

# Formalizing a Seligman-Style Tableau System for Hybrid Logic

Asta Halkjr From

October 24, 2020

## 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:

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

# Contents

|           |  |            |
|-----------|--|------------|
| <b>1</b>  | <b>Syntax</b>                              | <b>3</b>   |
| <b>2</b>  | <b>Semantics</b>                           | <b>4</b>   |
| 2.1       | Examples . . . . .                         | 4          |
| <b>3</b>  | <b>Tableau</b>                             | <b>5</b>   |
| <b>4</b>  | <b>Soundness</b>                           | <b>8</b>   |
| <b>5</b>  | <b>No Detours</b>                          | <b>11</b>  |
| 5.1       | Free GoTo . . . . .                        | 18         |
| <b>6</b>  | <b>Indexed Mapping</b>                     | <b>18</b>  |
| 6.1       | Indexing . . . . .                         | 18         |
| 6.2       | Mapping . . . . .                          | 20         |
| <b>7</b>  | <b>Duplicate Formulas</b>                  | <b>22</b>  |
| 7.1       | Removable indices . . . . .                | 22         |
| 7.2       | Omitting formulas . . . . .                | 23         |
| 7.3       | Induction . . . . .                        | 30         |
| 7.4       | Unrestricted rules . . . . .               | 35         |
| <b>8</b>  | <b>Substitution</b>                        | <b>37</b>  |
| 8.1       | Unrestricted ( $\diamond$ ) rule . . . . . | 45         |
| <b>9</b>  | <b>Structural Properties</b>               | <b>46</b>  |
| <b>10</b> | <b>Bridge</b>                              | <b>54</b>  |
| 10.1      | Replacing . . . . .                        | 54         |
| 10.2      | Descendants . . . . .                      | 57         |
| 10.3      | Induction . . . . .                        | 60         |
| 10.4      | Derivation . . . . .                       | 75         |
| <b>11</b> | <b>Completeness</b>                        | <b>76</b>  |
| 11.1      | Hintikka . . . . .                         | 76         |
| 11.1.1    | Named model . . . . .                      | 78         |
| 11.2      | Lindenbaum-Henkin . . . . .                | 86         |
| 11.2.1    | Consistency . . . . .                      | 87         |
| 11.2.2    | Maximality . . . . .                       | 94         |
| 11.2.3    | Saturation . . . . .                       | 94         |
| 11.3      | Smullyan-Fitting . . . . .                 | 97         |
| 11.4      | Result . . . . .                           | 103        |
|           | <b>References</b>                          | <b>105</b> |

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

## 1 Syntax

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

We can give other connectives as abbreviations.

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

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

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

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

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

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

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

**lemma** *sub-id*:  $\langle \text{sub } \text{id } p \rangle = p$   
**by** (*induct*  $p$ ) *simp-all*

**lemma** *sub-upd-fresh*:  $\langle i \notin \text{nominals } p \implies \text{sub } (f(i := j)) p = \text{sub } f p \rangle$   
**by** (*induct*  $p$ ) *auto*

## 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**

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

**lemma**  $\langle M, g, w \models \top \rangle$   
**by** *simp*

**lemma semantics-fresh:**

$\langle i \notin \text{nominals } p \implies (M, g, w \models p) = (M, g(i := v), w \models p) \rangle$   
**by** *(induct p arbitrary: w) auto*

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

**proof** –

**assume**  $\langle \text{irreflexive } M \rangle$

**then have**  $\langle g i \notin R M (g i) \rangle$

**by** *simp*

**then have**  $\langle \neg M, g, g i \models \Diamond \text{Nom } i \rangle$

**by** *simp*

**then have**  $\langle M, g, g i \models \neg (\Diamond \text{Nom } i) \rangle$

**by** *simp*

**then show**  $\langle M, g, w \models @ i \neg (\Diamond \text{Nom } i) \rangle$

**by** *simp*

**qed**

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

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

**by** *auto*

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

**by** *auto*

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

**by** *auto*

### 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**  $\langle 'a, 'b \rangle \text{ block} = \langle ('a, 'b) \text{ fm list} \times 'b \rangle$

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

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

$\langle x \in \cdot. xs \equiv x \in \text{set } xs \rangle$

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

**primrec**  $\text{on} :: \langle ('a, 'b) \text{ fm} \Rightarrow ('a, 'b) \text{ block} \Rightarrow \text{bool} \rangle (\langle \cdot \text{ on } \cdot \rangle [51, 51] 50)$  **where**

$\langle p \text{ on } (ps, i) = (p \in ps \vee p = \text{Nom } i) \rangle$

**syntax**

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

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

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

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

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

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

$\langle \text{new } p \text{ a branch} \equiv \neg p \text{ at } a \text{ in branch} \rangle$

**definition** *witnessed* ::  $\langle 'a, 'b \rangle \text{ fm} \Rightarrow 'b \Rightarrow \langle 'a, 'b \rangle \text{ branch} \Rightarrow \text{bool} \rangle$  **where**

$\langle \text{witnessed } p \text{ a branch} \equiv \exists i. (@ i p) \text{ at } a \text{ in branch} \wedge (\diamond \text{Nom } i) \text{ at } a \text{ in branch} \rangle$

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  $\text{Nom}$ . This set rules out the copying of nominals and accessibility formulas introduced by  $\text{DiaP}$ . The second argument is "potential", used to restrict the  $\text{GoTo}$  rule.

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

**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:*

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

$A, n \vdash (ps, a) \# \text{branch}$   
| *DiaN*:  
 $\langle (\neg (\diamond p)) \text{ at } a \text{ in } (ps, a) \# \text{branch} \implies$   
 $\langle \diamond \text{Nom } i \text{ at } a \text{ in } (ps, a) \# \text{branch} \implies$   
 $\text{new } (\neg (@ i p)) a ((ps, a) \# \text{branch}) \implies$   
 $A, \text{Suc } n \vdash ((\neg (@ i p)) \# ps, a) \# \text{branch} \implies$   
 $A, n \vdash (ps, a) \# \text{branch}$   
| *SatP*:  
 $\langle (@ a p) \text{ at } b \text{ in } (ps, a) \# \text{branch} \implies$   
 $\text{new } p a ((ps, a) \# \text{branch}) \implies$   
 $A, \text{Suc } n \vdash (p \# ps, a) \# \text{branch} \implies$   
 $A, n \vdash (ps, a) \# \text{branch}$   
| *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* ::  $\langle 'b \text{ set} \implies ('a, 'b) \text{ branch} \implies \text{bool} \rangle (\langle - \vdash - \rangle [50, 50])$  **where**

$\langle A \vdash \text{branch} \equiv \exists n. A, n \vdash \text{branch} \rangle$

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

**by** (*induct*  $n$  *branch* rule: *STA.induct*) (*simp-all* add: *STA.intros*)

A verified derivation in the calculus.

**lemma**

**fixes**  $i$

**defines**  $\langle p \equiv \neg (@ i (\text{Nom } i)) \rangle$

**shows**  $\langle A, \text{Suc } n \vdash [([p], a)] \rangle$

**proof** –

**have**  $\langle i \in \text{branch-nominals } [([p], a)] \rangle$

**unfolding** *p-def* *branch-nominals-def* **by** *simp*

**then have** *?thesis* **if**  $\langle A, n \vdash [([], i), ([p], a)] \rangle$

**using** *that* *GoTo* **by** *fast*

**moreover have**  $\langle \text{new } (\neg \text{Nom } i) i [([], i), ([p], a)] \rangle$

**unfolding** *p-def* *new-def* **by** *auto*

**moreover have**  $\langle (\neg (@ i (\text{Nom } i))) \text{ at } a \text{ in } [([], i), ([p], a)] \rangle$

**unfolding** *p-def* **by** *fastforce*

**ultimately have** *?thesis* if  $\langle A, \text{Suc } n \vdash [(\neg \text{Nom } i), i], ([p], a) \rangle$   
**using** *that SatN by fast*  
**then show** *?thesis*  
**by** (*meson Close list.set-intros(1) on.simps*)  
**qed**

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

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

**lemma** *block-nominals*:  
 $\langle p \text{ on block} \Longrightarrow i \in \text{nominals } p \Longrightarrow i \in \text{block-nominals block} \rangle$   
**by** (*induct block*) *auto*

**lemma** *block-sat-fresh*:  
**assumes**  $\langle M, g \models_B \text{block} \rangle \langle i \notin \text{block-nominals block} \rangle$   
**shows**  $\langle M, g(i := v) \models_B \text{block} \rangle$   
**using** *assms*  
**proof** (*induct block*)  
**case** (*Pair ps a*)  
**then have**  $\langle \forall p \text{ on } (ps, a). i \notin \text{nominals } p \rangle$   
**using** *block-nominals by fast*  
**moreover have**  $\langle i \neq a \rangle$   
**using** *calculation by simp*  
**ultimately have**  $\langle \forall p \text{ on } (ps, a). M, g(i := v), (g(i := v))\ a \models p \rangle$   
**using** *Pair semantics-fresh by fastforce*  
**then show** *?case*  
**by** (*meson block-sat.simps*)  
**qed**

**lemma** *branch-sat-fresh*:  
**assumes**  $\langle M, g \models_{\Theta} \text{branch} \rangle \langle i \notin \text{branch-nominals branch} \rangle$   
**shows**  $\langle M, g(i := v) \models_{\Theta} \text{branch} \rangle$   
**using** *assms using block-sat-fresh unfolding branch-nominals-def by fast*

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

**lemma** *soundness'*:  $\langle A, n \vdash \text{branch} \Longrightarrow M, g \models_{\Theta} \text{branch} \Longrightarrow \text{False} \rangle$   
**proof** (*induct n branch arbitrary: g rule: STA.induct*)



```

case (Close p i branch)
then have  $\langle M, g, g \ i \models p \rangle \langle M, g, g \ i \models \neg p \rangle$ 
  by fastforce+
then show ?case
  by simp
next
case (Neg p a ps branch)
have  $\langle M, g, g \ a \models p \rangle$ 
  using Neg(1, 5) by fastforce
then have  $\langle M, g \models_{\ominus} (p \ \# \ ps, a) \ \# \ branch \rangle$ 
  using Neg(5) by simp
then show ?case
  using Neg(4) by blast
next
case (DisP p q a ps branch)
consider  $\langle M, g, g \ a \models p \rangle \mid \langle M, g, g \ a \models q \rangle$ 
  using DisP(1, 8) by fastforce
then consider
   $\langle M, g \models_{\ominus} (p \ \# \ ps, a) \ \# \ branch \rangle \mid$ 
   $\langle M, g \models_{\ominus} (q \ \# \ ps, a) \ \# \ branch \rangle$ 
  using DisP(8) by auto
then show ?case
  using DisP(5, 7) by metis
next
case (DisN p q a ps branch)
have  $\langle M, g, g \ a \models \neg p \rangle \langle M, g, g \ a \models \neg q \rangle$ 
  using DisN(1, 5) by fastforce+
then have  $\langle M, g \models_{\ominus} ((\neg q) \ \# \ (\neg p) \ \# \ ps, a) \ \# \ branch \rangle$ 
  using DisN(5) by simp
then show ?case
  using DisN(4) by blast
next
case (DiaP p a ps branch i)
then have *:  $\langle M, g \models_B (ps, a) \rangle$ 
  by simp

have  $\langle i \notin \text{nominals } p \rangle$ 
  using DiaP(1-2) unfolding branch-nominals-def by fastforce

have  $\langle M, g, g \ a \models \diamond p \rangle$ 
  using DiaP(1, 7) by fastforce
then obtain  $v$  where  $\langle v \in R \ M \ (g \ a) \rangle \langle M, g, v \models p \rangle$ 
  by auto
then have  $\langle M, g(i := v), v \models p \rangle$ 
  using  $\langle i \notin \text{nominals } p \rangle$  semantics-fresh by metis
then have  $\langle M, g(i := v), g \ a \models @ \ i \ p \rangle$ 
  by simp
moreover have  $\langle M, g(i := v), g \ a \models \diamond \text{Nom } i \rangle$ 
  using  $\langle v \in R \ M \ (g \ a) \rangle$  by simp

```

**moreover have**  $\langle M, g(i := v) \models_{\Theta} (ps, a) \# \text{branch} \rangle$   
**using** *DiaP(2, 7) branch-sat-fresh* **by** *fast*  
**moreover have**  $\langle i \notin \text{block-nominals } (ps, a) \rangle$   
**using** *DiaP(2) unfolding branch-nominals-def* **by** *simp*  
**then have**  $\langle \forall p \text{ on } (ps, a). M, g(i := v), g a \models p \rangle$   
**using** *\* semantics-fresh* **by** *fastforce*  
**ultimately have**  
 $\langle M, g(i := v) \models_{\Theta} ((@ i p) \# (\diamond \text{Nom } i) \# ps, a) \# \text{branch} \rangle$   
**by** *auto*  
**then show** *?case*  
**using** *DiaP* **by** *blast*  
**next**  
**case** (*DiaN p a ps branch i*)  
**have**  $\langle M, g, g a \models \neg (\diamond p) \rangle \langle M, g, g a \models \diamond \text{Nom } i \rangle$   
**using** *DiaN(1-2, 6)* **by** *fastforce+*  
**then have**  $\langle M, g, g a \models \neg (@ i p) \rangle$   
**by** *simp*  
**then have**  $\langle M, g \models_{\Theta} ((\neg (@ i p)) \# ps, a) \# \text{branch} \rangle$   
**using** *DiaN(6)* **by** *simp*  
**then show** *?thesis*  
**using** *DiaN(5)* **by** *blast*  
**next**  
**case** (*SatP a p b ps branch*)  
**have**  $\langle M, g, g a \models p \rangle$   
**using** *SatP(1, 5)* **by** *fastforce*  
**then have**  $\langle M, g \models_{\Theta} (p \# ps, a) \# \text{branch} \rangle$   
**using** *SatP(5)* **by** *simp*  
**then show** *?case*  
**using** *SatP(4)* **by** *blast*  
**next**  
**case** (*SatN a p b ps branch*)  
**have**  $\langle M, g, g a \models \neg p \rangle$   
**using** *SatN(1, 5)* **by** *fastforce*  
**then have**  $\langle M, g \models_{\Theta} ((\neg p) \# ps, a) \# \text{branch} \rangle$   
**using** *SatN(5)* **by** *simp*  
**then show** *?case*  
**using** *SatN(4)* **by** *blast*  
**next**  
**case** (*GoTo i branch*)  
**then show** *?case*  
**by** *auto*  
**next**  
**case** (*Nom p b ps a branch*)  
**have**  $\langle M, g, g b \models p \rangle \langle M, g, g b \models \text{Nom } a \rangle$   
**using** *Nom(1-2, 7)* **by** *fastforce+*  
**moreover have**  $\langle M, g \models_B (ps, a) \rangle$   
**using** *Nom(7)* **by** *simp*  
**ultimately have**  $\langle M, g \models_B (p \# ps, a) \rangle$   
**by** *simp*

**then have**  $\langle M, g \models_{\Theta} (p \# ps, a) \# \text{branch} \rangle$   
**using** *Nom(7)* **by** *simp*  
**then show** *?case*  
**using** *Nom(6)* **by** *blast*  
**qed**

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

**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**  $\langle M, g \models_{\Theta} \text{branch} \rangle$   
**using** *assms block-sat* **by** *fast*

**lemma** *soundness*:  
**assumes**  $\langle A, n \vdash \text{branch} \rangle$   
**shows**  $\langle \exists \text{block} \in \text{set branch. } \exists p \text{ on block. } \neg M, g, w \models p \rangle$   
**using** *assms soundness' branch-sat* **by** *fast*

**corollary**  $\langle \neg A, n \vdash [] \rangle$   
**using** *soundness* **by** *fastforce*

**theorem** *soundness-fresh*:  
**assumes**  $\langle A, n \vdash [(\neg p), i] \rangle \langle i \notin \text{nominals } p \rangle$   
**shows**  $\langle M, g, w \models p \rangle$   
**proof** –  
**from** *assms(1)* **have**  $\langle M, g, g \ i \models p \rangle$  **for** *g*  
**using** *soundness* **by** *fastforce*  
**then have**  $\langle M, g(i := w), (g(i := w)) \ i \models p \rangle$   
**by** *blast*  
**then have**  $\langle M, g(i := w), w \models p \rangle$   
**by** *simp*  
**then have**  $\langle M, g(i := g \ i), w \models p \rangle$   
**using** *assms(2)* *semantics-fresh* **by** *metis*  
**then show** *?thesis*  
**by** *simp*  
**qed**

## 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**  $\langle A, \text{Suc } n \vdash \text{filter nonempty } ((ps, a) \# \text{left}) @ \text{right} \rangle$   
**proof** (cases ps)  
**case** Nil  
**then have**  $\langle a \in \text{branch-nominals (filter nonempty left @ right)} \rangle$   
**unfolding** branch-nominals-def **using** assms(2-3) **by** fastforce  
**then show** ?thesis  
**using** assms(1) Nil GoTo **by** auto  
**next**  
**case** Cons  
**then show** ?thesis  
**using** assms(1) STA-Suc **by** auto  
**qed**

**lemma** STA-nonempty:  
 $\langle A, n \vdash \text{left @ right} \implies A, \text{Suc } m \vdash \text{filter nonempty left @ right} \rangle$   
**proof** (induct n <left @ right> arbitrary: left right rule: STA.induct)  
**case** (Close p i n)  
**have**  $\langle (\neg p) \text{ at } i \text{ in filter nonempty left @ right} \rangle$   
**using** Close(2) **by** fastforce  
**moreover from this have**  $\langle p \text{ at } i \text{ in filter nonempty left @ right} \rangle$   
**using** Close(1) **by** fastforce  
**ultimately show** ?case  
**using** STA.Close **by** fast  
**next**  
**case** (Neg p a ps branch n)  
**then show** ?case  
**proof** (cases left)  
**case** Nil  
**then have**  $\langle A, \text{Suc } m \vdash (p \# ps, a) \# \text{branch} \rangle$   
**using** Neg(4) **by** fastforce  
**then have**  $\langle A, m \vdash (ps, a) \# \text{branch} \rangle$   
**using** Neg(1-2) STA.Neg **by** fast  
**then show** ?thesis  
**using** Nil Neg(5) STA-Suc **by** auto  
**next**  
**case** (Cons - left')  
**then have**  $\langle A, \text{Suc } m \vdash (p \# ps, a) \# \text{filter nonempty left' @ right} \rangle$   
**using** Neg(4)[**where** left= $\langle - \# \text{left}' \rangle$ ] Neg(5) **by** fastforce  
**moreover have** \*:  $\langle (\neg \neg p) \text{ at } a \text{ in } (ps, a) \# \text{filter nonempty left' @ right} \rangle$   
**using** Cons Neg(1, 5) **by** fastforce  
**moreover have**  $\langle \text{new } p \text{ a } ((ps, a) \# \text{filter nonempty left' @ right}) \rangle$   
**using** Cons Neg(2, 5) **unfolding** new-def **by** auto  
**ultimately have**  $\langle A, m \vdash (ps, a) \# \text{filter nonempty left' @ right} \rangle$   
**using** STA.Neg **by** fast  
**then have**  $\langle A, \text{Suc } m \vdash \text{filter nonempty } ((ps, a) \# \text{left}') @ \text{right} \rangle$   
**using** \* nonempty-Suc **by** fast

```

    then show ?thesis
      using Cons Neg(5) by auto
  qed
next
case (DisP p q a ps branch n)
then show ?case
proof (cases left)
  case Nil
  then have ⟨A, Suc m ⊢ (p # ps, a) # branch⟩ ⟨A, Suc m ⊢ (q # ps, a) #
branch⟩
    using DisP(5, 7) by fastforce+
  then have ⟨A, m ⊢ (ps, a) # branch⟩
    using DisP(1-3) STA.DisP by fast
  then show ?thesis
    using Nil DisP(8) STA-Suc by auto
next
case (Cons - left')
then have
  ⟨A, Suc m ⊢ (p # ps, a) # filter nonempty left' @ right⟩
  ⟨A, Suc m ⊢ (q # ps, a) # filter nonempty left' @ right⟩
  using DisP(5, 7)[where left=⟨- # left'⟩] DisP(8) by fastforce+
moreover have *: ⟨p ∨ q⟩ at a in (ps, a) # filter nonempty left' @ right
  using Cons DisP(1, 8) by fastforce
moreover have
  ⟨new p a ((ps, a) # filter nonempty left' @ right)⟩
  ⟨new q a ((ps, a) # filter nonempty left' @ right)⟩
  using Cons DisP(2-3, 8) unfolding new-def by auto
ultimately have ⟨A, m ⊢ (ps, a) # filter nonempty left' @ right⟩
  using STA.DisP by fast
then have ⟨A, Suc m ⊢ filter nonempty ((ps, a) # left') @ right⟩
  using * nonempty-Suc by fast
then show ?thesis
  using Cons DisP(8) by auto
qed
next
case (DisN p q a ps branch n)
then show ?case
proof (cases left)
  case Nil
  then have ⟨A, Suc m ⊢ ((¬ q) # (¬ p) # ps, a) # branch⟩
    using DisN(4) by fastforce
  then have ⟨A, m ⊢ (ps, a) # branch⟩
    using DisN(1-2) STA.DisN by fast
  then show ?thesis
    using Nil DisN(5) STA-Suc by auto
next
case (Cons - left')
then have ⟨A, Suc m ⊢ ((¬ q) # (¬ p) # ps, a) # filter nonempty left' @
right⟩

