{block. block} \notin S \longrightarrow \neg \text{consistent } A \text{ } (\{\text{block}\} \cup S)) \rangle$

lemma *extend-not-mem*:
 $\langle f \text{ } n \notin \text{extend } A \text{ } S \text{ } f \text{ } (Suc \text{ } n) \implies \neg \text{consistent } A \text{ } (\{f \text{ } n\} \cup \text{extend } A \text{ } S \text{ } f \text{ } n) \rangle$
 $\langle \text{proof} \rangle$

lemma *maximal-Extend*:
fixes $S :: \langle ('a, 'b) \text{ block set} \rangle$
assumes *inf*: $\langle \text{infinite } (UNIV :: 'b \text{ set}) \rangle$ **and** $\langle \text{finite } A \rangle$
 $\langle \text{consistent } A \text{ } S \rangle \langle \text{finite } (\bigcup (\text{block-nominals } 'S)) \rangle \langle \text{surj } f \rangle$
shows $\langle \text{maximal } A \text{ (Extend } A \text{ } S \text{ } f) \rangle$
 $\langle \text{proof} \rangle$

11.2.3 Saturation

A set of blocks is saturated if every $\diamond p$ is witnessed.

definition *saturated* :: $\langle ('a, 'b) \text{ block set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{saturated } S \equiv \forall p \text{ } i. (\diamond p) \text{ at } i \text{ in } S \longrightarrow (\exists a. p = \text{Nom } a) \longrightarrow$
 $(\exists j. (@ j p) \text{ at } i \text{ in } S \wedge (\diamond \text{Nom } j) \text{ at } i \text{ in } S) \rangle$

lemma *witness-list-append*:
 $\langle \exists \text{extra. witness-list } (ps @ qs) \text{ used} = \text{witness-list } ps \text{ used} @ \text{witness-list } qs \text{ (extra} \cup \text{used)} \rangle$

⟨proof⟩

lemma *ex-witness-list*:

assumes ⟨ $p \in . ps$ ⟩ ⟨proper-dia $p = \text{Some } q$ ⟩
shows ⟨ $\exists i. \{ @ i q, \diamond \text{Nom } i \} \subseteq \text{set } (\text{witness-list } ps \text{ used})$ ⟩
⟨proof⟩

lemma *saturated-Extend*:

fixes $S :: \langle 'a, 'b \rangle \text{ block set}$
assumes *inf*: ⟨infinite (UNIV :: 'b set)⟩ **and** *fin*: ⟨finite A ⟩ **and**
⟨consistent $A S$ ⟩ ⟨finite $(\bigcup (\text{block-nominals } 'S))$ ⟩ ⟨surj f ⟩
shows ⟨saturated (Extend $A S f$)⟩
⟨proof⟩

11.3 Smullyan-Fitting

lemma *Hintikka-Extend*:

fixes $S :: \langle 'a, 'b \rangle \text{ block set}$
assumes *inf*: ⟨infinite (UNIV :: 'b set)⟩ **and** *fin*: ⟨finite A ⟩ **and**
⟨maximal $A S$ ⟩ ⟨consistent $A S$ ⟩ ⟨saturated S ⟩
shows ⟨Hintikka $A S$ ⟩
⟨proof⟩

11.4 Result

theorem *completeness*:

fixes $p :: \langle 'a :: \text{countable}, 'b :: \text{countable} \rangle \text{ fm}$
assumes
 inf: ⟨infinite (UNIV :: 'b set)⟩ **and**
 valid: ⟨ $\forall (M :: \langle 'b \text{ set}, 'a \rangle \text{ model}) g w. M, g, w \models p$ ⟩
shows ⟨nominals $p, 1 \vdash [(\neg p), i]$ ⟩
⟨proof⟩

We arbitrarily fix nominal and propositional symbols to be natural numbers (any countably infinite type suffices) and define validity as truth in all models with sets of natural numbers as worlds. We show below that this implies validity for any type of worlds.

abbreviation

⟨valid $p \equiv \forall (M :: (\text{nat set}, \text{nat}) \text{ model}) (g :: \text{nat} \Rightarrow -) w. M, g, w \models p$ ⟩

A formula is valid iff its negation has a closing tableau from a fresh world. We can assume a single unit of potential and take the allowed nominals to be the root nominals.

theorem *main*:

assumes ⟨ $i \notin \text{nominals } p$ ⟩
shows ⟨valid $p \longleftrightarrow \text{nominals } p, 1 \vdash [(\neg p), i]$ ⟩
⟨proof⟩

The restricted validity implies validity in general.

theorem *valid-semantics*:
⟨*valid* $p \longrightarrow M, g, w \models p$ ⟩
⟨*proof*⟩

end

References

- [1] P. Blackburn, T. Bolander, T. Braüner, and K. F. Jørgensen. Completeness and Termination for a Seligman-style Tableau System. *Journal of Logic and Computation*, 27(1):81–107, 2017.
- [2] K. F. Jørgensen, P. Blackburn, T. Bolander, and T. Braüner. Synthetic Completeness Proofs for Seligman-style Tableau Systems. In *Advances in Modal Logic*, volume 11, pages 302–321, 2016.