```

```

    using DisN(4)[where left= $\langle - \# left' \rangle$ ] DisN(5) by fastforce
  moreover have *:  $\langle (\neg (p \vee q)) \text{ at } a \text{ in } (ps, a) \# \text{filter nonempty left}' @ \text{right} \rangle$ 
    using Cons DisN(1, 5) by fastforce
  moreover consider
     $\langle \text{new } (\neg p) a ((ps, a) \# \text{filter nonempty left}' @ \text{right}) \rangle \mid$ 
     $\langle \text{new } (\neg q) a ((ps, a) \# \text{filter nonempty left}' @ \text{right}) \rangle$ 
    using Cons DisN(2, 5) unfolding new-def by auto
  ultimately have  $\langle A, m \vdash (ps, a) \# \text{filter nonempty left}' @ \text{right} \rangle$ 
    using STA.DisN by metis
  then have  $\langle A, \text{Suc } m \vdash \text{filter nonempty } ((ps, a) \# \text{left}') @ \text{right} \rangle$ 
    using * nonempty-Suc by fast
  then show ?thesis
    using Cons DisN(5) by auto
qed
next
case (DiaP p a ps branch i n)
then show ?case
proof (cases left)
  case Nil
  then have  $\langle A, \text{Suc } m \vdash ((@ i p) \# (\diamond \text{Nom } i) \# ps, a) \# \text{branch} \rangle$ 
    using DiaP(6) by fastforce
  then have  $\langle A, m \vdash (ps, a) \# \text{branch} \rangle$ 
    using DiaP(1-4) STA.DiaP by fast
  then show ?thesis
    using Nil DiaP(7) STA-Suc by auto
next
case (Cons - left')
then have  $\langle A, \text{Suc } m \vdash ((@ i p) \# (\diamond \text{Nom } i) \# ps, a) \# \text{filter nonempty left}' @ \text{right} \rangle$ 
  using DiaP(6)[where left= $\langle - \# left' \rangle$ ] DiaP(7) by fastforce
  moreover have *:  $\langle (\diamond p) \text{ at } a \text{ in } (ps, a) \# \text{filter nonempty left}' @ \text{right} \rangle$ 
    using Cons DiaP(1, 7) by fastforce
  moreover have  $\langle i \notin A \cup \text{branch-nominals } ((ps, a) \# \text{filter nonempty left}' @ \text{right}) \rangle$ 
    using Cons DiaP(2, 7) unfolding branch-nominals-def by auto
  moreover have  $\langle \neg \text{witnessed } p a ((ps, a) \# \text{filter nonempty left}' @ \text{right}) \rangle$ 
    using Cons DiaP(4, 7) unfolding witnessed-def by auto
  ultimately have  $\langle A, m \vdash (ps, a) \# \text{filter nonempty left}' @ \text{right} \rangle$ 
    using DiaP(3) STA.DiaP by fast
  then have  $\langle A, \text{Suc } m \vdash \text{filter nonempty } ((ps, a) \# \text{left}') @ \text{right} \rangle$ 
    using * nonempty-Suc by fast
  then show ?thesis
    using Cons DiaP(7) by auto
qed
next
case (DiaN p a ps branch i n)
then show ?case
proof (cases left)
  case Nil

```

```

then have ⟨A, Suc m ⊢ ((¬ (@ i p)) # ps, a) # branch⟩
  using DiaN(5) by fastforce
then have ⟨A, m ⊢ (ps, a) # branch⟩
  using DiaN(1-3) STA.DiaN by fast
then show ?thesis
  using Nil DiaN(6) STA-Suc by auto
next
case (Cons - left')
then have ⟨A, Suc m ⊢ ((¬ (@ i p)) # ps, a) # filter nonempty left' @ right⟩
  using DiaN(5)[where left=⟨- # left'⟩] DiaN(6) by fastforce
moreover have *: ⟨(¬ (◇ p)) at a in (ps, a) # filter nonempty left' @ right⟩
  using Cons DiaN(1, 6) by fastforce
moreover have *: ⟨(◇ Nom i) at a in (ps, a) # filter nonempty left' @ right⟩
  using Cons DiaN(2, 6) by fastforce
moreover have ⟨new (¬ (@ i p)) a ((ps, a) # filter nonempty left' @ right)⟩
  using Cons DiaN(3, 6) unfolding new-def by auto
ultimately have ⟨A, m ⊢ (ps, a) # filter nonempty left' @ right⟩
  using STA.DiaN by fast
then have ⟨A, Suc m ⊢ filter nonempty ((ps, a) # left') @ right⟩
  using * nonempty-Suc by fast
then show ?thesis
  using Cons DiaN(6) by auto
qed
next
case (SatP a p b ps branch n)
then show ?case
proof (cases left)
  case Nil
  then have ⟨A, Suc m ⊢ (p # ps, a) # branch⟩
    using SatP(4) by fastforce
  then have ⟨A, m ⊢ (ps, a) # branch⟩
    using SatP(1-2) STA.SatP by fast
  then show ?thesis
    using Nil SatP(5) STA-Suc by auto
next
case (Cons - left')
then have ⟨A, Suc m ⊢ (p # ps, a) # filter nonempty left' @ right⟩
  using SatP(4)[where left=⟨- # left'⟩] SatP(5) by fastforce
moreover have ⟨(@ a p) at b in (ps, a) # filter nonempty left' @ right⟩
  using Cons SatP(1, 5) by fastforce
moreover have ⟨new p a ((ps, a) # filter nonempty left' @ right)⟩
  using Cons SatP(2, 5) unfolding new-def by auto
ultimately have *: ⟨A, m ⊢ (ps, a) # filter nonempty left' @ right⟩
  using STA.SatP by fast
then have ⟨A, Suc m ⊢ filter nonempty ((ps, a) # left') @ right⟩
proof (cases ps)
  case Nil
  then have ⟨a ∈ branch-nominals (filter nonempty left' @ right)⟩
    unfolding branch-nominals-def using SatP(1, 5) Cons by fastforce

```

```

    then show ?thesis
      using * Nil GoTo by fastforce
  next
    case Cons
    then show ?thesis
      using * STA-Suc by auto
  qed
  then show ?thesis
    using Cons SatP(5) by auto
  qed
next
case (SatN a p b ps branch n)
then show ?case
proof (cases left)
  case Nil
  then have ⟨A, Suc m ⊢ ((¬ p) # ps, a) # branch⟩
    using SatN(4) by fastforce
  then have ⟨A, m ⊢ (ps, a) # branch⟩
    using SatN(1-2) STA.SatN by fast
  then show ?thesis
    using Nil SatN(5) STA-Suc by auto
next
case (Cons - left')
then have ⟨A, Suc m ⊢ ((¬ p) # ps, a) # filter nonempty left' @ right⟩
  using SatN(4)[where left=⟨- # left'⟩] SatN(5) by fastforce
moreover have ⟨(¬ (@ a p)) at b in (ps, a) # filter nonempty left' @ right⟩
  using Cons SatN(1, 5) by fastforce
moreover have ⟨new (¬ p) a ((ps, a) # filter nonempty left' @ right)⟩
  using Cons SatN(2, 5) unfolding new-def by auto
ultimately have *: ⟨A, m ⊢ (ps, a) # filter nonempty left' @ right⟩
  using STA.SatN by fast
then have ⟨A, Suc m ⊢ filter nonempty ((ps, a) # left') @ right⟩
proof (cases ps)
  case Nil
  then have ⟨a ∈ branch-nominals (filter nonempty left' @ right)⟩
    unfolding branch-nominals-def using SatN(1, 5) Cons by fastforce
  then show ?thesis
    using * Nil GoTo by fastforce
next
case Cons
  then show ?thesis
    using * STA-Suc by auto
  qed
  then show ?thesis
    using Cons SatN(5) by auto
  qed
next
case (GoTo i n)
show ?case

```



```

    using GoTo(3)[where left= $\langle [], i \rangle \# left$ ] by simp
next
case (Nom p b ps a branch n)
then show ?case
proof (cases left)
  case Nil
  then have  $\langle A, Suc\ m \vdash (p \# ps, a) \# branch \rangle$ 
    using Nom(6) by fastforce
  then have  $\langle A, m \vdash (ps, a) \# branch \rangle$ 
    using Nom(1-4) STA.Nom by metis
  then show ?thesis
    using Nil Nom(7) STA-Suc by auto
next
case (Cons - left')
then have  $\langle A, Suc\ m \vdash (p \# ps, a) \# filter\ nonempty\ left' \ @\ right \rangle$ 
  using Nom(6)[where left= $\langle - \# left' \rangle$ ] Nom(7) by fastforce
moreover have
   $\langle p\ at\ b\ in\ (ps, a) \# filter\ nonempty\ left' \ @\ right \rangle$  and a:
   $\langle Nom\ a\ at\ b\ in\ (ps, a) \# filter\ nonempty\ left' \ @\ right \rangle$ 
  using Cons Nom(1-2, 7) by simp-all (metis empty-iff empty-set)+
moreover have  $\langle new\ p\ a\ ((ps, a) \# filter\ nonempty\ left' \ @\ right) \rangle$ 
  using Cons Nom(4, 7) unfolding new-def by auto
ultimately have *:  $\langle A, m \vdash (ps, a) \# filter\ nonempty\ left' \ @\ right \rangle$ 
  using Nom(3) STA.Nom by metis
then have  $\langle A, Suc\ m \vdash filter\ nonempty\ ((ps, a) \# left') \ @\ right \rangle$ 
proof (cases ps)
  case Nil
  moreover have  $\langle a \neq b \rangle$ 
    using Nom(1, 4) unfolding new-def by blast
  ultimately have  $\langle a \in branch\ nominals\ (filter\ nonempty\ left' \ @\ right) \rangle$ 
    using a unfolding branch-nominals-def by fastforce
  then show ?thesis
    using * Nil GoTo by auto
next
case Cons
  then show ?thesis
    using * STA-Suc by auto
qed
then show ?thesis
  using Cons Nom(7) by auto
qed
qed

theorem STA-potential:  $\langle A, n \vdash branch \implies A, Suc\ m \vdash branch \rangle$ 
  using STA-nonempty[where left= $\langle [] \rangle$ ] by auto

corollary STA-one:  $\langle A, n \vdash branch \implies A, 1 \vdash branch \rangle$ 
  using STA-potential by auto

```

## 5.1 Free GoTo

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

**lemma** *GoTo'*:

**assumes**  $\langle A, \text{Suc } n \vdash ([], i) \# \text{branch} \rangle \langle i \in \text{branch-nominals } \text{branch} \rangle$   
**shows**  $\langle A, \text{Suc } n \vdash \text{branch} \rangle$   
**using** *assms GoTo STA-potential by fast*

## 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**  
 $\langle [] !. v = \text{None} \rangle$   
 $| \langle (x \# xs) !. v = (\text{if } \text{length } xs = v \text{ then } \text{Some } x \text{ else } xs !. v) \rangle$

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

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

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

**lemma** *rev-nth-Suc*:  $\langle (xs @ [x]) !. \text{Suc } v = xs !. v \rangle$   
**by** (*induct xs*) *auto*

**lemma** *rev-nth-bounded*:  $\langle v < \text{length } xs \implies \exists x. xs !. v = \text{Some } x \rangle$   
**by** (*induct xs*) *simp-all*

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

**proof** (*induct xs arbitrary: v rule: rev-induct*)

**case** (*snoc a xs*)

**then show** *?case*

**proof** (*induct v*)

**case** (*Suc v*)

**then have**  $\langle xs !. v = \text{Some } y \rangle$

**using** *rev-nth-snoc by fast*

**then have**  $\langle (x \# xs) !. v = \text{Some } y \rangle$

**using** *Suc(2) by blast*

```

    then show ?case
      using Suc( $\beta$ ) by auto
    qed simp
  qed simp

```

```

lemma rev-nth-append:  $\langle xs \ !. v = \text{Some } y \implies (ys @ xs) \ !. v = \text{Some } y \rangle$ 
  using rev-nth-Cons[where  $xs = \_ @ xs$ ] by (induct ys) simp-all

```

```

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

```

```

proof
  assume  $\langle block \in. branch \rangle$ 
  then show  $\langle \exists v. branch \ !. v = \text{Some } block \rangle$ 
  proof (induct branch)
    case (Cons block' branch)
    then show ?case
    proof (cases  $\langle block = block' \rangle$ )
      case False
      then have  $\langle \exists v. branch \ !. v = \text{Some } block \rangle$ 
        using Cons by simp
      then show ?thesis
        using rev-nth-Cons by fast
    qed auto
  qed simp

```

```

next
  assume  $\langle \exists v. branch \ !. v = \text{Some } block \rangle$ 
  then show  $\langle block \in. branch \rangle$ 
  proof (induct branch)
    case (Cons block' branch)
    then show ?case
      by simp (metis option.sel)
  qed simp
qed

```

```

lemma rev-nth-on:  $\langle p \text{ on } (ps, i) \longleftrightarrow (\exists v. ps \ !. v = \text{Some } p) \vee p = \text{Nom } i \rangle$ 
  by (simp add: rev-nth-mem)

```

```

lemma rev-nth-Some:  $\langle xs \ !. v = \text{Some } y \implies v < \text{length } xs \rangle$ 

```

```

proof (induct xs arbitrary: v rule: rev-induct)
  case (snoc x xs)
  then show ?case
    by (induct v) (simp-all, metis rev-nth-snoc)
qed simp

```

```

lemma index-Cons:

```

```

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

```

$\langle (p \# ps, a) \# branch \rangle !. v = \text{Some } (p \# qs, b) \rangle$   
**using** *assms(1)* **by** *auto*  
**moreover have**  $\langle qs \rangle !. v' = \text{Some } q \rangle \langle p \# qs \rangle !. v' = \text{Some } q \rangle$   
**using** *assms(2)* *rev-nth-Cons* **by** *fast+*  
**ultimately show** *?thesis*  
**by** *fastforce*  
**qed**

## 6.2 Mapping

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

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

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

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

**lemma** *mapi-block-add-oob*:  
**assumes**  $\langle length \ ps \leq v' \rangle$   
**shows**  
 $\langle mapi\text{-block} \ (mapper \ f \ (\{(v, v')\} \cup xs)) \ v \ (ps, i) =$   
 $mapi\text{-block} \ (mapper \ f \ xs \ v) \ (ps, i) \rangle$   
**using** *assms* **by** *(induct ps) simp-all*

**lemma** *mapi-branch-add-oob*:  
**assumes**  $\langle length \ branch \leq v \rangle$   
**shows**  
 $\langle mapi\text{-branch} \ (mapper \ f \ (\{(v, v')\} \cup xs)) \ branch =$   
 $mapi\text{-branch} \ (mapper \ f \ xs) \ branch \rangle$   
**unfolding** *mapi-branch-def* **using** *assms* **by** *(induct branch) simp-all*

**lemma** *mapi-branch-head-add-oob*:  
 $\langle mapi\text{-branch} \ (mapper \ f \ (\{(length \ branch, length \ ps)\} \cup xs)) \ ((ps, a) \# branch) \rangle$   
 $=$   
 $mapi\text{-branch} \ (mapper \ f \ xs) \ ((ps, a) \# branch) \rangle$   
**using** *mapi-branch-add-oob* [**where** *branch=branch*] **unfolding** *mapi-branch-def*  
**using** *mapi-block-add-oob* [**where** *ps=ps*] **by** *simp*

```

lemma mapi-branch-mem:
  assumes  $\langle (ps, i) \in .\text{branch} \rangle$ 
  shows  $\langle \exists v. (mapi\ f\ v)\ ps, i \in .\text{mapi-branch}\ f\ \text{branch} \rangle$ 
  unfolding mapi-branch-def using assms by (induct branch) auto

lemma rev-nth-mapi-branch:
  assumes  $\langle \text{branch} \neq .\text{Some}\ (ps, a) \rangle$ 
  shows  $\langle (mapi\ f\ v)\ ps, a \in .\text{mapi-branch}\ f\ \text{branch} \rangle$ 
  unfolding mapi-branch-def using assms
  by (induct branch) (simp-all, metis mapi-block.simps option.inject)

lemma rev-nth-mapi-block:
  assumes  $\langle ps \neq .\text{Some}\ p \rangle$ 
  shows  $\langle f\ v'\ p\ \text{on}\ (mapi\ f\ ps, a) \rangle$ 
  using assms by (induct ps) (simp-all, metis option.sel)

lemma mapi-append:
   $\langle mapi\ f\ (xs\ @\ ys) = mapi\ (\lambda v. f\ (v + \text{length}\ ys))\ xs\ @\ mapi\ f\ ys \rangle$ 
  by (induct xs) simp-all

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

lemma mapi-branch-id:  $\langle mapi\ \text{branch}\ (\text{mapper}\ f\ \{\})\ \text{branch} = \text{branch} \rangle$ 
  unfolding mapi-branch-def using mapi-block-id by (induct branch) auto

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

lemma mapi-rev-nth:
  assumes  $\langle xs \neq .\text{Some}\ x \rangle$ 
  shows  $\langle mapi\ f\ xs \neq .\text{Some}\ (f\ v\ x) \rangle$ 
  using assms
proof (induct xs arbitrary: v)
  case (Cons y xs)
  have *:  $\langle mapi\ f\ (y\ \#\ xs) = f\ (\text{length}\ xs)\ y\ \#\ mapi\ f\ xs \rangle$ 
    by simp
  show ?case
  proof (cases v = length xs)
  case True
  then have  $\langle mapi\ f\ (y\ \#\ xs) \neq .\text{Some}\ (f\ (\text{length}\ xs)\ y) \rangle$ 
    using length-mapi * by (metis rev-nth.simps(2))
  then show ?thesis
    using Cons.prem True by auto
  next
  case False
  then show ?thesis
    using * Cons length-mapi by (metis rev-nth.simps(2))
qed

```

qed simp

## 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**  
 $\langle \text{all-is } p \text{ ps } xs \equiv \forall v \in xs. ps !. v = \text{Some } p \rangle$

**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**  
 $\langle \text{is-at } p \ i \ \text{branch } v \ v' \equiv \exists ps. \text{branch} !. v = \text{Some } (ps, i) \wedge ps !. v' = \text{Some } p \rangle$

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**  
 $\langle \text{is-elsewhere } p \ i \ \text{branch } xs \equiv \exists w \ w' \ ps. (w, w') \notin xs \wedge \text{branch} !. w = \text{Some } (ps, i) \wedge (p = \text{Nom } i \vee ps !. w' = \text{Some } p) \rangle$

**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**  
 $\langle \text{Dup } p \ i \ \text{branch } xs \equiv \forall (v, v') \in xs. \text{is-at } p \ i \ \text{branch } v \ v' \wedge \text{is-elsewhere } p \ i \ \text{branch } xs \rangle$

**lemma**  $\text{Dup-all-is}$ :

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

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

**using** *assms* **unfolding**  $\text{Dup-def}$   $\text{is-at-def}$   $\text{all-is-def}$   $\text{proj-def}$  **by** *auto*

**lemma**  $\text{Dup-branch}$ :

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

**unfolding**  $\text{Dup-def}$   $\text{is-at-def}$   $\text{is-elsewhere-def}$  **using**  $\text{rev-nth-append}$  **by** *fast*

**lemma**  $\text{Dup-block}$ :

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

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

**unfolding**  $\text{Dup-def}$

**proof** *safe*

**fix**  $v \ v'$

**assume**  $\langle (v, v') \in xs \rangle$

**then show**  $\langle \text{is-at } p \ i \ ((ps' @ ps, a) \# \text{branch}) \ v \ v' \rangle$

**using** *assms*  $\text{rev-nth-append}$  **unfolding**  $\text{Dup-def}$   $\text{is-at-def}$  **by** *fastforce*

**next**

**fix**  $v \ v'$

**assume**  $\langle (v, v') \in xs \rangle$

**then obtain**  $w \ w' \ qs$  **where**

$\langle (w, w') \notin xs \rangle \langle ((ps, a) \# branch) !. w = Some (qs, i) \rangle$   
 $\langle p = Nom i \vee qs !. w' = Some p \rangle$   
**using** *assms unfolding Dup-def is-elsewhere-def by blast*  
**then have**  
 $\langle \exists qs. ((ps' @ ps, a) \# branch) !. w = Some (qs, i) \wedge$   
 $(p = Nom i \vee qs !. w' = Some p) \rangle$   
**using** *rev-nth-append by fastforce*  
**then show**  $\langle is-elsewhere p i ((ps' @ ps, a) \# branch) xs \rangle$   
**unfolding** *is-elsewhere-def using*  $\langle (w, w') \notin xs \rangle$  **by** *blast*  
**qed**

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

**lemma** *Dup-touches*:  $\langle Dup p i branch xs \Longrightarrow only-touches i branch xs \rangle$   
**unfolding** *Dup-def is-at-def only-touches-def by auto*

**lemma** *only-touches-opening*:  
**assumes**  $\langle only-touches i branch xs \rangle \langle (v, v') \in xs \rangle \langle branch !. v = Some (ps, a) \rangle$   
**shows**  $\langle i = a \rangle$   
**using** *assms unfolding only-touches-def is-at-def by auto*

**lemma** *Dup-head*:  
 $\langle Dup p i ((ps, a) \# branch) xs \Longrightarrow Dup p i ((q \# ps, a) \# branch) xs \rangle$   
**using** *Dup-block[where ps'=[-]] by simp*

**lemma** *Dup-head-oob'*:  
**assumes**  $\langle Dup p i ((ps, a) \# branch) xs \rangle$   
**shows**  $\langle (length branch, k + length ps) \notin xs \rangle$   
**using** *assms rev-nth-Some unfolding Dup-def is-at-def by fastforce*

**lemma** *Dup-head-oob*:  
**assumes**  $\langle Dup p i ((ps, a) \# branch) xs \rangle$   
**shows**  $\langle (length branch, length ps) \notin xs \rangle$   
**using** *assms Dup-head-oob'[where k=0] by fastforce*

## 7.2 Omitting formulas

**primrec** *omit* ::  $\langle nat set \Rightarrow ('a, 'b) fm list \Rightarrow ('a, 'b) fm list \rangle$  **where**  
 $\langle omit xs [] = [] \rangle$   
 $| \langle omit xs (p \# ps) = (if length ps \in xs then omit xs ps else p \# omit xs ps) \rangle$

**primrec** *omit-block* ::  $\langle nat set \Rightarrow ('a, 'b) block \Rightarrow ('a, 'b) block \rangle$  **where**  
 $\langle omit-block xs (ps, a) = (omit xs ps, a) \rangle$

**definition** *omit-branch* ::  $\langle (nat \times nat) set \Rightarrow ('a, 'b) branch \Rightarrow ('a, 'b) branch \rangle$   
**where**  
 $\langle omit-branch xs branch \equiv mapi (\lambda v. omit-block (proj xs v)) branch \rangle$

**lemma** *omit-mem*:  $\langle ps \ !. \ v = \text{Some } p \implies v \notin xs \implies p \in . \text{omit } xs \ ps \rangle$   
**proof** *(induct ps)*  
    **case** *(Cons q ps)*  
    **then show** *?case*  
        **by** *(cases (v = length ps) simp-all)*  
**qed** *simp*

**lemma** *omit-id*:  $\langle \text{omit } \{ \} \ ps = ps \rangle$   
    **by** *(induct ps) auto*

**lemma** *omit-block-id*:  $\langle \text{omit-block } \{ \} \ block = block \rangle$   
    **using** *omit-id by (cases block) simp*

**lemma** *omit-branch-id*:  $\langle \text{omit-branch } \{ \} \ branch = branch \rangle$   
    **unfolding** *omit-branch-def proj-def using omit-block-id*  
    **by** *(induct branch) fastforce+*

**lemma** *omit-branch-mem-diff-opening*:  
    **assumes**  $\langle \text{only-touches } i \ branch \ xs \rangle \langle (ps, a) \in . \ branch \rangle \langle i \neq a \rangle$   
    **shows**  $\langle (ps, a) \in . \ \text{omit-branch } xs \ branch \rangle$   
**proof** *–*  
    **obtain** *v where v: (branch !. v = Some (ps, a))*  
        **using** *assms(2) rev-nth-mem by fast*  
    **then have**  $\langle \text{omit-branch } xs \ branch \ !. \ v = \text{Some } (\text{omit } (\text{proj } xs \ v) \ ps, \ a) \rangle$   
        **unfolding** *omit-branch-def by (simp add: mapi-rev-nth)*  
    **then have**  $\ast: \langle (\text{omit } (\text{proj } xs \ v) \ ps, \ a) \in . \ \text{omit-branch } xs \ branch \rangle$   
        **using** *rev-nth-mem by fast*  
    **moreover have**  $\langle \text{proj } xs \ v = \{ \} \rangle$   
        **unfolding** *proj-def using assms(1, 3) v only-touches-opening by fast*  
    **then have**  $\langle \text{omit } (\text{proj } xs \ v) \ ps = ps \rangle$   
        **using** *omit-id by auto*  
    **ultimately show** *?thesis*  
        **by** *simp*  
**qed**

**lemma** *Dup-omit-branch-mem-same-opening*:  
    **assumes**  $\langle \text{Dup } p \ i \ branch \ xs \rangle \langle p \ \text{at } i \ \text{in } branch \rangle$   
    **shows**  $\langle p \ \text{at } i \ \text{in } \text{omit-branch } xs \ branch \rangle$   
**proof** *–*  
    **obtain** *ps where ps: (ps, i) ∈ . branch (p on (ps, i))*  
        **using** *assms(2) by blast*  
    **then obtain** *v where v: (branch !. v = Some (ps, i))*  
        **using** *rev-nth-mem by fast*  
    **then have**  $\langle \text{omit-branch } xs \ branch \ !. \ v = \text{Some } (\text{omit } (\text{proj } xs \ v) \ ps, \ i) \rangle$   
        **unfolding** *omit-branch-def by (simp add: mapi-rev-nth)*  
    **then have**  $\ast: \langle (\text{omit } (\text{proj } xs \ v) \ ps, \ i) \in . \ \text{omit-branch } xs \ branch \rangle$   
        **using** *rev-nth-mem by fast*



**consider**  
 $v'$  **where**  $\langle ps \ !. v' = \text{Some } p \rangle \langle (v, v') \in xs \rangle \mid$   
 $v'$  **where**  $\langle ps \ !. v' = \text{Some } p \rangle \langle (v, v') \notin xs \rangle \mid$   
 $\langle p = \text{Nom } i \rangle$   
**using**  $ps \ v \ \text{rev-nth-mem}$  **by**  $\text{fastforce}$   
**then show**  $?thesis$   
**proof**  $\text{cases}$   
**case**  $(1 \ v')$   
**then obtain**  $qs \ w \ w'$  **where**  $qs$ :  
 $\langle (w, w') \notin xs \rangle \langle \text{branch} \ !. w = \text{Some } (qs, i) \rangle \langle p = \text{Nom } i \vee qs \ !. w' = \text{Some } p \rangle$   
**using**  $\text{assms}(1)$  **unfolding**  $\text{Dup-def is-elsewhere-def}$  **by**  $\text{blast}$   
**then have**  $\langle \text{omit-branch } xs \ \text{branch} \ !. w = \text{Some } (\text{omit } (\text{proj } xs \ w) \ qs, i) \rangle$   
**unfolding**  $\text{omit-branch-def}$  **by**  $(\text{simp add: mapi-rev-nth})$   
**then have**  $\langle (\text{omit } (\text{proj } xs \ w) \ qs, i) \in. \text{omit-branch } xs \ \text{branch} \rangle$   
**using**  $\text{rev-nth-mem}$  **by**  $\text{fast}$   
**moreover have**  $\langle p \ \text{on } (\text{omit } (\text{proj } xs \ w) \ qs, i) \rangle$   
**unfolding**  $\text{proj-def}$  **using**  $qs(1, 3)$   $\text{omit-mem}$  **by**  $\text{fastforce}$   
**ultimately show**  $?thesis$   
**by**  $\text{blast}$   
**next**  
**case**  $(2 \ v')$   
**then show**  $?thesis$   
**using**  $*$   $\text{omit-mem}$  **unfolding**  $\text{proj-def}$   
**by**  $(\text{metis Image-singleton-iff on.simps})$   
**next**  
**case**  $3$   
**then show**  $?thesis$   
**using**  $*$  **by**  $\text{auto}$   
**qed**  
**qed**

**lemma**  $\text{omit-del}$ :  
**assumes**  $\langle p \in. ps \rangle \langle p \notin \text{set } (\text{omit } xs \ ps) \rangle$   
**shows**  $\langle \exists v. ps \ !. v = \text{Some } p \wedge v \in xs \rangle$   
**using**  $\text{assms}$   $\text{omit-mem}$   $\text{rev-nth-mem}$  **by**  $\text{metis}$

**lemma**  $\text{omit-all-is}$ :  
**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**  $\langle q = p \rangle$   
**using**  $\text{assms}$   $\text{omit-del}$  **unfolding**  $\text{all-is-def}$  **by**  $\text{fastforce}$

**definition**  $\text{all-is-branch} :: \langle ('a, 'b) \text{ fm} \Rightarrow 'b \Rightarrow ('a, 'b) \text{ branch} \Rightarrow (\text{nat} \times \text{nat}) \text{ set} \Rightarrow \text{bool} \rangle$  **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**  $\text{all-is-branch}$ :  
 $\langle \text{all-is-branch } p \ i \ \text{branch } xs \Longrightarrow \text{branch} \ !. v = \text{Some } (ps, a) \Longrightarrow \text{all-is } p \ ps \ (\text{proj } xs \ v) \rangle$

**unfolding** *all-is-branch-def is-at-def all-is-def proj-def* **using** *rev-nth-Some* **by** *fastforce*

**lemma** *Dup-all-is-branch*:  $\langle \text{Dup } p \text{ } i \text{ } \text{branch } xs \implies \text{all-is-branch } p \text{ } i \text{ } \text{branch } xs \rangle$   
**unfolding** *all-is-branch-def Dup-def* **by** *fast*

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

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

**proof** –

**obtain** *ps* **where** *ps*:  $\langle (ps, i) \in . \text{branch} \rangle \langle q \text{ on } (ps, i) \rangle$   
**using** *assms(2)* **by** *blast*  
**then obtain** *v* **where** *v*:  $\langle \text{branch } !. v = \text{Some } (ps, i) \rangle$   
**using** *rev-nth-mem* **by** *fast*  
**then have**  $\langle \text{omit-branch } xs \text{ } \text{branch } !. v = \text{Some } (\text{omit } (\text{proj } xs \text{ } v) \text{ } ps, i) \rangle$   
**unfolding** *omit-branch-def* **by** *(simp add: mapi-rev-nth)*  
**then have**  $\ast$ :  $\langle (\text{omit } (\text{proj } xs \text{ } v) \text{ } ps, i) \in . \text{omit-branch } xs \text{ } \text{branch} \rangle$   
**using** *rev-nth-mem* **by** *fast*  
**moreover have**  $\langle \text{all-is } p \text{ } ps \text{ } (\text{proj } xs \text{ } v) \rangle$   
**using** *assms(1)* *v* *all-is-branch* **by** *fast*  
**then have**  $\langle q \text{ on } (\text{omit } (\text{proj } xs \text{ } v) \text{ } ps, i) \rangle$   
**using** *ps* *assms(3)* *omit-all-is* **by** *auto*  
**ultimately show** *?thesis*  
**by** *blast*

**qed**

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

**assumes**  $\langle \text{Dup } p \text{ } i \text{ } \text{branch } xs \rangle \langle q \text{ at } a \text{ in } \text{branch} \rangle$   
**shows**  $\langle q \text{ at } a \text{ in } \text{omit-branch } xs \text{ } \text{branch} \rangle$   
**using** *assms* *omit-branch-mem-diff-opening Dup-touches Dup-omit-branch-mem-same-opening*  
*omit-branch-mem-diff-formula Dup-all-is-branch* **by** *fast*

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

**lemma** *on-omit*:  $\langle p \text{ on } (\text{omit } xs \text{ } ps, i) \implies p \text{ on } (ps, i) \rangle$   
**using** *omit-set* **by** *auto*

**lemma** *all-is-set*:

**assumes**  $\langle \text{all-is } p \text{ } ps \text{ } xs \rangle$   
**shows**  $\langle \{p\} \cup \text{set } (\text{omit } xs \text{ } ps) = \{p\} \cup \text{set } ps \rangle$   
**using** *assms* *omit-all-is* *omit-set* **unfolding** *all-is-def* **by** *fast*

**lemma** *all-is-list-nominals*:

**assumes**  $\langle \text{all-is } p \text{ } ps \text{ } xs \rangle$   
**shows**  $\langle \text{nominals } p \cup \text{list-nominals } (\text{omit } xs \text{ } ps) = \text{nominals } p \cup \text{list-nominals } ps \rangle$   
**using** *assms* *all-is-set* **by** *fastforce*

```

lemma all-is-block-nominals:
  assumes  $\langle \text{all-is } p \text{ ps } xs \rangle$ 
  shows  $\langle \text{nominals } p \cup \text{block-nominals } (\text{omit } xs \text{ ps}, i) = \text{nominals } p \cup \text{block-nominals } (ps, i) \rangle$ 
  using assms by (simp add: all-is-list-nominals)

lemma all-is-branch-nominals':
  assumes  $\langle \text{all-is-branch } p \text{ i branch } xs \rangle$ 
  shows
     $\langle \text{nominals } p \cup \text{branch-nominals } (\text{omit-branch } xs \text{ branch}) = \text{nominals } p \cup \text{branch-nominals } \text{branch} \rangle$ 
proof –
  have  $\langle \forall (v, v') \in xs. v < \text{length } \text{branch} \longrightarrow \text{is-at } p \text{ i branch } v \ v' \rangle$ 
    using assms unfolding all-is-branch-def is-at-def by auto
  then show ?thesis
proof (induct branch)
  case Nil
  then show ?case
    unfolding omit-branch-def by simp
next
case (Cons block branch)
then show ?case
proof (cases block)
  case (Pair ps a)
  have  $\langle \forall (v, v') \in xs. v < \text{length } \text{branch} \longrightarrow \text{is-at } p \text{ i branch } v \ v' \rangle$ 
    using Cons(2) rev-nth-Cons unfolding is-at-def by auto
  then have
     $\langle \text{nominals } p \cup \text{branch-nominals } (\text{omit-branch } xs \text{ branch}) = \text{nominals } p \cup \text{branch-nominals } \text{branch} \rangle$ 
    using Cons(1) by blast
  then have
     $\langle \text{nominals } p \cup \text{branch-nominals } (\text{omit-branch } xs \ ((ps, a) \# \text{branch})) = \text{nominals } p \cup \text{block-nominals } (\text{omit } (\text{proj } xs \ (\text{length } \text{branch})) \text{ ps}, a) \cup \text{branch-nominals } \text{branch} \rangle$ 
    unfolding branch-nominals-def omit-branch-def by auto
  moreover have  $\langle \text{all-is } p \text{ ps } (\text{proj } xs \ (\text{length } \text{branch})) \rangle$ 
    using Cons(2) Pair unfolding proj-def all-is-def is-at-def by auto
  then have
     $\langle \text{nominals } p \cup \text{block-nominals } (\text{omit } (\text{proj } xs \ (\text{length } \text{branch})) \text{ ps}, a) = \text{nominals } p \cup \text{block-nominals } (ps, a) \rangle$ 
    using all-is-block-nominals by fast
  then have
     $\langle \text{nominals } p \cup \text{block-nominals } (\text{omit-block } (\text{proj } xs \ (\text{length } \text{branch})) (ps, a)) =$ 
    =
     $\text{nominals } p \cup \text{block-nominals } (ps, a) \rangle$ 
    by simp
  ultimately have
     $\langle \text{nominals } p \cup \text{branch-nominals } (\text{omit-branch } xs \ ((ps, a) \# \text{branch})) = \text{nominals } p \cup \text{block-nominals } (ps, a) \cup \text{branch-nominals } \text{branch} \rangle$ 

```

```

    by auto
  then show ?thesis
    unfolding branch-nominals-def using Pair by auto
qed
qed
qed

```

**lemma** *Dup-branch-nominals*:

```

  assumes ⟨Dup p i branch xs⟩
  shows ⟨branch-nominals (omit-branch xs branch) = branch-nominals branch⟩
proof (cases xs = {})
  case True
  then show ?thesis
    using omit-branch-id by metis
next
  case False
  with assms obtain ps w w' where
    ⟨(w, w') ∉ xs⟩ ⟨branch !. w = Some (ps, i)⟩ ⟨p = Nom i ∨ ps !. w' = Some p⟩
    unfolding Dup-def is-elsewhere-def by fast
  then have *: ⟨(ps, i) ∈. branch⟩ ⟨p on (ps, i)⟩
    using rev-nth-mem rev-nth-on by fast+
  then have ⟨nominals p ⊆ branch-nominals branch⟩
    unfolding branch-nominals-def using block-nominals by fast
  moreover obtain ps' where
    ⟨(ps', i) ∈. omit-branch xs branch⟩ ⟨p on (ps', i)⟩
    using assms * Dup-omit-branch-mem by fast
  then have ⟨nominals p ⊆ branch-nominals (omit-branch xs branch)⟩
    unfolding branch-nominals-def using block-nominals by fast
  moreover have
    ⟨nominals p ∪ branch-nominals (omit-branch xs branch) =
      nominals p ∪ branch-nominals branch⟩
    using assms all-is-branch-nominals' Dup-all-is-branch by fast
  ultimately show ?thesis
    by blast
qed

```

**lemma** *omit-branch-mem-dual*:

```

  assumes ⟨p at i in omit-branch xs branch⟩
  shows ⟨p at i in branch⟩
proof –
  obtain ps where ps: ⟨(ps, i) ∈. omit-branch xs branch⟩ ⟨p on (ps, i)⟩
    using assms(1) by blast
  then obtain v where v: ⟨omit-branch xs branch !. v = Some (ps, i)⟩
    using rev-nth-mem unfolding omit-branch-def by fast
  then have ⟨v < length (omit-branch xs branch)⟩
    using rev-nth-Some by fast
  then have ⟨v < length branch⟩
    unfolding omit-branch-def using length-mapi by metis
  then obtain ps' i' where ps': ⟨branch !. v = Some (ps', i')⟩

```

**using** *rev-nth-bounded* **by** (*metis surj-pair*)  
**then have**  $\langle \text{omit-branch } xs \text{ branch } !. v = \text{Some } (\text{omit } (\text{proj } xs \ v) \ ps', i') \rangle$   
**unfolding** *omit-branch-def* **by** (*simp add: mapi-rev-nth*)  
**then have**  $\langle ps = \text{omit } (\text{proj } xs \ v) \ ps' \rangle \langle i = i' \rangle$   
**using** *v* **by** *simp-all*  
**then have**  $\langle p \text{ on } (ps', i) \rangle$   
**using** *ps omit-set* **by** *auto*  
**moreover have**  $\langle (ps', i) \in. \text{branch} \rangle$   
**using** *ps' i = i' rev-nth-mem* **by** *fast*  
**ultimately show** *?thesis*  
**using**  $\langle ps = \text{omit } (\text{proj } xs \ v) \ ps' \rangle$  **by** *blast*  
**qed**

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

**obtain** *ps qs i* **where**  
 $ps: \langle (ps, a) \in. \text{omit-branch } xs \ \text{branch} \rangle \langle (@ \ i \ p) \ \text{on } (ps, a) \rangle$  **and**  
 $qs: \langle (qs, a) \in. \text{omit-branch } xs \ \text{branch} \rangle \langle (\diamond \ \text{Nom } i) \ \text{on } (qs, a) \rangle$   
**using** *assms unfolding witnessed-def* **by** *blast*  
**from** *ps* **obtain** *ps'* **where**  
 $\langle (ps', a) \in. \text{branch} \rangle \langle (@ \ i \ p) \ \text{on } (ps', a) \rangle$   
**using** *omit-branch-mem-dual* **by** *fast*  
**moreover from** *qs* **obtain** *qs'* **where**  
 $\langle (qs', a) \in. \text{branch} \rangle \langle (\diamond \ \text{Nom } i) \ \text{on } (qs', a) \rangle$   
**using** *omit-branch-mem-dual* **by** *fast*  
**ultimately show** *?thesis*  
**unfolding** *witnessed-def* **by** *blast*  
**qed**

**lemma** *new-omit-branch*:  
**assumes**  $\langle \text{new } p \ a \ \text{branch} \rangle$   
**shows**  $\langle \text{new } p \ a \ (\text{omit-branch } xs \ \text{branch}) \rangle$   
**using** *assms omit-branch-mem-dual* **unfolding** *new-def* **by** *fast*

**lemma** *omit-oob*:  
**assumes**  $\langle \text{length } ps \leq v \rangle$   
**shows**  $\langle \text{omit } (\{v\} \cup xs) \ ps = \text{omit } xs \ ps \rangle$   
**using** *assms* **by** (*induct ps*) *simp-all*

**lemma** *omit-branch-oob*:  
**assumes**  $\langle \text{length } \text{branch} \leq v \rangle$   
**shows**  $\langle \text{omit-branch } (\{(v, v')\} \cup xs) \ \text{branch} = \text{omit-branch } xs \ \text{branch} \rangle$   
**using** *assms*  
**proof** (*induct branch*)  
**case** *Nil*  
**then show** *?case*  
**unfolding** *omit-branch-def* **by** *simp*

```

next
case (Cons block branch)
let ?xs = ⟨{v, v'} ∪ xs⟩
show ?case
proof (cases block)
  case (Pair ps a)
  then have
    ⟨omit-branch ?xs ((ps, a) # branch) =
      (omit (proj ?xs (length branch)) ps, a) # omit-branch xs branch⟩
  using Cons unfolding omit-branch-def by simp
  moreover have ⟨proj ?xs (length branch) = proj xs (length branch)⟩
  using Cons(2) unfolding proj-def by auto
  ultimately show ?thesis
  unfolding omit-branch-def by simp
qed
qed

```

### 7.3 Induction

```

lemma STA-Dup:
  assumes ⟨A, n ⊢ branch⟩ ⟨Dup q i branch xs⟩
  shows ⟨A, n ⊢ omit-branch xs branch⟩
  using assms
proof (induct n branch)
  case (Close p i' branch n)
  have ⟨p at i' in omit-branch xs branch⟩
  using Close(1, 3) Dup-omit-branch-mem by fast
  moreover have ⟨¬ p at i' in omit-branch xs branch⟩
  using Close(2, 3) Dup-omit-branch-mem by fast
  ultimately show ?case
  using STA.Close by fast
next
case (Neg p a ps branch n)
have ⟨A, Suc n ⊢ omit-branch xs ((p # ps, a) # branch)⟩
  using Neg(4 -) Dup-head by fast
moreover have ⟨(length branch, length ps) ∉ xs⟩
  using Neg(5) Dup-head-oob by fast
ultimately have
  ⟨A, Suc n ⊢ (p # omit (proj xs (length branch)) ps, a) # omit-branch xs branch⟩
  unfolding omit-branch-def proj-def by simp
moreover have ⟨¬ ¬ p at a in omit-branch xs ((ps, a) # branch)⟩
  using Neg(1, 5) Dup-omit-branch-mem by fast
moreover have ⟨new p a (omit-branch xs ((ps, a) # branch))⟩
  using Neg(2) new-omit-branch by fast
ultimately show ?case
  by (simp add: omit-branch-def STA.Neg)
next
case (DisP p q a ps branch n)
have

```

$\langle A, \text{Suc } n \vdash \text{omit-branch } xs \ ((p \# ps, a) \# \text{branch}) \rangle$   
 $\langle A, \text{Suc } n \vdash \text{omit-branch } xs \ ((q \# ps, a) \# \text{branch}) \rangle$   
**using**  $\text{DisP}(4-)$  *Dup-head* **by** *fast+*  
**moreover have**  $\langle \text{length } \text{branch}, \text{length } ps \notin xs \rangle$   
**using**  $\text{DisP}(8)$  *Dup-head-oob* **by** *fast*  
**ultimately have**  
 $\langle A, \text{Suc } n \vdash (p \# \text{omit } (\text{proj } xs \ (\text{length } \text{branch})) \ ps, a) \# \text{omit-branch } xs \ \text{branch} \rangle$   
 $\langle A, \text{Suc } n \vdash (q \# \text{omit } (\text{proj } xs \ (\text{length } \text{branch})) \ ps, a) \# \text{omit-branch } xs \ \text{branch} \rangle$   
**unfolding** *omit-branch-def proj-def* **by** *simp-all*  
**moreover have**  $\langle (p \vee q) \text{ at } a \text{ in } \text{omit-branch } xs \ ((ps, a) \# \text{branch}) \rangle$   
**using**  $\text{DisP}(1, 8)$  *Dup-omit-branch-mem* **by** *fast*  
**moreover have**  $\langle \text{new } p \ a \ (\text{omit-branch } xs \ ((ps, a) \# \text{branch})) \rangle$   
**using**  $\text{DisP}(2)$  *new-omit-branch* **by** *fast*  
**moreover have**  $\langle \text{new } q \ a \ (\text{omit-branch } xs \ ((ps, a) \# \text{branch})) \rangle$   
**using**  $\text{DisP}(3)$  *new-omit-branch* **by** *fast*  
**ultimately show** *?case*  
**by** (*simp add: omit-branch-def STA.DisP*)

**next**

**case** ( $\text{DisN } p \ q \ a \ ps \ \text{branch } n$ )  
**have**  $\langle A, \text{Suc } n \vdash \text{omit-branch } xs \ (((\neg q) \# (\neg p) \# ps, a) \# \text{branch}) \rangle$   
**using**  $\text{DisN}(4-)$  *Dup-block* [**where**  $ps' = \langle [-, -] \rangle$ ] **by** *fastforce*  
**moreover have**  $\langle \text{length } \text{branch}, \text{length } ps \notin xs \rangle$   
**using**  $\text{DisN}(5)$  *Dup-head-oob* **by** *fast*  
**moreover have**  $\langle \text{length } \text{branch}, 1 + \text{length } ps \notin xs \rangle$   
**using**  $\text{DisN}(5)$  *Dup-head-oob'* **by** *fast*  
**ultimately have**  
 $\langle A, \text{Suc } n \vdash ((\neg q) \# (\neg p) \# \text{omit } (\text{proj } xs \ (\text{length } \text{branch})) \ ps, a) \# \text{omit-branch } xs \ \text{branch} \rangle$   
**unfolding** *omit-branch-def proj-def* **by** *simp*  
**moreover have**  $\langle (\neg (p \vee q)) \text{ at } a \text{ in } \text{omit-branch } xs \ ((ps, a) \# \text{branch}) \rangle$   
**using**  $\text{DisN}(1, 5)$  *Dup-omit-branch-mem* **by** *fast*  
**moreover have**  
 $\langle \text{new } (\neg p) \ a \ (\text{omit-branch } xs \ ((ps, a) \# \text{branch})) \vee$   
 $\text{new } (\neg q) \ a \ (\text{omit-branch } xs \ ((ps, a) \# \text{branch})) \rangle$   
**using**  $\text{DisN}(2)$  *new-omit-branch* **by** *fast*  
**ultimately show** *?case*  
**by** (*simp add: omit-branch-def STA.DisN*)

**next**

**case** ( $\text{DiaP } p \ a \ ps \ \text{branch } i \ n$ )  
**have**  $\langle A, \text{Suc } n \vdash \text{omit-branch } xs \ (((@ i p) \# (\diamond \text{Nom } i) \# ps, a) \# \text{branch}) \rangle$   
**using**  $\text{DiaP}(4-)$  *Dup-block* [**where**  $ps' = \langle [-, -] \rangle$ ] **by** *fastforce*  
**moreover have**  $\langle \text{length } \text{branch}, \text{length } ps \notin xs \rangle$   
**using**  $\text{DiaP}(7)$  *Dup-head-oob* **by** *fast*  
**moreover have**  $\langle \text{length } \text{branch}, 1 + \text{length } ps \notin xs \rangle$   
**using**  $\text{DiaP}(7)$  *Dup-head-oob'* **by** *fast*  
**ultimately have**  
 $\langle A, \text{Suc } n \vdash ((@ i p) \# (\diamond \text{Nom } i) \# \text{omit } (\text{proj } xs \ (\text{length } \text{branch})) \ ps, a) \# \text{omit-branch } xs \ \text{branch} \rangle$   
**unfolding** *omit-branch-def proj-def* **by** *simp*

**moreover have**  $\langle \diamond p \text{ at } a \text{ in omit-branch } xs \ ((ps, a) \# \text{branch}) \rangle$   
**using** *DiaP(1, 7) Dup-omit-branch-mem by fast*  
**moreover have**  $\langle i \notin A \cup \text{branch-nominals (omit-branch } xs \ ((ps, a) \# \text{branch})) \rangle$   
**using** *DiaP(2, 7) Dup-branch-nominals by fast*  
**moreover have**  $\langle \neg \text{witnessed } p \ a \text{ (omit-branch } xs \ ((ps, a) \# \text{branch})) \rangle$   
**using** *DiaP(4) witnessed-omit-branch by fast*  
**ultimately show** *?case*  
**using** *DiaP(3) by (simp add: omit-branch-def STA.DiaP)*  
**next**  
**case** *(DiaN p a ps branch i n)*  
**have**  $\langle A, Suc \ n \vdash \text{omit-branch } xs \ (((\neg (@ \ i \ p)) \# \ ps, a) \# \text{branch}) \rangle$   
**using** *DiaN(4-) Dup-head by fast*  
**moreover have**  $\langle \text{length branch, length ps} \notin xs \rangle$   
**using** *DiaN(6) Dup-head-oob by fast*  
**ultimately have**  
 $\langle A, Suc \ n \vdash ((\neg (@ \ i \ p)) \# \text{omit (proj } xs \ (\text{length branch})) \ ps, a) \# \text{omit-branch } xs \ \text{branch} \rangle$   
**unfolding** *omit-branch-def proj-def by simp*  
**moreover have**  $\langle \neg (\diamond p) \text{ at } a \text{ in omit-branch } xs \ ((ps, a) \# \text{branch}) \rangle$   
**using** *DiaN(1, 6) Dup-omit-branch-mem by fast*  
**moreover have**  $\langle \diamond \text{Nom } i \text{ at } a \text{ in omit-branch } xs \ ((ps, a) \# \text{branch}) \rangle$   
**using** *DiaN(2, 6) Dup-omit-branch-mem by fast*  
**moreover have**  $\langle \text{new } (\neg (@ \ i \ p)) \ a \text{ (omit-branch } xs \ ((ps, a) \# \text{branch})) \rangle$   
**using** *DiaN(3) new-omit-branch by fast*  
**ultimately show** *?case*  
**by** *(simp add: omit-branch-def STA.DiaN)*  
**next**  
**case** *(SatP a p b ps branch n)*  
**have**  $\langle A, Suc \ n \vdash \text{omit-branch } xs \ ((p \# \ ps, a) \# \text{branch}) \rangle$   
**using** *SatP(4-) Dup-head by fast*  
**moreover have**  $\langle \text{length branch, length ps} \notin xs \rangle$   
**using** *SatP(5) Dup-head-oob by fast*  
**ultimately have**  
 $\langle A, Suc \ n \vdash (p \# \text{omit (proj } xs \ (\text{length branch})) \ ps, a) \# \text{omit-branch } xs \ \text{branch} \rangle$   
**unfolding** *omit-branch-def proj-def by simp*  
**moreover have**  $\langle (@ \ a \ p) \text{ at } b \text{ in omit-branch } xs \ ((ps, a) \# \text{branch}) \rangle$   
**using** *SatP(1, 5) Dup-omit-branch-mem by fast*  
**moreover have**  $\langle \text{new } p \ a \text{ (omit-branch } xs \ ((ps, a) \# \text{branch})) \rangle$   
**using** *SatP(2) new-omit-branch by fast*  
**ultimately show** *?case*  
**by** *(simp add: omit-branch-def STA.SatP)*  
**next**  
**case** *(SatN a p b ps branch n)*  
**have**  $\langle A, Suc \ n \vdash \text{omit-branch } xs \ (((\neg p) \# \ ps, a) \# \text{branch}) \rangle$   
**using** *SatN(4-) Dup-head by fast*  
**moreover have**  $\langle \text{length branch, length ps} \notin xs \rangle$   
**using** *SatN(5) Dup-head-oob by fast*  
**ultimately have**  
 $\langle A, Suc \ n \vdash ((\neg p) \# \text{omit (proj } xs \ (\text{length branch})) \ ps, a) \# \text{omit-branch } xs \rangle$



*branch*  
**unfolding** *omit-branch-def proj-def* **by** *simp*  
**moreover have**  $\langle \neg (@ a p) \rangle$  *at b in omit-branch xs ((ps, a) # branch)*  
**using** *SatN(1, 5) Dup-omit-branch-mem* **by** *fast*  
**moreover have**  $\langle \text{new } (\neg p) a \rangle$  *(omit-branch xs ((ps, a) # branch))*  
**using** *SatN(2) new-omit-branch* **by** *fast*  
**ultimately show** *?case*  
**by** *(simp add: omit-branch-def STA.SatN)*

**next**  
**case** *(GoTo i branch n)*  
**then have**  $\langle A, n \vdash \text{omit-branch xs } ([], i) \# \text{branch} \rangle$   
**using** *Dup-branch[where extra= $\langle [([], i)] \rangle$ ]* **by** *fastforce*  
**then have**  $\langle A, n \vdash ([], i) \# \text{omit-branch xs branch} \rangle$   
**unfolding** *omit-branch-def* **by** *simp*  
**moreover have**  $\langle i \in \text{branch-nominals } (\text{omit-branch xs branch}) \rangle$   
**using** *GoTo(1, 4) Dup-branch-nominals* **by** *fast*  
**ultimately show** *?case*  
**unfolding** *omit-branch-def* **by** *(simp add: STA.GoTo)*

**next**  
**case** *(Nom p b ps a branch n)*  
**have**  $\langle A, \text{Suc } n \vdash \text{omit-branch xs } (p \# ps, a) \# \text{branch} \rangle$   
**using** *Nom(4-) Dup-head* **by** *fast*  
**moreover have**  $\langle \text{length branch, length ps} \notin xs \rangle$   
**using** *Nom(7) Dup-head-oob* **by** *fast*  
**ultimately have**  
 $\langle A, \text{Suc } n \vdash (p \# \text{omit } (\text{proj xs } (\text{length branch})) ps, a) \# \text{omit-branch xs branch} \rangle$   
**unfolding** *omit-branch-def proj-def* **by** *simp*  
**moreover have**  $\langle p \text{ at } b \text{ in } \text{omit-branch xs } ((ps, a) \# \text{branch}) \rangle$   
**using** *Nom(1, 7) Dup-omit-branch-mem* **by** *fast*  
**moreover have**  $\langle \text{Nom } a \text{ at } b \text{ in } \text{omit-branch xs } ((ps, a) \# \text{branch}) \rangle$   
**using** *Nom(2, 7) Dup-omit-branch-mem* **by** *fast*  
**moreover have**  $\langle \text{new } p a \rangle$  *(omit-branch xs ((ps, a) # branch))*  
**using** *Nom(4) new-omit-branch* **by** *fast*  
**ultimately show** *?case*  
**using** *Nom(3)* **by** *(simp add: omit-branch-def STA.Nom)*

**qed**

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

**proof** –  
**obtain** *qs* **where** *qs*:  
 $\langle (qs, a) \in. (ps, a) \# \text{branch} \rangle$   $\langle p \text{ on } (qs, a) \rangle$   
**using** *assms(2) unfolding new-def* **by** *blast*

**let**  $?xs = \langle \{(\text{length branch}, \text{length ps})\} \rangle$

**have**  $*$ :  $\langle \text{is-at } p a \rangle$  *((p # ps, a) # branch) (length branch) (length ps)*  
**unfolding** *is-at-def* **by** *simp*

```

have ⟨Dup p a ((p # ps, a) # branch) ?xs⟩
proof (cases ⟨p = Nom a⟩)
  case True
  moreover have ⟨((p # ps, a) # branch) !. length branch = Some (p # ps, a)⟩
    by simp
  moreover have ⟨p on (p # ps, a)⟩
    by simp
  ultimately have ⟨is-elsewhere p a ((p # ps, a) # branch) ?xs⟩
    unfolding is-elsewhere-def using assms(2) rev-nth-Some
    by (metis (mono-tags, lifting) Pair-inject less-le singletonD)
  then show ?thesis
    unfolding Dup-def using * by blast
next
case false: False
then show ?thesis
proof (cases ⟨ps = qs⟩)
  case True
  then obtain w' where w': ⟨qs !. w' = Some p⟩
    using qs(2) false rev-nth-mem by fastforce
  then have ⟨(p # ps) !. w' = Some p⟩
    using True rev-nth-Cons by fast
  moreover have ⟨((p # ps, a) # branch) !. length branch = Some (p # ps,
a)⟩
    by simp
  moreover have ⟨(length branch, w') ∉ ?xs⟩
    using True w' rev-nth-Some by fast
  ultimately have ⟨is-elsewhere p a ((p # ps, a) # branch) ?xs⟩
    unfolding is-elsewhere-def by fast
  then show ?thesis
    unfolding Dup-def using * by fast
next
case False
then obtain w where w: ⟨branch !. w = Some (qs, a)⟩
  using qs(1) rev-nth-mem by fastforce
  moreover obtain w' where w': ⟨qs !. w' = Some p⟩
  using qs(2) false rev-nth-mem by fastforce
  moreover have ⟨(w, w') ∉ ?xs⟩
  using rev-nth-Some w by fast
  ultimately have ⟨is-elsewhere p a ((p # ps, a) # branch) ?xs⟩
  unfolding is-elsewhere-def using rev-nth-Cons by fast
  then show ?thesis
  unfolding Dup-def using * by fast
qed
qed

then have ⟨A, n ⊢ omit-branch ?xs ((p # ps, a) # branch)⟩
  using assms(1) STA-Dup by fast
then have ⟨A, n ⊢ (omit (proj ?xs (length branch)) ps, a) # omit-branch ?xs⟩

```

*branch*  
**unfolding** *omit-branch-def proj-def* **by** *simp*  
**moreover have**  $\langle \text{omit-branch } ?xs \text{ branch} = \text{omit-branch } \{\} \text{ branch} \rangle$   
**using** *omit-branch-oob* **by** *auto*  
**then have**  $\langle \text{omit-branch } ?xs \text{ branch} = \text{branch} \rangle$   
**using** *omit-branch-id* **by** *simp*  
**moreover have**  $\langle \text{proj } ?xs \text{ (length branch)} = \{\text{length ps}\} \rangle$   
**unfolding** *proj-def* **by** *blast*  
**then have**  $\langle \text{omit (proj } ?xs \text{ (length branch)) ps} = \text{omit } \{\} \text{ ps} \rangle$   
**using** *omit-oob* **by** *auto*  
**then have**  $\langle \text{omit (proj } ?xs \text{ (length branch)) ps} = \text{ps} \rangle$   
**using** *omit-id* **by** *simp*  
**ultimately show** *?thesis*  
**by** *simp*  
**qed**

## 7.4 Unrestricted rules

**lemma** *STA-add*:  $\langle A, n \vdash \text{branch} \implies A, m + n \vdash \text{branch} \rangle$   
**using** *STA-Suc* **by** (*induct m*) *auto*

**lemma** *STA-le*:  $\langle A, n \vdash \text{branch} \implies n \leq m \implies A, m \vdash \text{branch} \rangle$   
**using** *STA-add* **by** (*metis le-add-diff-inverse2*)

**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**  $\langle A, n \vdash (ps, a) \# \text{branch} \rangle$   
**using** *assms Neg Dup STA-Suc* **by** *metis*

**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**  $\langle A, n \vdash (ps, a) \# \text{branch} \rangle$   
**proof** (*cases*  $\langle \text{new } p \text{ a } ((ps, a) \# \text{branch}) \wedge \text{new } q \text{ a } ((ps, a) \# \text{branch}) \rangle$ )  
**case** *True*  
**moreover have**  $\langle A, \text{Suc } n \vdash (p \# ps, a) \# \text{branch} \rangle \langle A, \text{Suc } n \vdash (q \# ps, a) \# \text{branch} \rangle$   
**using** *assms(2-3) STA-Suc* **by** *fast+*  
**ultimately show** *?thesis*  
**using** *assms(1) DisP* **by** *fast*  
**next**  
**case** *False*  
**then show** *?thesis*  
**using** *assms Dup* **by** *fast*  
**qed**

**lemma** *DisP''*:  
**assumes**  
 ⟨⟨ $p \vee q$ ⟩ at  $a$  in  $(ps, a)$  # branch⟩  
 ⟨ $A, n \vdash (p \# ps, a)$  # branch⟩ ⟨ $A, m \vdash (q \# ps, a)$  # branch⟩  
**shows** ⟨ $A, \max n m \vdash (ps, a)$  # branch⟩  
**proof** (cases ⟨ $n \leq m$ ⟩)  
**case** *True*  
**then have** ⟨ $A, m \vdash (p \# ps, a)$  # branch⟩  
**using** *assms(2) STA-le by blast*  
**then show** *?thesis*  
**using** *assms True by (simp add: DisP' max.absorb2)*  
**next**  
**case** *False*  
**then have** ⟨ $A, n \vdash (q \# ps, a)$  # branch⟩  
**using** *assms(3) STA-le by fastforce*  
**then show** *?thesis*  
**using** *assms False by (simp add: DisP' max.absorb1)*  
**qed**

**lemma** *DisN'*:  
**assumes**  
 ⟨⟨ $\neg (p \vee q)$ ⟩ at  $a$  in  $(ps, a)$  # branch⟩  
 ⟨ $A, n \vdash ((\neg q) \# (\neg p) \# ps, a)$  # branch⟩  
**shows** ⟨ $A, n \vdash (ps, a)$  # branch⟩  
**proof** (cases ⟨ $\text{new } (\neg q) a ((ps, a) \# \text{branch}) \vee \text{new } (\neg p) a ((ps, a) \# \text{branch})$ ⟩)  
**case** *True*  
**then show** *?thesis*  
**using** *assms DisN STA-Suc by fast*  
**next**  
**case** *False*  
**then show** *?thesis*  
**using** *assms Dup*  
**by** (*metis (no-types, lifting) list.set-intros(1-2) new-def on.simps set-ConsD*)  
**qed**

**lemma** *DiaP'*:  
**assumes**  
 ⟨⟨ $\diamond p$ ⟩ at  $a$  in  $(ps, a)$  # branch⟩  
 ⟨ $i \notin A \cup \text{branch-nominals } ((ps, a) \# \text{branch})$ ⟩  
 ⟨ $\nexists a. p = \text{Nom } a$ ⟩  
 ⟨ $\neg \text{witnessed } p a ((ps, a) \# \text{branch})$ ⟩  
 ⟨ $A, n \vdash ((@ i p) \# (\diamond \text{Nom } i) \# ps, a)$  # branch⟩  
**shows** ⟨ $A, n \vdash (ps, a)$  # branch⟩  
**using** *assms DiaP STA-Suc by fast*

**lemma** *DiaN'*:  
**assumes**  
 ⟨⟨ $\neg (\diamond p)$ ⟩ at  $a$  in  $(ps, a)$  # branch⟩  
 ⟨⟨ $\diamond \text{Nom } i$ ⟩ at  $a$  in  $(ps, a)$  # branch⟩

$\langle A, n \vdash ((\neg (@ i p)) \# ps, a) \# branch \rangle$   
**shows**  $\langle A, n \vdash (ps, a) \# branch \rangle$   
**using** *assms DiaN Dup STA-Suc* **by fast**

**lemma** *SatP'*:

**assumes**  
 $\langle (@ a p) \text{ at } b \text{ in } (ps, a) \# branch \rangle$   
 $\langle A, n \vdash (p \# ps, a) \# branch \rangle$   
**shows**  $\langle A, n \vdash (ps, a) \# branch \rangle$   
**using** *assms SatP Dup STA-Suc* **by fast**

**lemma** *SatN'*:

**assumes**  
 $\langle (\neg (@ a p)) \text{ at } b \text{ in } (ps, a) \# branch \rangle$   
 $\langle A, n \vdash ((\neg p) \# ps, a) \# branch \rangle$   
**shows**  $\langle A, n \vdash (ps, a) \# branch \rangle$   
**using** *assms SatN Dup STA-Suc* **by fast**

**lemma** *Nom'*:

**assumes**  
 $\langle p \text{ at } b \text{ in } (ps, a) \# branch \rangle$   
 $\langle Nom a \text{ at } b \text{ in } (ps, a) \# branch \rangle$   
 $\langle \forall i. p = Nom i \vee p = (\diamond Nom i) \longrightarrow i \in A \rangle$   
 $\langle A, n \vdash (p \# ps, a) \# branch \rangle$   
**shows**  $\langle A, n \vdash (ps, a) \# branch \rangle$   
**proof** (*cases*  $\langle new p a ((ps, a) \# branch) \rangle$ )  
**case** *True*  
**moreover have**  $\langle A, Suc n \vdash (p \# ps, a) \# branch \rangle$   
**using** *assms(4) STA-Suc* **by blast**  
**ultimately show** *?thesis*  
**using** *assms(1-3) Nom* **by metis**  
**next**  
**case** *False*  
**then show** *?thesis*  
**using** *assms Dup* **by fast**  
**qed**

## 8 Substitution

**lemma** *finite-nominals*:  $\langle finite (nominals p) \rangle$   
**by** (*induct p*) *simp-all*

**lemma** *finite-block-nominals*:  $\langle finite (block-nominals block) \rangle$   
**using** *finite-nominals* **by** (*induct block*) *auto*

**lemma** *finite-branch-nominals*:  $\langle finite (branch-nominals branch) \rangle$   
**unfolding** *branch-nominals-def* **by** (*induct branch*) (*auto simp: finite-block-nominals*)

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

$\langle \text{sub-list } f \text{ } ps \equiv \text{map } (\text{sub } f) \text{ } ps \rangle$

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

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

**lemma** *sub-block-mem*:  $\langle p \text{ on } \text{block} \Longrightarrow \text{sub } f \text{ } p \text{ on } \text{sub-block } f \text{ } \text{block} \rangle$   
**by** (*induct block*) *auto*

**lemma** *sub-branch-mem*:  
**assumes**  $\langle (ps, i) \in . \text{branch} \rangle$   
**shows**  $\langle (\text{sub-list } f \text{ } ps, f \text{ } i) \in . \text{sub-branch } f \text{ } \text{branch} \rangle$   
**unfolding** *sub-branch-def* **using** *assms image-iff* **by** *fastforce*

**lemma** *sub-block-nominals*:  $\langle \text{block-nominals } (\text{sub-block } f \text{ } \text{block}) = f \text{ } \langle \text{block-nominals } \text{block} \rangle$   
**by** (*induct block*) (*auto simp: sub-nominals*)

**lemma** *sub-branch-nominals*:  
 $\langle \text{branch-nominals } (\text{sub-branch } f \text{ } \text{branch}) = f \text{ } \langle \text{branch-nominals } \text{branch} \rangle$   
**unfolding** *branch-nominals-def sub-branch-def*  
**by** (*induct branch*) (*auto simp: sub-block-nominals*)

**lemma** *sub-list-id*:  $\langle \text{sub-list } \text{id } ps = ps \rangle$   
**using** *sub-id* **by** (*induct ps*) *auto*

**lemma** *sub-block-id*:  $\langle \text{sub-block } \text{id } \text{block} = \text{block} \rangle$   
**using** *sub-list-id* **by** (*induct block*) *auto*

**lemma** *sub-branch-id*:  $\langle \text{sub-branch } \text{id } \text{branch} = \text{branch} \rangle$   
**unfolding** *sub-branch-def* **using** *sub-block-id* **by** (*induct branch*) *auto*

**lemma** *sub-block-upd-fresh*:  
**assumes**  $\langle i \notin \text{block-nominals } \text{block} \rangle$   
**shows**  $\langle \text{sub-block } (f(i := j)) \text{ } \text{block} = \text{sub-block } f \text{ } \text{block} \rangle$   
**using** *assms* **by** (*induct block*) (*auto simp add: sub-upd-fresh*)

**lemma** *sub-branch-upd-fresh*:  
**assumes**  $\langle i \notin \text{branch-nominals } \text{branch} \rangle$   
**shows**  $\langle \text{sub-branch } (f(i := j)) \text{ } \text{branch} = \text{sub-branch } f \text{ } \text{branch} \rangle$   
**using** *assms* **unfolding** *branch-nominals-def sub-branch-def*  
**by** (*induct branch*) (*auto simp: sub-block-upd-fresh*)

**lemma** *sub-comp*:  $\langle \text{sub } f \text{ } (\text{sub } g \text{ } p) = \text{sub } (f \text{ } o \text{ } g) \text{ } p \rangle$   
**by** (*induct p*) *simp-all*

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

```

using sub-comp by (induct ps) auto

lemma sub-block-comp:  $\langle \text{sub-block } f \ (\text{sub-block } g \ \text{block}) = \text{sub-block } (f \ o \ g) \ \text{block} \rangle$ 
using sub-list-comp by (induct block) simp-all

lemma sub-branch-comp:
 $\langle \text{sub-branch } f \ (\text{sub-branch } g \ \text{branch}) = \text{sub-branch } (f \ o \ g) \ \text{branch} \rangle$ 
unfolding sub-branch-def using sub-block-comp by (induct branch) fastforce+

lemma swap-id:  $\langle (\text{id}(i := j, j := i)) \ o \ (\text{id}(i := j, j := i)) = \text{id} \rangle$ 
by auto

lemma at-in-sub-branch:
assumes  $\langle p \ \text{at } i \ \text{in } (ps, a) \ \# \ \text{branch} \rangle$ 
shows  $\langle \text{sub } f \ p \ \text{at } f \ i \ \text{in } (\text{sub-list } f \ ps, f \ a) \ \# \ \text{sub-branch } f \ \text{branch} \rangle$ 
using assms sub-branch-mem by fastforce

lemma sub-still-allowed:
assumes  $\langle \forall i. p = \text{Nom } i \vee p = (\diamond \ \text{Nom } i) \longrightarrow i \in A \rangle$ 
shows  $\langle \text{sub } f \ p = \text{Nom } i \vee \text{sub } f \ p = (\diamond \ \text{Nom } i) \longrightarrow i \in f \ ' \ A \rangle$ 
proof safe
assume  $\langle \text{sub } f \ p = \text{Nom } i \rangle$ 
then obtain  $i'$  where  $i': p = \text{Nom } i' \ \langle f \ i' = i \rangle$ 
by (cases p) simp-all
then have  $\langle i' \in A \rangle$ 
using assms by fast
then show  $\langle i \in f \ ' \ A \rangle$ 
using  $i'$  by fast
next
assume  $\langle \text{sub } f \ p = (\diamond \ \text{Nom } i) \rangle$ 
then obtain  $i'$  where  $i': p = (\diamond \ \text{Nom } i') \ \langle f \ i' = i \rangle$ 
proof (induct p)
case (Dia q)
then show ?case
by (cases q) simp-all
qed simp-all
then have  $\langle i' \in A \rangle$ 
using assms by fast
then show  $\langle i \in f \ ' \ A \rangle$ 
using  $i'$  by fast
qed

```

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  $\langle \bigwedge (f :: 'b \Rightarrow 'c) \ i \ A. \ \text{finite } A \implies i \notin A \implies \exists j. j \notin f \ 'A \rangle$ 
   $\langle \text{finite } A \rangle \langle A, n \vdash \text{branch} \rangle$ 
  shows  $\langle f \ 'A \vdash \text{sub-branch } f \ \text{branch} \rangle$ 
  using assms(3-)
proof (induct n branch arbitrary: f rule: STA.induct)
  case (Close p i branch n)
  have  $\langle \text{sub } f \ p \ \text{at } f \ i \ \text{in } \text{sub-branch } f \ \text{branch} \rangle$ 
  using Close(1) sub-branch-mem by fastforce
  moreover have  $\langle \neg \text{sub } f \ p \rangle \ \text{at } f \ i \ \text{in } \text{sub-branch } f \ \text{branch} \rangle$ 
  using Close(2) sub-branch-mem by force
  ultimately show ?case
  using STA.Close by fast
next
  case (Neg p a ps branch n f)
  then have  $\langle f \ 'A \vdash (\text{sub } f \ p \ \# \ \text{sub-list } f \ ps, f \ a) \ \# \ \text{sub-branch } f \ \text{branch} \rangle$ 
  unfolding sub-branch-def by simp
  moreover have  $\langle \neg \neg \text{sub } f \ p \rangle \ \text{at } f \ a \ \text{in } (\text{sub-list } f \ ps, f \ a) \ \# \ \text{sub-branch } f \ \text{branch} \rangle$ 
  using Neg(1) at-in-sub-branch by (metis (no-types, hide-lams) sub.simps(3))
  ultimately have  $\langle f \ 'A \vdash (\text{sub-list } f \ ps, f \ a) \ \# \ \text{sub-branch } f \ \text{branch} \rangle$ 
  using Neg' by fast
  then show ?case
  unfolding sub-branch-def by simp
next
  case (DisP p q a ps branch n)
  then have
   $\langle f \ 'A \vdash (\text{sub } f \ p \ \# \ \text{sub-list } f \ ps, f \ a) \ \# \ \text{sub-branch } f \ \text{branch} \rangle$ 
   $\langle f \ 'A \vdash (\text{sub } f \ q \ \# \ \text{sub-list } f \ ps, f \ a) \ \# \ \text{sub-branch } f \ \text{branch} \rangle$ 
  unfolding sub-branch-def by simp-all
  moreover have  $\langle (\text{sub } f \ p \ \vee \ \text{sub } f \ q) \ \text{at } f \ a \ \text{in } (\text{sub-list } f \ ps, f \ a) \ \# \ \text{sub-branch } f \ \text{branch} \rangle$ 
  using DisP(1) at-in-sub-branch by (metis (no-types, hide-lams) sub.simps(4))
  ultimately have  $\langle f \ 'A \vdash (\text{sub-list } f \ ps, f \ a) \ \# \ \text{sub-branch } f \ \text{branch} \rangle$ 
  using DisP'' by fast
  then show ?case
  unfolding sub-branch-def by simp
next
  case (DisN p q a ps branch n)
  then have  $\langle f \ 'A \vdash ((\neg \text{sub } f \ q) \ \# \ (\neg \text{sub } f \ p) \ \# \ \text{sub-list } f \ ps, f \ a) \ \# \ \text{sub-branch } f \ \text{branch} \rangle$ 
  unfolding sub-branch-def by simp
  moreover have  $\langle \neg (\text{sub } f \ p \ \vee \ \text{sub } f \ q) \rangle \ \text{at } f \ a \ \text{in } (\text{sub-list } f \ ps, f \ a) \ \# \ \text{sub-branch } f \ \text{branch} \rangle$ 
  using DisN(1) at-in-sub-branch by (metis (no-types, hide-lams) sub.simps(3-4))
  ultimately have  $\langle f \ 'A \vdash (\text{sub-list } f \ ps, f \ a) \ \# \ \text{sub-branch } f \ \text{branch} \rangle$ 
  using DisN' by fast
  then show ?case
  unfolding sub-branch-def by simp

```



```

next
  case (DiaP p a ps branch i n)
  have ⟨i ∉ A⟩
  using DiaP(2) by simp

  show ?case
  proof (cases ⟨witnessed (sub f p) (f a) (sub-branch f ((ps, a) # branch))⟩)
    case True
    then obtain i' where
      rs: ⟨(@ i' (sub f p)) at f a in (sub-list f ps, f a) # sub-branch f branch⟩ and
      ts: ⟨(◇ Nom i') at f a in (sub-list f ps, f a) # sub-branch f branch⟩
    unfolding sub-branch-def witnessed-def by auto
    from rs have rs':
      ⟨(@ i' (sub f p)) at f a in ((◇ Nom i') # sub-list f ps, f a) # sub-branch f
      branch⟩
    by fastforce

    let ?f = ⟨f(i := i')⟩
    let ?branch = ⟨sub-branch ?f branch⟩
    have ⟨sub-branch ?f ((ps, a) # branch) = sub-branch f ((ps, a) # branch)⟩
    using DiaP(2) sub-branch-upd-fresh by fast
    then have **: ⟨sub-list ?f ps = sub-list f ps⟩ ⟨?f a = f a⟩ ⟨?branch = sub-branch
    f branch⟩
    unfolding sub-branch-def by simp-all

    have p: ⟨sub ?f p = sub f p⟩
    using DiaP(1-2) sub-upd-fresh unfolding branch-nominals-def by fastforce

    have ⟨?f ' A ⊢ sub-branch ?f (((@ i p) # (◇ Nom i) # ps, a) # branch)⟩
    using DiaP(6) by blast
    then have
      ⟨?f ' A ⊢ ((@ (?f i) (sub ?f p)) # (◇ Nom (?f i)) # sub-list ?f ps, ?f a) #
      ?branch⟩
    unfolding sub-branch-def by fastforce
    then have
      ⟨?f ' A ⊢ ((@ i' (sub f p)) # (◇ Nom i') # sub-list f ps, f a) # sub-branch f
      branch⟩
    using p ** by simp
    then have ⟨?f ' A ⊢ ((◇ Nom i') # sub-list f ps, f a) # sub-branch f branch⟩
    using rs' by (meson Dup new-def)
    then have ⟨?f ' A ⊢ (sub-list f ps, f a) # sub-branch f branch⟩
    using ts by (meson Dup new-def)
    moreover have ⟨?f ' A = f ' A⟩
    using ⟨i ∉ A⟩ by auto
    ultimately show ?thesis
    unfolding sub-branch-def by auto
  next
  case False
  have ⟨finite (branch-nominals ((ps, a) # branch))⟩

```

```

    by (simp add: finite-branch-nominals)
  then have ⟨finite (A ∪ branch-nominals ((ps, a) # branch))⟩
    using ⟨finite A⟩ by simp
  then obtain j where *: ⟨j ∉ f ‘ (A ∪ branch-nominals ((ps, a) # branch))⟩
    using DiaP(2) assms by metis
  then have ⟨j ∉ f ‘ A⟩
    by blast

  let ?f = ⟨f(i := j)⟩
  let ?branch = ⟨sub-branch ?f branch⟩
  have **: ⟨sub-branch ?f ((ps, a) # branch) = sub-branch f ((ps, a) # branch)⟩
    using DiaP(2) sub-branch-upd-fresh by fast
  then have ***: ⟨sub-list ?f ps = sub-list f ps⟩ ⟨?f a = f a⟩ ⟨?branch = sub-branch
f branch⟩
    unfolding sub-branch-def by simp-all
  moreover have p: ⟨sub ?f p = sub f p⟩
    using DiaP(1-2) sub-upd-fresh unfolding branch-nominals-def by fastforce
  ultimately have ⟨¬ witnessed (sub ?f p) (?f a) (sub-branch ?f ((ps, a) #
branch))⟩
    using False ** by simp
  then have w: ⟨¬ witnessed (sub ?f p) (?f a) ((sub-list ?f ps, ?f a) # ?branch)⟩
    unfolding sub-branch-def by simp

  have f: ⟨?f ‘ A = f ‘ A⟩
    using ⟨i ∉ A⟩ by auto

  have ⟨?f ‘ A ⊢ sub-branch ?f (((@ i p) # (◇ Nom i) # ps, a) # branch)⟩
    using DiaP(6) by blast
  then have ⟨f ‘ A ⊢ ((@ (?f i) (sub ?f p)) # (◇ Nom (?f i)) # sub-list ?f ps,
?f a) # ?branch⟩
    unfolding sub-branch-def using f by simp
  moreover have ⟨sub ?f (◇ p) at ?f a in (sub-list ?f ps, ?f a) # sub-branch ?f
branch⟩
    using DiaP(1) at-in-sub-branch by fast
  then have ⟨(◇ sub ?f p) at ?f a in (sub-list ?f ps, ?f a) # sub-branch ?f branch⟩
    by simp
  moreover have ⟨∃ a. sub ?f p = Nom a⟩
    using DiaP(3) by (cases p) simp-all
  moreover have ⟨j ∉ f ‘ (branch-nominals ((ps, a) # branch))⟩
    using * by blast
  then have ⟨?f i ∉ branch-nominals ((sub-list ?f ps, ?f a) # ?branch)⟩
    using ** sub-branch-nominals unfolding sub-branch-def
    by (metis fun-upd-same list.simps(9) sub-block.simps)
  ultimately have ⟨f ‘ A ⊢ (sub-list ?f ps, ?f a) # ?branch⟩
    using w DiaP' ⟨j ∉ f ‘ A⟩ by (metis Un-iff fun-upd-same)
  then show ?thesis
    using *** unfolding sub-branch-def by simp
qed
next

```

```

    case (DiaN p a ps branch i n)
    then have ⟨f ‘ A ⊢ ((¬ (@ (f i) (sub f p))) # sub-list f ps, f a) # sub-branch f
branch⟩
      unfolding sub-branch-def by simp
    moreover have ⟨(¬ (◇ sub f p)) at f a in (sub-list f ps, f a) # sub-branch f
branch⟩
      using DiaN(1) at-in-sub-branch by (metis (no-types, hide-lams) sub.simps(3,
5))
    moreover have ⟨(◇ Nom (f i)) at f a in (sub-list f ps, f a) # sub-branch f branch⟩
      using DiaN(2) at-in-sub-branch by (metis (no-types, hide-lams) sub.simps(2,
5))
    ultimately have ⟨f ‘ A ⊢ (sub-list f ps, f a) # sub-branch f branch⟩
      using DiaN' by fast
    then show ?case
      unfolding sub-branch-def by simp
next
    case (SatP a p b ps branch n)
    then have ⟨f ‘ A ⊢ (sub f p # sub-list f ps, f a) # sub-branch f branch⟩
      unfolding sub-branch-def by simp
    moreover have ⟨(@ (f a) (sub f p)) at f b in (sub-list f ps, f a) # sub-branch f
branch⟩
      using SatP(1) at-in-sub-branch by (metis (no-types, hide-lams) sub.simps(6))
    ultimately have ⟨f ‘ A ⊢ (sub-list f ps, f a) # sub-branch f branch⟩
      using SatP' by fast
    then show ?case
      unfolding sub-branch-def by simp
next
    case (SatN a p b ps branch n)
    then have ⟨f ‘ A ⊢ ((¬ sub f p) # sub-list f ps, f a) # sub-branch f branch⟩
      unfolding sub-branch-def by simp
    moreover have ⟨(¬ (@ (f a) (sub f p))) at f b in (sub-list f ps, f a) # sub-branch
f branch⟩
      using SatN(1) at-in-sub-branch by (metis (no-types, hide-lams) sub.simps(3,
6))
    ultimately have ⟨f ‘ A ⊢ (sub-list f ps, f a) # sub-branch f branch⟩
      using SatN' by fast
    then show ?case
      unfolding sub-branch-def by simp
next
    case (GoTo i branch n)
    then have ⟨f ‘ A ⊢ ([], f i) # sub-branch f branch⟩
      unfolding sub-branch-def by simp
    moreover have ⟨f i ∈ branch-nominals (sub-branch f branch)⟩
      using GoTo(1) sub-branch-nominals by fast
    ultimately show ?case
      using STA.GoTo by fast
next
    case (Nom p b ps a branch n)
    then have ⟨f ‘ A ⊢ sub-branch f ((p # ps, a) # branch)⟩

```

by *blast*  
 then have  $\langle f ' A \vdash (sub\ f\ p \# sub\ list\ f\ ps, f\ a) \# sub\ branch\ f\ branch \rangle$   
 unfolding *sub-branch-def* by *simp*  
 moreover have  $\langle sub\ f\ p\ at\ f\ b\ in\ (sub\ list\ f\ ps, f\ a) \# sub\ branch\ f\ branch \rangle$   
 using *Nom(1)* at-in-sub-branch by *fast*  
 moreover have  $\langle Nom\ (f\ a)\ at\ f\ b\ in\ (sub\ list\ f\ ps, f\ a) \# sub\ branch\ f\ branch \rangle$   
 using *Nom(2)* at-in-sub-branch by (*metis* (*no-types*, *hide-lams*) *sub.simps(2)*)  
 moreover have  $\langle \forall i. sub\ f\ p = Nom\ i \vee sub\ f\ p = (\Diamond\ Nom\ i) \longrightarrow i \in f ' A \rangle$   
 using *Nom(3)* *sub-still-allowed* by *metis*  
 ultimately have  $\langle f ' A \vdash (sub\ list\ f\ ps, f\ a) \# sub\ branch\ f\ branch \rangle$   
 using *Nom'* by *metis*  
 then show *?case*  
 unfolding *sub-branch-def* by *simp*  
 qed

lemma *ex-fresh-gt*:  
 fixes  $f :: \langle 'b \Rightarrow 'c \rangle$   
 assumes  $\langle \exists g :: 'c \Rightarrow 'b. surj\ g \rangle \langle finite\ A \rangle \langle i \notin A \rangle$   
 shows  $\langle \exists j. j \notin f ' A \rangle$   
 proof (rule *ccontr*)  
 assume  $\langle \#j. j \notin f ' A \rangle$   
 moreover obtain  $g :: \langle 'c \Rightarrow 'b \rangle$  where  $\langle surj\ g \rangle$   
 using *assms(1)* by *blast*  
 ultimately show *False*  
 using *assms(2-3)*  
 by (*metis UNIV-I UNIV-eq-I card-image-le card-seteq finite-imageI image-comp subsetI*)  
 qed

corollary *STA-sub-gt*:  
 fixes  $f :: \langle 'b \Rightarrow 'c \rangle$   
 assumes  $\langle \exists g :: 'c \Rightarrow 'b. surj\ g \rangle \langle A \vdash branch \rangle$   
 $\langle finite\ A \rangle \langle \forall i \in branch\ nominals\ branch. f\ i \in f ' A \longrightarrow i \in A \rangle$   
 shows  $\langle f ' A \vdash sub\ branch\ f\ branch \rangle$   
 using *assms ex-fresh-gt STA-sub'* by *metis*

corollary *STA-sub-inf*:  
 fixes  $f :: \langle 'b \Rightarrow 'c \rangle$   
 assumes  $\langle infinite\ (UNIV :: 'c\ set) \rangle \langle A \vdash branch \rangle$   
 $\langle finite\ A \rangle \langle \forall i \in branch\ nominals\ branch. f\ i \in f ' A \longrightarrow i \in A \rangle$   
 shows  $\langle f ' A \vdash sub\ branch\ f\ branch \rangle$   
 proof –  
 have  $\langle finite\ A \implies \exists j. j \notin f ' A \rangle$  for  $A$  and  $f :: \langle 'b \Rightarrow 'c \rangle$   
 using *assms(1)* *ex-new-if-finite* by *blast*  
 then show *?thesis*  
 using *assms(2-)* *STA-sub'* by *metis*  
 qed

corollary *STA-sub*:

**fixes**  $f :: \langle 'b \Rightarrow 'b \rangle$   
**assumes**  $\langle A \vdash \text{branch} \rangle \langle \text{finite } A \rangle$   
**shows**  $\langle f ' A \vdash \text{sub-branch } f \text{ branch} \rangle$   
**proof** –  
**have**  $\langle \text{finite } A \implies i \notin A \implies \exists j. j \notin f ' A \rangle$  **for**  $i \in A$  **and**  $f :: \langle 'b \Rightarrow 'b \rangle$   
**by**  $(\text{metis card-image-le card-seteq finite-imageI subsetI})$   
**then show**  $?thesis$   
**using**  $\text{assms STA-sub'}$  **by**  $\text{metis}$   
**qed**

## 8.1 Unrestricted ( $\diamond$ ) rule

**lemma**  $\text{DiaP''}$ :

**assumes**  
 $\langle \diamond p \rangle$  **at**  $a$  **in**  $(ps, a) \# \text{branch}$   
 $\langle i \notin A \cup \text{branch-nominals } ((ps, a) \# \text{branch}) \rangle \langle \nexists a. p = \text{Nom } a \rangle$   
 $\langle \text{finite } A \rangle$   
 $\langle A \vdash ((@ i p) \# (\diamond \text{Nom } i) \# ps, a) \# \text{branch} \rangle$   
**shows**  $\langle A \vdash (ps, a) \# \text{branch} \rangle$   
**proof**  $(\text{cases } \langle \text{witnessed } p \ a \ ((ps, a) \# \text{branch}) \rangle)$   
**case**  $\text{True}$   
**then obtain**  $i'$  **where**  
 $rs: \langle (@ i' p) \rangle$  **at**  $a$  **in**  $(ps, a) \# \text{branch}$  **and**  
 $ts: \langle \diamond \text{Nom } i' \rangle$  **at**  $a$  **in**  $(ps, a) \# \text{branch}$   
**unfolding**  $\text{witnessed-def}$  **by**  $\text{blast}$   
**then have**  $rs'$ :  
 $\langle (@ i' p) \rangle$  **at**  $a$  **in**  $((\diamond \text{Nom } i') \# ps, a) \# \text{branch}$   
**by**  $\text{fastforce}$   
**let**  $?f = \langle \text{id}(i := i') \rangle$   
**have**  $\langle ?f ' A \vdash \text{sub-branch } ?f (((@ i p) \# (\diamond \text{Nom } i) \# ps, a) \# \text{branch}) \rangle$   
**using**  $\text{assms}(4-5)$   $\text{STA-sub}$  **by**  $\text{blast}$   
**then have**  $\langle ?f ' A \vdash ((@ i' (sub ?f p)) \# (\diamond \text{Nom } i') \# \text{sub-list } ?f \ ps, ?f \ a) \# \text{sub-branch } ?f \ \text{branch} \rangle$   
**unfolding**  $\text{sub-branch-def}$  **by**  $\text{simp}$   
**moreover have**  $\langle i \notin \text{nominals } p \rangle \langle i \notin \text{list-nominals } ps \rangle \langle i \neq a \rangle \langle i \notin \text{branch-nominals } \text{branch} \rangle$   
**using**  $\text{assms}(1-3)$  **unfolding**  $\text{branch-nominals-def}$  **by**  $\text{fastforce+}$   
**then have**  $\langle \text{sub } ?f \ p = p \rangle$   
**by**  $(\text{simp add: sub-id sub-upd-fresh})$   
**moreover have**  $\langle \text{sub-list } ?f \ ps = ps \rangle$   
**using**  $\langle i \notin \text{list-nominals } ps \rangle$  **by**  $(\text{simp add: map-idI sub-id sub-upd-fresh})$   
**moreover have**  $\langle ?f \ a = a \rangle$   
**using**  $\langle i \neq a \rangle$  **by**  $\text{simp}$   
**moreover have**  $\langle \text{sub-branch } ?f \ \text{branch} = \text{branch} \rangle$   
**using**  $\langle i \notin \text{branch-nominals } \text{branch} \rangle$  **by**  $(\text{simp add: sub-branch-id sub-branch-upd-fresh})$   
**ultimately have**  $\langle ?f ' A \vdash ((@ i' p) \# (\diamond \text{Nom } i') \# ps, a) \# \text{branch} \rangle$   
**by**  $\text{simp}$

```

then have ⟨?f ‘ A ⊢ ((◇ Nom i′) # ps, a) # branch⟩
  using rs′ by (meson Dup new-def)
then have ⟨?f ‘ A ⊢ (ps, a) # branch⟩
  using ts by (meson Dup new-def)
moreover have ⟨?f ‘ A = A⟩
  using assms(2) by auto
ultimately show ?thesis
  by simp
next
case False
then show ?thesis
  using assms DiaP′ STA-Suc by fast
qed

```

## 9 Structural Properties

**lemma** *block-nominals-branch*:  
**assumes** ⟨block ∈. branch⟩  
**shows** ⟨block-nominals block ⊆ branch-nominals branch⟩  
**unfolding** branch-nominals-def **using** assms **by** blast

**lemma** *sub-block-fresh*:  
**assumes** ⟨i ∉ branch-nominals branch⟩ ⟨block ∈. branch⟩  
**shows** ⟨sub-block (f(i := j)) block = sub-block f block⟩  
**using** assms block-nominals-branch sub-block-upd-fresh **by** fast

**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

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** ⟨set ps ⊆ set qs⟩ ⟨(qs, a) ∈. branch⟩ ⟨A, n ⊢ (ps @ ps′, a) # branch⟩  
**shows** ⟨A, n ⊢ (ps′, a) # branch⟩  
**proof** –  
**have** ⟨∀ p ∈ set ps. p on (qs, a)⟩  
**using** assms(1) **by** auto  
**then show** ?thesis  
**proof** (induct ps′ rule: list-down-induct)  
 case Start  
**then show** ?case  
**using** assms(3) .  
 next  
 case (Cons p ps)

```

    then show ?case
      using assms(2) by (meson Dup new-def list.set-intros(2))
  qed
qed

```

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

**lemma** *STA-drop-block*:

```

  assumes
    ⟨set ps ⊆ set ps'⟩ ⟨(ps', a) ∈. branch⟩
    ⟨A, n ⊢ (ps, a) # branch⟩
  shows ⟨A, Suc n ⊢ branch⟩
  using assms
proof (induct branch)
  case Nil
  then show ?case
    by simp
next
  case (Cons block branch)
  then show ?case
  proof (cases block)
  case (Pair qs b)
  then have ⟨A, n ⊢ ([], a) # (qs, b) # branch⟩
    using Cons(2-4) STA-drop-prefix[where branch=⟨(qs, b) # branch⟩] by
simp
  moreover have ⟨a ∈ branch-nominals ((qs, b) # branch)⟩
    unfolding branch-nominals-def using Cons(3) Pair by fastforce
  ultimately have ⟨A, Suc n ⊢ (qs, b) # branch⟩
    by (simp add: GoTo)
  then show ?thesis
    using Pair Dup by fast
  qed
qed

```

**lemma** *STA-drop-block'*:

```

  assumes ⟨A, n ⊢ (ps, a) # branch⟩ ⟨(ps, a) ∈. branch⟩
  shows ⟨A, Suc n ⊢ branch⟩
  using assms STA-drop-block by fastforce

```

**lemma** *sub-branch-image*: ⟨set (sub-branch f branch) = sub-block f ' set branch⟩

```

  unfolding sub-branch-def by simp

```

**lemma** *sub-block-repl*:

```

  assumes ⟨j ∉ block-nominals block⟩
  shows ⟨i ∉ block-nominals (sub-block (id(i := j, j := i)) block)⟩
  using assms by (simp add: image-iff sub-block-nominals)

```

**lemma** *sub-branch-repl*:

```

  assumes ⟨j ∉ branch-nominals branch⟩
  shows ⟨i ∉ branch-nominals (sub-branch (id(i := j, j := i)) branch)⟩

```

using *assms* by (*simp add: image-iff sub-branch-nominals*)

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

**lemma** *STA-struct*:

**fixes** *branch* ::  $\langle 'a, 'b \rangle$  *branch*

**assumes**

*inf*:  $\langle$ infinite (UNIV :: 'b set) $\rangle$  **and** *fin*:  $\langle$ finite A $\rangle$  **and**  
 $\langle A, n \vdash$  *branch* $\rangle$   $\langle$ set *branch*  $\subseteq$  set *branch* $\rangle$

**shows**  $\langle A \vdash$  *branch* $\rangle$

**using** *assms*( $\exists$ -)

**proof** (*induct n branch arbitrary: branch' rule: STA.induct*)

**case** (*Close p i branch n*)

**then show** ?*case*

**using** *STA.Close* **by** *fast*

**next**

**case** (*Neg p a ps branch n*)

**have**  $\langle A \vdash$  (*p* # *ps*, *a*) # *branch* $\rangle$

**using** *Neg*(4-) **by** (*simp add: subset-code*(1))

**moreover have**  $\langle$ ( $\neg$   $\neg$  *p*) at *a* in (*ps*, *a*) # *branch* $\rangle$

**using** *Neg*(1, 5) **by** *auto*

**ultimately have**  $\langle A \vdash$  (*ps*, *a*) # *branch* $\rangle$

**using** *Neg'* **by** *fast*

**moreover have**  $\langle$ (*ps*, *a*)  $\in$ . *branch* $\rangle$

**using** *Neg*(5) **by** *simp*

**ultimately show** ?*case*

**using** *STA-drop-block'* **by** *fast*

**next**

**case** (*DisP p q a ps branch n*)

**have**  $\langle A \vdash$  (*p* # *ps*, *a*) # *branch* $\rangle$   $\langle A \vdash$  (*q* # *ps*, *a*) # *branch* $\rangle$

**using** *DisP*(5, 7-) **by** (*simp-all add: subset-code*(1))

**moreover have**  $\langle$ (*p*  $\vee$  *q*) at *a* in (*ps*, *a*) # *branch* $\rangle$

**using** *DisP*(1, 8) **by** *auto*

**ultimately have**  $\langle A \vdash$  (*ps*, *a*) # *branch* $\rangle$

**using** *DisP''* **by** *fast*

**moreover have**  $\langle$ (*ps*, *a*)  $\in$ . *branch* $\rangle$

**using** *DisP*(8) **by** *simp*

**ultimately show** ?*case*

**using** *STA-drop-block'* **by** *fast*

**next**

**case** (*DisN p q a ps branch n*)

**have**  $\langle A \vdash$  ( $\neg$  *q*) # ( $\neg$  *p*) # *ps*, *a*) # *branch* $\rangle$

**using** *DisN*(4-) **by** (*simp add: subset-code*(1))

**moreover have**  $\langle$ ( $\neg$  (*p*  $\vee$  *q*)) at *a* in (*ps*, *a*) # *branch* $\rangle$

**using** *DisN*(1, 5) **by** *auto*

**ultimately have**  $\langle A \vdash$  (*ps*, *a*) # *branch* $\rangle$

**using** *DisN'* **by** *fast*

**moreover have**  $\langle$ (*ps*, *a*)  $\in$ . *branch* $\rangle$

**using** *DisN*(5) **by** *simp*

**ultimately show** ?*case*



```

    using STA-drop-block' by fast
next
case (DiaP p a ps branch i n)
have ⟨finite (A ∪ branch-nominals branch)⟩
  using fin by (simp add: finite-branch-nominals)
then obtain j where j: ⟨j ∉ A ∪ branch-nominals branch⟩
  using assms ex-new-if-finite by blast
then have j': ⟨j ∉ branch-nominals ((ps, a) # branch)⟩
  using DiaP(7) unfolding branch-nominals-def by blast

let ?f = ⟨id(i := j, j := i)⟩
let ?branch' = ⟨sub-branch ?f branch⟩
have branch': ⟨sub-branch ?f ?branch' = branch⟩
  using sub-branch-comp sub-branch-id swap-id by metis

have ⟨i ∉ branch-nominals ((ps, a) # branch)⟩
  using DiaP(2) by blast
then have branch: ⟨sub-branch ?f ((ps, a) # branch) = (ps, a) # branch⟩
  using DiaP(2) j' sub-branch-id sub-branch-upd-fresh by metis
moreover have
  ⟨set (sub-branch ?f ((ps, a) # branch)) ⊆ set ?branch'⟩
  using DiaP(7) sub-branch-image by blast
ultimately have *: ⟨set ((ps, a) # branch) ⊆ set ?branch'⟩
  unfolding sub-branch-def by auto

have ⟨i ∉ block-nominals (ps, a)⟩
  using DiaP unfolding branch-nominals-def by simp
moreover have ⟨i ∉ branch-nominals ?branch'⟩
  using j sub-branch-repl by fast
ultimately have i: ⟨i ∉ branch-nominals ((ps, a) # ?branch')⟩
  unfolding branch-nominals-def by simp

have ⟨?f ' A = A⟩
  using DiaP(2) j by auto

have ⟨A ⊢ ((@ i p) # (◇ Nom i) # ps, a) # ?branch'⟩
  using DiaP(6) *
  by (metis (no-types, lifting) subset-code(1) insert-mono list.set(2) set-subset-Cons)
moreover have ⟨(◇ p) at a in (ps, a) # ?branch'⟩
  using DiaP(1, 7) * by (meson set-subset-Cons subset-code(1))
ultimately have ⟨A ⊢ (ps, a) # ?branch'⟩
  using inf DiaP(2-3) fin i DiaP'' by (metis Un-iff)
then have ⟨?f ' A ⊢ sub-branch ?f ((ps, a) # ?branch')⟩
  using STA-sub fin by blast
then have ⟨A ⊢ (ps, a) # branch'⟩
  using ⟨?f ' A = A⟩ branch' branch unfolding sub-branch-def by simp
moreover have ⟨(ps, a) ∈ branch'⟩
  using ⟨set ((ps, a) # branch) ⊆ set branch'⟩ by simp
ultimately show ?case

```

```

    using STA-drop-block' by fast
next
case (DiaN p a ps branch i n)
have  $\langle A \vdash ((\neg (@ i p)) \# ps, a) \# branch \rangle$ 
  using DiaN(5-) by (simp add: subset-code(1))
moreover have
   $\langle (\neg (\diamond p)) \text{ at } a \text{ in } (ps, a) \# branch \rangle$ 
   $\langle (\diamond \text{Nom } i) \text{ at } a \text{ in } (ps, a) \# branch \rangle$ 
  using DiaN(1-2, 6) by auto
ultimately have  $\langle A \vdash (ps, a) \# branch \rangle$ 
  using DiaN' by fast
moreover have  $\langle (ps, a) \in. branch \rangle$ 
  using DiaN(6) by simp
ultimately show ?case
  using STA-drop-block' by fast
next
case (SatP a p b ps branch n)
have  $\langle A \vdash (p \# ps, a) \# branch \rangle$ 
  using SatP(4-) by (simp add: subset-code(1))
moreover have  $\langle (@ a p) \text{ at } b \text{ in } (ps, a) \# branch \rangle$ 
  using SatP(1, 5) by auto
ultimately have  $\langle A \vdash (ps, a) \# branch \rangle$ 
  using SatP' by fast
moreover have  $\langle (ps, a) \in. branch \rangle$ 
  using SatP(5) by simp
ultimately show ?case
  using STA-drop-block' by fast
next
case (SatN a p b ps branch n)
have  $\langle A \vdash ((\neg p) \# ps, a) \# branch \rangle$ 
  using SatN(4-) by (simp add: subset-code(1))
moreover have  $\langle (\neg (@ a p)) \text{ at } b \text{ in } (ps, a) \# branch \rangle$ 
  using SatN(1, 5) by auto
ultimately have  $\langle A \vdash (ps, a) \# branch \rangle$ 
  using SatN' by fast
moreover have  $\langle (ps, a) \in. branch \rangle$ 
  using SatN(5) by simp
ultimately show ?case
  using STA-drop-block' by fast
next
case (GoTo i branch n)
then have  $\langle A \vdash ([], i) \# branch \rangle$ 
  by (simp add: subset-code(1))
moreover have  $\langle i \in \text{branch-nominals } branch \rangle$ 
  using GoTo(1, 4) unfolding branch-nominals-def by auto
ultimately show ?case
  using GoTo(2) STA.GoTo by fast
next
case (Nom p b ps a branch n)

```

```

have  $\langle A \vdash (p \# ps, a) \# \text{branch}' \rangle$ 
  using Nom(6-) by (simp add: subset-code(1))
moreover have  $\langle p \text{ at } b \text{ in } (ps, a) \# \text{branch}' \rangle$ 
  using Nom(1, 7) by auto
moreover have  $\langle \text{Nom } a \text{ at } b \text{ in } (ps, a) \# \text{branch}' \rangle$ 
  using Nom(2, 7) by auto
ultimately have  $\langle A \vdash (ps, a) \# \text{branch}' \rangle$ 
  using Nom(3) Nom' by metis
moreover have  $\langle (ps, a) \in. \text{branch}' \rangle$ 
  using Nom(7) by simp
ultimately show ?case
  using STA-drop-block' by fast
qed

```

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 branch ::  $\langle ('a, 'b) \text{ branch} \rangle$ 
assumes
  inf:  $\langle \text{infinite } (UNIV :: 'b \text{ set}) \rangle$  and 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  $\langle A \vdash (ps', a) \# \text{branch} \rangle$ 
using assms(3-)
proof (induct n  $\langle (ps, a) \# \text{branch} \rangle$  arbitrary: ps ps' rule: STA.induct)
  case (Close p i n ts ts')
  then have  $\langle p \text{ at } i \text{ in } (ts', a) \# \text{branch} \rangle$   $\langle (\neg p) \text{ at } i \text{ in } (ts', a) \# \text{branch} \rangle$ 
    by auto
  then show ?case
    using STA.Close by fast
next
  case (Neg p ps n)
  then have  $\langle (\neg \neg p) \text{ at } a \text{ in } (ps', a) \# \text{branch} \rangle$ 
    by auto
  moreover have  $\langle A \vdash (p \# ps', a) \# \text{branch} \rangle$ 
    using Neg(4-) by (simp add: subset-code(1))
  ultimately show ?case
    using Neg' by fast
next
  case (DisP p q ps n)
  then have  $\langle (p \vee q) \text{ at } a \text{ in } (ps', a) \# \text{branch} \rangle$ 
    by auto
  moreover have  $\langle A \vdash (p \# ps', a) \# \text{branch} \rangle$   $\langle A \vdash (q \# ps', a) \# \text{branch} \rangle$ 
    using DisP(5, 7-) by (simp-all add: subset-code(1))
  ultimately show ?case
    using DisP'' by fast
next
  case (DisN p q ps n)
  then have  $\langle (\neg (p \vee q)) \text{ at } a \text{ in } (ps', a) \# \text{branch} \rangle$ 

```

```

    by auto
  moreover have  $\langle A \vdash ((\neg q) \# (\neg p) \# ps', a) \# \text{branch} \rangle$ 
    using  $\text{DisN}(\neg)$  by (simp add: subset-code(1))
  ultimately show ?case
    using  $\text{DisN}'$  by fast
next
  case (DiaP p ps i n)
  have  $\langle \text{finite } (A \cup \text{branch-nominals } ((ps', a) \# \text{branch})) \rangle$ 
    using fin finite-branch-nominals by blast
  then obtain j where  $\langle j \notin A \cup \text{branch-nominals } ((ps', a) \# \text{branch}) \rangle$ 
    using assms ex-new-if-finite by blast
  then have  $\langle j' \notin \text{block-nominals } (ps, a) \rangle$ 
    using DiaP.prem unfolding branch-nominals-def by auto

  let ?f =  $\langle \text{id } (i := j, j := i) \rangle$ 
  let ?ps' =  $\langle \text{sub-list } ?f \text{ } ps' \rangle$ 
  have  $\langle ps' : \langle \text{sub-list } ?f \text{ } ?ps' = ps' \rangle$ 
    using sub-list-comp sub-list-id swap-id by metis

  have  $\langle i \notin \text{block-nominals } (ps, a) \rangle$ 
    using DiaP(1-2) unfolding branch-nominals-def by simp
  then have  $\langle ps : \langle \text{sub-block } ?f \text{ } (ps, a) = (ps, a) \rangle$ 
    using j' sub-block-id sub-block-upd-fresh by metis
  moreover have  $\langle \text{set } (\text{sub-list } ?f \text{ } ps) \subseteq \text{set } (\text{sub-list } ?f \text{ } ps') \rangle$ 
    using  $\langle \text{set } ps \subseteq \text{set } ps' \rangle$  by auto
  ultimately have *:  $\langle \text{set } ps \subseteq \text{set } ?ps' \rangle$ 
    by simp

  have  $\langle i \notin \text{branch-nominals } \text{branch} \rangle$ 
    using DiaP unfolding branch-nominals-def by simp
  moreover have  $\langle j \notin \text{branch-nominals } \text{branch} \rangle$ 
    using j unfolding branch-nominals-def by simp
  ultimately have  $\langle \text{branch} : \langle \text{sub-branch } ?f \text{ } \text{branch} = \text{branch} \rangle$ 
    using sub-branch-id sub-branch-upd-fresh by metis

  have  $\langle i \neq a \rangle \langle j \neq a \rangle$ 
    using DiaP j unfolding branch-nominals-def by simp-all
  then have  $\langle ?f \text{ } a = a \rangle$ 
    by simp
  moreover have  $\langle j \notin \text{block-nominals } (ps', a) \rangle$ 
    using j unfolding branch-nominals-def by simp
  ultimately have  $\langle i \notin \text{block-nominals } (?ps', a) \rangle$ 
    using sub-block-repl[where block= $\langle (ps', a) \rangle$  and  $i=i$  and  $j=j$ ] by simp

  have  $\langle ?f \text{ } A = A \rangle$ 
    using DiaP(2) j by auto

  have  $\langle (\Diamond p) \text{ at } a \text{ in } (?ps', a) \# \text{branch} \rangle$ 
    using DiaP(1) * by fastforce

```

```

moreover have  $\langle A \vdash ((@ i p) \# (\diamond Nom i) \# ?ps', a) \# branch \rangle$ 
  using * DiaP(6) fin by (simp add: subset-code(1))
moreover have  $\langle i \notin A \cup branch\text{-nominals } ((?ps', a) \# branch) \rangle$ 
  using DiaP(2)  $\langle i \notin block\text{-nominals } (?ps', a) \rangle$  unfolding branch-nominals-def
by simp
ultimately have  $\langle A \vdash (?ps', a) \# branch \rangle$ 
  using DiaP(3) fin DiaP'' by metis
then have  $\langle ?f ' A \vdash sub\text{-branch } ?f ((?ps', a) \# branch) \rangle$ 
  using STA-sub fin by blast
then have  $\langle A \vdash (sub\text{-list } ?f ?ps', ?f a) \# sub\text{-branch } ?f branch \rangle$ 
  unfolding sub-branch-def using  $\langle ?f ' A = A \rangle$  by simp
then show ?case
  using  $\langle ?f a = a \rangle$  ps' branch by simp
next
case (DiaN p ps i n)
then have
   $\langle (\neg (\diamond p)) \text{ at } a \text{ in } (ps', a) \# branch \rangle$ 
   $\langle (\diamond Nom i) \text{ at } a \text{ in } (ps', a) \# branch \rangle$ 
  by auto
moreover have  $\langle A \vdash ((\neg (@ i p)) \# ps', a) \# branch \rangle$ 
  using DiaN(5-) by (simp add: subset-code(1))
ultimately show ?case
  using DiaN' by fast
next
case (SatP p b ps n)
then have  $\langle (@ a p) \text{ at } b \text{ in } (ps', a) \# branch \rangle$ 
  by auto
moreover have  $\langle A \vdash (p \# ps', a) \# branch \rangle$ 
  using SatP(4-) by (simp add: subset-code(1))
ultimately show ?case
  using SatP' by fast
next
case (SatN p b ps n)
then have  $\langle (\neg (@ a p)) \text{ at } b \text{ in } (ps', a) \# branch \rangle$ 
  by auto
moreover have  $\langle A \vdash ((\neg p) \# ps', a) \# branch \rangle$ 
  using SatN(4-) by (simp add: subset-code(1))
ultimately show ?case
  using SatN' by fast
next
case (GoTo i n ps)
then have  $\langle A, Suc n \vdash (ps, a) \# branch \rangle$ 
  using STA.GoTo by fast
then obtain m where  $\langle A, m \vdash (ps, a) \# (ps', a) \# branch \rangle$ 
  using inf fin STA-struct[where branch' =  $\langle (ps, a) \# - \# - \rangle$ ] by fastforce
then have  $\langle A, Suc m \vdash (ps', a) \# branch \rangle$ 
  using GoTo(4) by (simp add: STA-drop-block[where a=a])
then show ?case
  by blast

```

```

next
  case  $\langle \text{Nom } p \text{ b } ps \ n \rangle$ 
  have  $\langle p \text{ at } b \text{ in } (ps', a) \# \text{ branch} \rangle$ 
    using  $\text{Nom}(1, \gamma)$  by auto
  moreover have  $\langle \text{Nom } a \text{ at } b \text{ in } (ps', a) \# \text{ branch} \rangle$ 
    using  $\text{Nom}(2, \gamma)$  by auto
  moreover have  $\langle A \vdash (p \# ps', a) \# \text{ branch} \rangle$ 
    using  $\text{Nom}(6-)$  by (simp add: subset-code(1))
  ultimately show ?case
    using  $\text{Nom}(3)$   $\text{Nom}'$  by metis
qed

```

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

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

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

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

**proof** (*induct p*)  
**case** (*Neg p*)

```

then show ?case by (cases p) auto
next
  case (Dia p)
  then show ?case by (cases p) auto
qed auto

```

```

lemma bridge-nominals:
  ⟨nominals (bridge k j xs v v' p) ∪ {k, j} = nominals p ∪ {k, j}⟩
proof (cases ⟨(v, v') ∈ xs⟩)
  case True
  then have ⟨nominals (bridge k j xs v v' p) = nominals (bridge' k j p)⟩
    by simp
  then show ?thesis
    using bridge'-nominals by metis
qed simp

```

```

lemma bridge-block-nominals:
  ⟨block-nominals (mapi-block (bridge k j xs v) (ps, a)) ∪ {k, j} =
    block-nominals (ps, a) ∪ {k, j}⟩
proof (induct ps)
  case Nil
  then show ?case
    by simp
next
  case (Cons p ps)
  have ⟨?case ⟷
    (nominals (bridge k j xs v (length ps) p)) ∪
    (block-nominals (mapi-block (bridge k j xs v) (ps, a)) ∪ {k, j}) =
    (nominals p) ∪ (block-nominals (ps, a) ∪ {k, j})⟩
    by simp
  also have ⟨... ⟷
    (nominals (bridge k j xs v (length ps) p) ∪ {k, j}) ∪
    (block-nominals (mapi-block (bridge k j xs v) (ps, a)) ∪ {k, j}) =
    (nominals p ∪ {k, j}) ∪ (block-nominals (ps, a) ∪ {k, j})⟩
    by blast
  moreover have
    ⟨nominals (bridge k j xs v (length ps) p) ∪ {k, j} = nominals p ∪ {k, j}⟩
    using bridge-nominals by metis
  moreover note Cons
  ultimately show ?case
    by argo
qed

```

```

lemma bridge-branch-nominals:
  ⟨branch-nominals (mapi-branch (bridge k j xs) branch) ∪ {k, j} =
    branch-nominals branch ∪ {k, j}⟩
proof (induct branch)
  case Nil
  then show ?case

```

**unfolding** *branch-nominals-def mapi-branch-def*  
**by** *simp*  
**next**  
**case** (*Cons block branch*)  
**have**  $\langle ?case \longleftrightarrow$   
 $(\text{block-nominals } (\text{mapi-block } (\text{bridge } k \ j \ xs \ (\text{length } \text{branch}))) \ \text{block}) \cup$   
 $(\text{branch-nominals } (\text{mapi-branch } (\text{bridge } k \ j \ xs) \ \text{branch}) \cup \{k, j\}) =$   
 $(\text{block-nominals } \text{block}) \cup (\text{branch-nominals } \text{branch} \cup \{k, j\}) \rangle$   
**unfolding** *branch-nominals-def mapi-branch-def* **by** *simp*  
**also have**  $\langle \dots \longleftrightarrow$   
 $(\text{block-nominals } (\text{mapi-block } (\text{bridge } k \ j \ xs \ (\text{length } \text{branch}))) \ \text{block}) \cup \{k, j\} \cup$   
 $(\text{branch-nominals } (\text{mapi-branch } (\text{bridge } k \ j \ xs) \ \text{branch}) \cup \{k, j\}) =$   
 $(\text{block-nominals } \text{block} \cup \{k, j\}) \cup (\text{branch-nominals } \text{branch} \cup \{k, j\}) \rangle$   
**by** *blast*  
**moreover have**  
 $\langle \text{block-nominals } (\text{mapi-block } (\text{bridge } k \ j \ xs \ (\text{length } \text{branch}))) \ \text{block} \cup \{k, j\} =$   
 $\text{block-nominals } \text{block} \cup \{k, j\} \rangle$   
**using** *bridge-block-nominals* [**where**  $ps = \langle \text{fst } \text{block} \rangle$  **and**  $a = \langle \text{snd } \text{block} \rangle$ ] **by** *simp*  
**ultimately show** *?case*  
**using** *Cons* **by** *argo*  
**qed**

**lemma** *at-in-mapi-branch*:  
**assumes**  $\langle p \ \text{at } a \ \text{in } \text{branch} \rangle \langle p \neq \text{Nom } a \rangle$   
**shows**  $\langle \exists v \ v'. \ f \ v \ v' \ p \ \text{at } a \ \text{in } \text{mapi-branch } f \ \text{branch} \rangle$   
**using** *assms* **by** (*meson mapi-branch-mem rev-nth-mapi-block rev-nth-on*)

**lemma** *nom-at-in-bridge*:  
**fixes**  $k \ j \ xs$   
**defines**  $\langle f \equiv \text{bridge } k \ j \ xs \rangle$   
**assumes**  $\langle \text{Nom } i \ \text{at } a \ \text{in } \text{branch} \rangle$   
**shows**  $\langle \text{Nom } i \ \text{at } a \ \text{in } \text{mapi-branch } f \ \text{branch} \rangle$   
**proof** –  
**obtain**  $qs$  **where**  $qs: \langle (qs, a) \in. \text{branch} \rangle \langle \text{Nom } i \ \text{on } (qs, a) \rangle$   
**using** *assms(2)* **by** *blast*  
**then obtain**  $l$  **where**  $\langle (\text{mapi } (f \ l) \ qs, a) \in. \text{mapi-branch } f \ \text{branch} \rangle$   
**using** *mapi-branch-mem* **by** *fast*  
**moreover have**  $\langle \text{Nom } i \ \text{on } (\text{mapi } (f \ l) \ qs, a) \rangle$   
**unfolding** *f-def* **using** *qs(2)* **by** (*induct qs*) *auto*  
**ultimately show** *?thesis*  
**by** *blast*  
**qed**

**lemma** *nominals-mapi-branch-bridge*:  
**assumes**  $\langle \text{Nom } k \ \text{at } j \ \text{in } \text{branch} \rangle$   
**shows**  $\langle \text{branch-nominals } (\text{mapi-branch } (\text{bridge } k \ j \ xs) \ \text{branch}) = \text{branch-nominals } \text{branch} \rangle$   
**proof** –  
**let**  $?f = \langle \text{bridge } k \ j \ xs \rangle$



**have**  $\langle \text{Nom } k \text{ at } j \text{ in mapi-branch } ?f \text{ branch} \rangle$   
**using** *assms nom-at-in-bridge by fast*  
**then have**  
 $\langle j \in \text{branch-nominals (mapi-branch } ?f \text{ branch)} \rangle$   
 $\langle k \in \text{branch-nominals (mapi-branch } ?f \text{ branch)} \rangle$   
**unfolding** *branch-nominals-def by fastforce+*  
**moreover have**  $\langle j \in \text{branch-nominals branch} \rangle \langle k \in \text{branch-nominals branch} \rangle$   
**using** *assms unfolding branch-nominals-def by fastforce+*  
**moreover have**  
 $\langle \text{branch-nominals (mapi-branch } ?f \text{ branch)} \cup \{k, j\} = \text{branch-nominals branch} \cup \{k, j\} \rangle$   
**using** *bridge-branch-nominals by metis*  
**ultimately show** *?thesis*  
**by** *blast*  
**qed**

**lemma** *bridge-proper-dia:*  
**assumes**  $\langle \# a. p = \text{Nom } a \rangle$   
**shows**  $\langle \text{bridge } k \ j \ xs \ v \ v' \ (\Diamond p) = (\Diamond p) \rangle$   
**using** *assms by (induct p) simp-all*

**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**  
 $\langle (q = (\Diamond \text{Nom } j) \wedge q' = (\neg (\Diamond \text{Nom } k))) \vee$   
 $(\exists r. q = (\neg (@ j r)) \wedge q' = (\neg \neg (@ k r))) \vee$   
 $(\exists r. q = (@ k r) \wedge q' = (\neg (@ j r))) \vee$   
 $(q = p \wedge q' = (\neg p)) \rangle$   
**proof** *(cases p)*  
**case** *(Neg p)*  
**then show** *?thesis*  
**by** *(cases p) (simp-all add: q-def q'-def)*  
**next**  
**case** *(Dia p)*  
**then show** *?thesis*  
**by** *(cases p) (simp-all add: q-def q'-def)*  
**qed** *(simp-all add: q-def q'-def)*

## 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:*

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

$\langle \text{branch } !. w = \text{Some } (rs, a) \implies rs !. w' = \text{Some } (\diamond \text{Nom } k) \implies \text{descendants } k \text{ } i \text{ branch } (\{(v, v')\} \cup xs) \rangle$

| Copied:

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

**lemma** *descendants-initial*:

**assumes**  $\langle \text{descendants } k \text{ } i \text{ branch } xs \rangle$

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

**using** *assms* **by**  $(\text{induct } k \text{ } i \text{ branch } xs \text{ rule: descendants.induct}) \text{ simp-all}$

**lemma** *descendants-bounds-fst*:

**assumes**  $\langle \text{descendants } k \text{ } i \text{ branch } xs \rangle \langle (v, v') \in xs \rangle$

**shows**  $\langle v < \text{length branch} \rangle$

**using** *assms rev-nth-Some*

**by**  $(\text{induct } k \text{ } i \text{ branch } xs \text{ rule: descendants.induct}) \text{ fast+}$

**lemma** *descendants-bounds-snd*:

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

**shows**  $\langle v' < \text{length } ps \rangle$

**using** *assms*

**by**  $(\text{induct } k \text{ } i \text{ branch } xs \text{ rule: descendants.induct}) (\text{auto simp: rev-nth-Some})$

**lemma** *descendants-branch*:

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

**proof**  $(\text{induct } k \text{ } i \text{ branch } xs \text{ rule: descendants.induct})$

**case**  $(\text{Initial branch } v \text{ } qs \text{ } i \text{ } v' \text{ } k)$

**then show** *?case*

**using** *rev-nth-append descendants.Initial* **by** *fast*

**next**

**case**  $(\text{Derived branch } v \text{ } qs \text{ } a \text{ } v' \text{ } k \text{ } p \text{ } i \text{ } xs \text{ } w \text{ } w' \text{ } rs)$

**then have**

$\langle (\text{extra @ branch}) !. v = \text{Some } (qs, a) \rangle$

$\langle (\text{extra @ branch}) !. w = \text{Some } (rs, a) \rangle$

**using** *rev-nth-append* **by** *fast+*

**then show** *?case*

**using** *Derived(2, 4-5, 7) descendants.Derived* **by** *fast*

**next**

**case**  $(\text{Copied branch } v \text{ } qs \text{ } a \text{ } v' \text{ } p \text{ } k \text{ } i \text{ } xs \text{ } w \text{ } w' \text{ } rs \text{ } b)$

**then have**

$\langle (\text{extra @ branch}) !. v = \text{Some } (qs, a) \rangle$

$\langle (\text{extra @ branch}) !. w = \text{Some } (rs, b) \rangle$

**using** *rev-nth-append* **by** *fast+*

**moreover have**  $\langle \text{Nom } a \text{ at } b \text{ in (extra @ branch)} \rangle$

**using** *Copied(8)* **by** *auto*

```

ultimately show ?case
  using Copied(2-4, 5-7) descendants.Copied by fast
qed

lemma descendants-block:
  assumes ⟨descendants k i ((ps, a) # branch) xs⟩
  shows ⟨descendants k i ((ps' @ ps, a) # branch) xs⟩
  using assms
proof (induct k i ⟨(ps, a) # branch⟩ xs arbitrary: ps a branch rule: descen-
dants.induct)
  case (Initial v qs i v' k)
  have
    ⟨((ps' @ ps, a) # branch) !. v = Some (qs, i) ∨
    ((ps' @ ps, a) # branch) !. v = Some (ps' @ qs, i)⟩
    using Initial(1) by auto
  moreover have
    ⟨qs !. v' = Some (◇ Nom k)⟩ ⟨(ps' @ qs) !. v' = Some (◇ Nom k)⟩
    using Initial(2) rev-nth-append by simp-all
  ultimately show ?case
    using descendants.Initial by fast
next
  case (Derived v qs a' v' k p i xs w w' rs)
  have
    ⟨((ps' @ ps, a) # branch) !. v = Some (qs, a') ∨
    ((ps' @ ps, a) # branch) !. v = Some (ps' @ qs, a')⟩
    using Derived(1) by auto
  moreover have
    ⟨qs !. v' = Some (¬ (@ k p))⟩ ⟨(ps' @ qs) !. v' = Some (¬ (@ k p))⟩
    using Derived(2) rev-nth-append by simp-all
  moreover have
    ⟨((ps' @ ps, a) # branch) !. w = Some (rs, a') ∨
    ((ps' @ ps, a) # branch) !. w = Some (ps' @ rs, a')⟩
    using ⟨((ps, a) # branch) !. w = Some (rs, a')⟩ by auto
  moreover have
    ⟨rs !. w' = Some (◇ Nom k)⟩ ⟨(ps' @ rs) !. w' = Some (◇ Nom k)⟩
    using Derived(7) rev-nth-append by simp-all
  ultimately show ?case
    using Derived(4-5) descendants.Derived by fast
next
  case (Copied v qs a' v' p k i xs w w' rs b)
  have
    ⟨((ps' @ ps, a) # branch) !. v = Some (qs, a') ∨
    ((ps' @ ps, a) # branch) !. v = Some (ps' @ qs, a')⟩
    using Copied(1) by auto
  moreover have ⟨qs !. v' = Some p⟩ ⟨(ps' @ qs) !. v' = Some p⟩
    using Copied(2) rev-nth-append by simp-all
  moreover have
    ⟨((ps' @ ps, a) # branch) !. w = Some (rs, b) ∨
    ((ps' @ ps, a) # branch) !. w = Some (ps' @ rs, b)⟩

```

**using** *Copied(6)* **by** *auto*  
**moreover have**  $\langle rs \ !. \ w' = \text{Some } p \rangle \langle ps' \ @ \ rs \ !. \ w' = \text{Some } p \rangle$   
**using** *Copied(7)* *rev-nth-append* **by** *simp-all*  
**moreover have**  
 $\langle \langle ps' \ @ \ ps, a \rangle \# \text{branch} \rangle \ !. \ w = \text{Some } (rs, b) \vee$   
 $\langle \langle ps' \ @ \ ps, a \rangle \# \text{branch} \rangle \ !. \ w = \text{Some } (ps' \ @ \ rs, b) \rangle$   
**using** *Copied(6)* **by** *auto*  
**moreover have**  $\langle rs \ !. \ w' = \text{Some } p \rangle \langle ps' \ @ \ rs \ !. \ w' = \text{Some } p \rangle$   
**using** *Copied(7)* *rev-nth-append* **by** *simp-all*  
**moreover have**  $\langle \text{Nom } a' \text{ at } b \text{ in } (ps' \ @ \ ps, a) \# \text{branch} \rangle$   
**using** *Copied(8)* **by** *fastforce*  
**ultimately show** *?case*  
**using** *Copied(4-5)* *descendants.Copied* [**where**  $\text{branch} = \langle ps' \ @ \ ps, a \rangle \#$   
 $\text{branch}$ ] **by** *blast*  
**qed**

**lemma** *descendants-no-head*:  
**assumes**  $\langle \text{descendants } k \ i \ ((ps, a) \# \text{branch}) \ xs \rangle$   
**shows**  $\langle \text{descendants } k \ i \ ((p \ # \ ps, a) \# \text{branch}) \ xs \rangle$   
**using** *assms descendants-block* [**where**  $ps' = [-]$ ] **by** *simp*

**lemma** *descendants-types*:  
**assumes**  
 $\langle \text{descendants } k \ i \ \text{branch } \ xs \rangle \langle (v, v') \in xs \rangle$   
 $\langle \text{branch} \ !. \ v = \text{Some } (ps, a) \rangle \langle ps \ !. \ v' = \text{Some } p \rangle$   
**shows**  $\langle p = (\diamond \text{Nom } k) \vee (\exists q. p = (\neg (@ \ k \ q))) \rangle$   
**using** *assms* **by** (*induct*  $k \ i \ \text{branch } \ xs$  *arbitrary: v v' ps a*) *fastforce+*

**lemma** *descendants-oob-head'*:  
**assumes**  $\langle \text{descendants } k \ i \ ((ps, a) \# \text{branch}) \ xs \rangle$   
**shows**  $\langle (\text{length } \text{branch}, m + \text{length } ps) \notin xs \rangle$   
**using** *assms descendants-bounds-snd* **by** *fastforce*

**lemma** *descendants-oob-head*:  
**assumes**  $\langle \text{descendants } k \ i \ ((ps, a) \# \text{branch}) \ xs \rangle$   
**shows**  $\langle (\text{length } \text{branch}, \text{length } ps) \notin xs \rangle$   
**using** *assms descendants-oob-head'* [**where**  $m=0$ ] **by** *fastforce*

### 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**  
 $\text{inf}: \langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$  **and**  $\text{fin}: \langle \text{finite } A \rangle$  **and**  $\langle j \in A \rangle$   
 $\langle A, n \vdash (ps, a) \# \text{branch} \rangle$   
 $\langle \text{descendants } k \ i \ ((ps, a) \# \text{branch}) \ xs \rangle$

```

  ⟨Nom k at j in branch⟩
shows ⟨A ⊢ mapi-branch (bridge k j xs) ((ps, a) # branch)⟩
using assms(4-)
proof (induct n ⟨(ps, a) # branch⟩ arbitrary: ps a branch xs rule: STA.induct)
case (Close p i' n)
let ?f = ⟨bridge k j xs⟩
let ?branch = ⟨mapi-branch ?f ((ps, a) # branch)⟩

obtain qs where qs: ⟨(qs, i') ∈. (ps, a) # branch⟩ ⟨p on (qs, i')⟩
using Close(1) by blast
obtain rs where rs: ⟨(rs, i') ∈. (ps, a) # branch⟩ ⟨¬ p on (rs, i')⟩
using Close(2) by blast

obtain v where v: ⟨(mapi (?f v) qs, i') ∈. ?branch⟩
using qs mapi-branch-mem by fast
obtain w where w: ⟨(mapi (?f w) rs, i') ∈. ?branch⟩
using rs mapi-branch-mem by fast

have k: ⟨Nom k at j in ?branch⟩
using Close(4) nom-at-in-bridge unfolding mapi-branch-def by fastforce

show ?case
proof (cases ⟨∃ a. p = Nom a⟩)
case True
then have ⟨p on (mapi (?f v) qs, i')⟩
using qs bridge-on-Nom by fast
moreover have ⟨¬ p on (mapi (?f w) rs, i')⟩
using rs(2) True by (induct rs) auto
ultimately show ?thesis
using v w STA.Close by fast
next
case False
then obtain v' where ⟨qs !. v' = Some p⟩
using qs rev-nth-on by fast
then have qs': ⟨(?f v v' p) on (mapi (?f v) qs, i')⟩
using rev-nth-mapi-block by fast

then obtain w' where ⟨rs !. w' = Some (¬ p)⟩
using rs rev-nth-on by fast
then have rs': ⟨(?f w w' (¬ p)) on (mapi (?f w) rs, i')⟩
using rev-nth-mapi-block by fast

obtain q q' where q: ⟨?f v v' p = q⟩ and q': ⟨?f w w' (¬ p) = q'⟩
by simp-all
then consider
  (dia) ⟨q = (◇ Nom j)⟩ ⟨q' = (¬ (◇ Nom k))⟩ |
  (satn) ⟨∃ r. q = (¬ (@ j r)) ∧ q' = (¬ ¬ (@ k r))⟩ |
  (sat) ⟨∃ r. q = (@ k r) ∧ q' = (¬ (@ j r))⟩ |
  (old) ⟨q = p⟩ ⟨q' = (¬ p)⟩

```

```

    using bridge-compl-cases by fast
  then show ?thesis
proof cases
  case dia
  then have *:
    ⟨⟨◇ Nom j⟩ on (mapi (?f v) qs, i')⟩
    ⟨⟨¬ (◇ Nom k)⟩ on (mapi (?f w) rs, i')⟩
    using q qs' q' rs' by simp-all

  have ⟨i' ∈ branch-nominals ?branch⟩
    unfolding branch-nominals-def using v by fastforce
  then have ?thesis if ⟨A ⊢ ([], i') # ?branch⟩
    using that GoTo by fast
  moreover have ⟨(mapi (?f v) qs, i') ∈. ([], i') # ?branch⟩
    using v by simp
  moreover have ⟨(mapi (?f w) rs, i') ∈. ([], i') # ?branch⟩
    using w by simp
  ultimately have ?thesis if ⟨A ⊢ ([¬ (@ j (Nom k))], i') # ?branch⟩
    using that * by (meson DiaN')
  moreover have ⟨j ∈ branch-nominals ([¬ (@ j (Nom k))], i') # ?branch⟩
    unfolding branch-nominals-def by simp
  ultimately have ?thesis if ⟨A ⊢ ([], j) # ([¬ (@ j (Nom k))], i') # ?branch⟩
    using that GoTo by fast
  moreover have ⟨⟨¬ (@ j (Nom k))⟩ at i' in ([], j) # ([¬ (@ j (Nom k))],
i') # ?branch⟩
    by fastforce
  ultimately have ?thesis if ⟨A ⊢ ([¬ Nom k], j) # ([¬ (@ j (Nom k))], i')
# ?branch⟩
    using that SatN' by fast
  moreover have ⟨Nom k at j in ([¬ Nom k], j) # ([¬ (@ j (Nom k))], i') #
?branch⟩
    using k by fastforce
  moreover have ⟨⟨¬ Nom k⟩ at j in ([¬ Nom k], j) # ([¬ (@ j (Nom k))],
i') # ?branch⟩
    by fastforce
  ultimately show ?thesis
    using STA.Close by fast
next
case satn
  then obtain r where *:
    ⟨⟨¬ (@ j r)⟩ on (mapi (?f v) qs, i')⟩
    ⟨⟨¬ ¬ (@ k r)⟩ on (mapi (?f w) rs, i')⟩
    using q qs' q' rs' by auto

  have ⟨i' ∈ branch-nominals ?branch⟩
    unfolding branch-nominals-def using v by fastforce
  then have ?thesis if ⟨A ⊢ ([], i') # ?branch⟩
    using that GoTo by fast
  moreover have ⟨(mapi (?f w) rs, i') ∈. ([], i') # ?branch⟩

```

using  $w$  by *simp*  
 ultimately have *?thesis* if  $\langle A \vdash ([@ k r], i') \# ?branch \rangle$   
 using that  $*(2)$  by (*meson Neg'*)  
 moreover have  $\langle j \in \text{branch-nominals } ([@ k r], i') \# ?branch \rangle$   
 unfolding *branch-nominals-def* using  $k$  by *fastforce*  
 ultimately have *?thesis* if  $\langle A \vdash ([], j) \# ([@ k r], i') \# ?branch \rangle$   
 using that *GoTo* by *fast*  
 moreover have  $\langle (\neg (@ j r)) \text{ at } i' \text{ in } ([], j) \# ([@ k r], i') \# ?branch \rangle$   
 using  $*(1)$   $v$  by *auto*  
 ultimately have *?thesis* if  $\langle A \vdash ([\neg r], j) \# ([@ k r], i') \# ?branch \rangle$   
 using that *SatN'* by *fast*  
 moreover have  $\langle k \in \text{branch-nominals } ([\neg r], j) \# ([@ k r], i') \# ?branch \rangle$   
 unfolding *branch-nominals-def* using  $k$  by *fastforce*  
 ultimately have *?thesis* if  $\langle A \vdash ([], k) \# ([\neg r], j) \# ([@ k r], i') \# ?branch \rangle$   
 using that *GoTo* by *fast*  
 moreover have  $\langle (@ k r) \text{ at } i' \text{ in } ([], k) \# ([\neg r], j) \# ([@ k r], i') \# ?branch \rangle$   
 by *fastforce*  
 ultimately have *?thesis* if  $\langle A \vdash ([r], k) \# ([\neg r], j) \# ([@ k r], i') \# ?branch \rangle$   
 using that *SatP'* by *fast*  
 moreover have  
 $\langle \text{Nom } k \text{ at } j \text{ in } ([r], k) \# ([\neg r], j) \# ([@ k r], i') \# ?branch \rangle$   
 $\langle (\neg r) \text{ at } j \text{ in } ([r], k) \# ([\neg r], j) \# ([@ k r], i') \# ?branch \rangle$   
 using  $k$  by *fastforce+*  
 ultimately have *?thesis* if  $\langle A \vdash ([\neg r, r], k) \# ([\neg r], j) \# ([@ k r], i') \# ?branch \rangle$   
 using that by (*meson Nom' fm.distinct(21) fm.simps(18)*)  
 moreover have  
 $\langle r \text{ at } k \text{ in } ([\neg r, r], k) \# ([\neg r], j) \# ([@ k r], i') \# ?branch \rangle$   
 $\langle (\neg r) \text{ at } k \text{ in } ([\neg r, r], k) \# ([\neg r], j) \# ([@ k r], i') \# ?branch \rangle$   
 by *fastforce+*  
 ultimately show *?thesis*  
 using *STA.Close* by *fast*  
 next  
 case *sat*  
 then obtain  $r$  where  $*$ :  
 $\langle (@ k r) \text{ on } (\text{mapi } (?f v) qs, i') \rangle$   
 $\langle (\neg (@ j r)) \text{ on } (\text{mapi } (?f w) rs, i') \rangle$   
 using  $q qs' q' rs'$  by *auto*  
  
 have  $\langle j \in \text{branch-nominals } ?branch \rangle$   
 unfolding *branch-nominals-def* using  $k$  by *fastforce*  
 then have *?thesis* if  $\langle A \vdash ([], j) \# ?branch \rangle$   
 using that *GoTo* by *fast*  
 moreover have  $\langle (\neg (@ j r)) \text{ at } i' \text{ in } ([], j) \# ?branch \rangle$   
 using  $*(2)$   $w$  by *auto*  
 ultimately have *?thesis* if  $\langle A \vdash ([\neg r], j) \# ?branch \rangle$   
 using that by (*meson SatN'*)  
 moreover have  $\langle k \in \text{branch-nominals } ([\neg r], j) \# ?branch \rangle$

**unfolding** *branch-nominals-def* **using**  $k$  **by** *fastforce*  
**ultimately have**  $\langle A \vdash ([], k) \# ([\neg r], j) \# ?branch \rangle$   
**using** *that GoTo by fast*  
**moreover have**  $\langle (@ k r) \text{ at } i' \text{ in } ([], k) \# ([\neg r], j) \# ?branch \rangle$   
**using**  $*(1)$   $v$  **by** *auto*  
**ultimately have**  $\langle A \vdash ([r], k) \# ([\neg r], j) \# ?branch \rangle$   
**using** *that SatP' by fast*  
**moreover have**  
 $\langle \text{Nom } k \text{ at } j \text{ in } ([r], k) \# ([\neg r], j) \# ?branch \rangle$   
 $\langle (\neg r) \text{ at } j \text{ in } ([r], k) \# ([\neg r], j) \# ?branch \rangle$   
**using**  $k$  **by** *fastforce+*  
**ultimately have**  $\langle A \vdash ([\neg r, r], k) \# ([\neg r], j) \# ?branch \rangle$   
**using** *that by (meson Nom' fm.distinct(21) fm.simps(18))*  
**moreover have**  
 $\langle r \text{ at } k \text{ in } ([\neg r, r], k) \# ([\neg r], j) \# ?branch \rangle$   
 $\langle (\neg r) \text{ at } k \text{ in } ([\neg r, r], k) \# ([\neg r], j) \# ?branch \rangle$   
**by** *fastforce+*  
**ultimately show**  $?thesis$   
**using** *STA.Close by fast*  
**next**  
**case** *old*  
**then have**  $\langle p \text{ on } (mapi (?f v) qs, i') \rangle \langle (\neg p) \text{ on } (mapi (?f w) rs, i') \rangle$   
**using**  $q$   $qs'$   $q'$   $rs'$  **by** *simp-all*  
**then show**  $?thesis$   
**using**  $v$   $w$  *STA.Close* **[where**  $p=p$  **and**  $i=i'$  **]** **by** *fast*  
**qed**  
**qed**  
**next**  
**case**  $(Neg\ p\ a\ ps\ branch\ n)$   
**let**  $?f = \langle \text{bridge } k\ j\ xs \rangle$   
**have**  $p: \langle ?f\ l\ l' (\neg \neg p) = (\neg \neg p) \rangle$  **for**  $l\ l'$   
**by** *simp*  
  
**have**  $\langle \text{descendants } k\ i\ ((p \# ps, a) \# branch)\ xs \rangle$   
**using** *Neg(5) descendants-no-head by fast*  
**then have**  $\langle A \vdash \text{mapi-branch } ?f\ ((p \# ps, a) \# branch) \rangle$   
**using** *Neg(4-) by blast*  
**moreover have**  $\langle \text{length } branch, \text{length } ps \notin xs \rangle$   
**using** *Neg(5) descendants-oob-head by fast*  
**ultimately have**  $\langle A \vdash (p \# \text{mapi } (?f\ (\text{length } branch)))\ ps, a \# \text{mapi-branch } ?f\ branch \rangle$   
**unfolding** *mapi-branch-def* **by** *simp*  
**moreover have**  $\langle \exists l\ l'. ?f\ l\ l' (\neg \neg p) \text{ at } a \text{ in } \text{mapi-branch } ?f\ ((ps, a) \# branch) \rangle$   
**using** *Neg(1) at-in-mapi-branch by fast*  
**then have**  $\langle (\neg \neg p) \text{ at } a \text{ in } (\text{mapi } (?f\ (\text{length } branch)))\ ps, a \# \text{mapi-branch } ?f\ branch \rangle$   
**unfolding** *mapi-branch-def* **using**  $p$  **by** *simp*  
**ultimately have**  $\langle A \vdash (\text{mapi } (?f\ (\text{length } branch)))\ ps, a \# \text{mapi-branch } ?f\ branch \rangle$



```

    using Neg' by fast
  then show ?case
    unfolding mapi-branch-def by auto
next
case (DisP p q a ps branch n)
let ?f = ⟨bridge k j xs⟩
have p: ⟨?f l l' (p ∨ q) = (p ∨ q)⟩ for l l'
  by simp

have
  ⟨descendants k i ((p # ps, a) # branch) xs⟩
  ⟨descendants k i ((q # ps, a) # branch) xs⟩
  using DisP(8) descendants-no-head by fast+
then have
  ⟨A ⊢ mapi-branch ?f ((p # ps, a) # branch)⟩
  ⟨A ⊢ mapi-branch ?f ((q # ps, a) # branch)⟩
  using DisP(5-) by blast+
moreover have ⟨length branch, length ps⟩ ∉ xs
  using DisP(8) descendants-oob-head by fast
ultimately have
  ⟨A ⊢ (p # mapi (?f (length branch)) ps, a) # mapi-branch ?f branch)⟩
  ⟨A ⊢ (q # mapi (?f (length branch)) ps, a) # mapi-branch ?f branch)⟩
  unfolding mapi-branch-def by simp-all
moreover have ⟨∃ l l'. ?f l l' (p ∨ q) at a in mapi-branch ?f ((ps, a) # branch)⟩
  using DisP(1) at-in-mapi-branch by fast
then have ⟨(p ∨ q) at a in (mapi (?f (length branch)) ps, a) # mapi-branch ?f
branch)⟩
  unfolding mapi-branch-def using p by simp
ultimately have ⟨A ⊢ (mapi (?f (length branch)) ps, a) # mapi-branch ?f
branch)⟩
  using DisP'' by fast
then show ?case
  unfolding mapi-branch-def by auto
next
case (DisN p q a ps branch n)
let ?f = ⟨bridge k j xs⟩
have p: ⟨?f l l' (¬ (p ∨ q)) = (¬ (p ∨ q))⟩ for l l'
  by simp

have ⟨descendants k i ((¬ p) # ps, a) # branch) xs⟩
  using DisN(5) descendants-no-head by fast
then have ⟨descendants k i ((¬ q) # (¬ p) # ps, a) # branch) xs⟩
  using descendants-no-head by fast
then have ⟨A ⊢ mapi-branch ?f (((¬ q) # (¬ p) # ps, a) # branch)⟩
  using DisN(4-) by blast
moreover have ⟨length branch, length ps⟩ ∉ xs
  using DisN(5) descendants-oob-head by fast
moreover have ⟨length branch, 1 + length ps⟩ ∉ xs
  using DisN(5) descendants-oob-head' by fast

```

**ultimately have**  $\langle A \vdash ((\neg q) \# (\neg p) \# \text{mapi } (?f \text{ (length branch)) } ps, a) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**unfolding** *mapi-branch-def* **by** *simp*  
**moreover have**  $\langle \exists l l'. ?f l l' (\neg (p \vee q)) \text{ at } a \text{ in } \text{mapi-branch } ?f ((ps, a) \# \text{branch}) \rangle$   
**using** *DisN(1)* **at-in-mapi-branch** **by** *fast*  
**then have**  $\langle (\neg (p \vee q)) \text{ at } a \text{ in } (\text{mapi } (?f \text{ (length branch)) } ps, a) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**unfolding** *mapi-branch-def* **using** *p* **by** *simp*  
**ultimately have**  $\langle A \vdash (\text{mapi } (?f \text{ (length branch)) } ps, a) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**using** *DisN'* **by** *fast*  
**then show** *?case*  
**unfolding** *mapi-branch-def* **by** *auto*  
**next**  
**case**  $(\text{DiaP } p \ a \ ps \ \text{branch } i' \ n)$   
**let**  $?f = \langle \text{bridge } k \ j \ xs \rangle$   
**have**  $p: \langle ?f l l' (\diamond p) = (\diamond p) \rangle$  **for**  $l l'$   
**using** *DiaP(3)* *bridge-proper-dia* **by** *fast*  
  
**have**  $\langle \text{branch-nominals } (\text{mapi-branch } ?f ((ps, a) \# \text{branch})) = \text{branch-nominals } ((ps, a) \# \text{branch}) \rangle$   
**using** *DiaP(8-)* *nominals-mapi-branch-bridge* [**where**  $j=j$  **and**  $k=k$  **and**  $\text{branch}=\langle (ps, a) \# \text{branch} \rangle$ ]  
**by** *auto*  
**then have**  $i'$ :  
 $\langle i' \notin A \cup \text{branch-nominals } ((\text{mapi } (?f \text{ (length branch)) } ps, a) \# \text{mapi-branch } ?f \text{ branch}) \rangle$   
**unfolding** *mapi-branch-def* **using** *DiaP(2)* **by** *simp*  
  
**have**  $1: \langle ?f \text{ (length branch)} (1 + \text{length } ps) (\text{@ } i' \ p) = (\text{@ } i' \ p) \rangle$   
**by** *simp*  
**have**  $\langle i' \neq k \rangle$   
**using** *DiaP(2, 8)* **unfolding** *branch-nominals-def* **by** *fastforce*  
**then have**  $2: \langle ?f \text{ (length branch)} (\text{length } ps) (\diamond \text{Nom } i') = (\diamond \text{Nom } i') \rangle$   
**by** *simp*  
  
**have**  $\langle i' \neq j \rangle$   
**using** *DiaP(2, 8)* **unfolding** *branch-nominals-def* **by** *fastforce*  
**moreover have**  $\langle \text{descendants } k \ i \ (((\text{@ } i' \ p) \# (\diamond \text{Nom } i') \# ps, a) \# \text{branch}) \ xs \rangle$   
**using** *DiaP(7)* *descendants-block* [**where**  $ps'=\langle -, - \rangle$ ] **by** *fastforce*  
**ultimately have**  $\langle A \vdash \text{mapi-branch } ?f (((\text{@ } i' \ p) \# (\diamond \text{Nom } i') \# ps, a) \# \text{branch}) \rangle$   
**using** *DiaP(4-)* **by** *blast*  
**then have**  $\langle A \vdash (((\text{@ } i' \ p) \# (\diamond \text{Nom } i') \# \text{mapi } (?f \text{ (length branch)) } ps, a) \# \text{mapi-branch } ?f \text{ branch}) \rangle$   
**unfolding** *mapi-branch-def* **using** *1* **by**  $(\text{simp add: } 2)$   
**moreover have**  $\langle \exists l l'. ?f l l' (\diamond p) \text{ at } a \text{ in } \text{mapi-branch } ?f ((ps, a) \# \text{branch}) \rangle$

```

    using DiaP(1) at-in-mapi-branch by fast
  then have ⟨(◇ p) at a in (mapi (?f (length branch)) ps, a) # mapi-branch ?f
branch⟩
    unfolding mapi-branch-def using p by simp
  ultimately have ⟨A ⊢ (mapi (?f (length branch)) ps, a) # mapi-branch ?f
branch⟩
    using i' DiaP(3) fin DiaP'' by fast
  then show ?case
    unfolding mapi-branch-def by simp
next
case (DiaN p a ps branch i' n)
have p: ⟨bridge k j xs l l' (¬ (◇ p)) = (¬ (◇ p))⟩ for xs l l'
  by simp

obtain rs where rs: ⟨(rs, a) ∈. (ps, a) # branch⟩ ⟨(◇ Nom i') on (rs, a)⟩
  using DiaN(2) by fast
obtain v where v: ⟨(ps, a) # branch⟩ !. v = Some (rs, a)
  using rs(1) rev-nth-mem by fast
obtain v' where v': ⟨rs !. v' = Some (◇ Nom i')⟩
  using rs(2) rev-nth-on by fast

show ?case
proof (cases ⟨(v, v') ∈ xs⟩)
case True
  then have ⟨i' = k⟩
    using DiaN(6) v v' descendants-types by fast

let ?xs = ⟨{(length branch, length ps)} ∪ xs⟩
let ?f = ⟨bridge k j ?xs⟩
let ?branch = ⟨((¬ (@ i' p)) # ps, a) # branch⟩

obtain rs' where
  ⟨(((¬ (@ k p)) # ps, a) # branch) !. v = Some (rs', a)⟩
  ⟨rs' !. v' = Some (◇ Nom i')⟩
  using v v' index-Cons by fast
moreover have ⟨descendants k i (((¬ (@ k p)) # ps, a) # branch) xs)
  using DiaN(6) descendants-block[where ps'=[-]] by fastforce
moreover have ⟨?branch !. length branch = Some ((¬ (@ k p)) # ps, a)⟩
  using ⟨i' = k⟩ by simp
moreover have ⟨((¬ (@ k p)) # ps) !. length ps = Some (¬ (@ k p))⟩
  by simp
ultimately have ⟨descendants k i (((¬ (@ k p)) # ps, a) # branch) ?xs)
  using True ⟨i' = k⟩ Derived[where branch=[- # branch]] by simp

then have ⟨A ⊢ mapi-branch ?f (((¬ (@ k p)) # ps, a) # branch)⟩
  using ⟨i' = k⟩ DiaN(5-) by blast
then have ⟨A ⊢ ((¬ (@ j p)) # mapi (?f (length branch)) ps, a) #
mapi-branch (bridge k j ?xs) branch⟩
  unfolding mapi-branch-def using ⟨i' = k⟩ by simp

```

**moreover have**  $\langle \exists l l'. ?f l l' (\neg (\diamond p)) \text{ at } a \text{ in } \text{mapi-branch } ?f ((ps, a) \# \text{branch}) \rangle$   
**using** *DiaN(1) at-in-mapi-branch by fast*  
**then have**  $\langle (\neg (\diamond p)) \text{ at } a \text{ in } (\text{mapi } (?f (\text{length } \text{branch})) ps, a) \# \text{mapi-branch } ?f \text{branch} \rangle$   
**unfolding** *mapi-branch-def using p[where xs= $\langle ?xs \rangle$ ] by simp*  
**moreover have**  $\langle (\text{mapi } (?f v) rs, a) \in. \text{mapi-branch } ?f ((ps, a) \# \text{branch}) \rangle$   
**using** *v rev-nth-mapi-branch by fast*  
**then have**  $\langle (\text{mapi } (?f v) rs, a) \in$   
 $\text{set } ((\text{mapi } (?f (\text{length } \text{branch})) ps, a) \# \text{mapi-branch } ?f \text{branch}) \rangle$   
**unfolding** *mapi-branch-def by simp*  
**moreover have**  $\langle ?f v v' (\diamond \text{Nom } i') \text{ on } (\text{mapi } (?f v) rs, a) \rangle$   
**using** *v' rev-nth-mapi-block by fast*  
**then have**  $\langle (\diamond \text{Nom } j) \text{ on } (\text{mapi } (?f v) rs, a) \rangle$   
**using** *True  $\langle i' = k \rangle$  by simp*  
**ultimately have**  $\langle A \vdash (\text{mapi } (?f (\text{length } \text{branch})) ps, a) \# \text{mapi-branch } ?f \text{branch} \rangle$   
**by** *(meson DiaN')*  
**then have**  $\langle A \vdash (\text{mapi } (\text{bridge } k j xs (\text{length } \text{branch})) ps, a) \#$   
 $\text{mapi-branch } (\text{bridge } k j xs) \text{branch} \rangle$   
**using** *mapi-branch-head-add-oob[where branch=branch and ps=ps] unfolding*  
**ing** *mapi-branch-def*  
**by** *simp*  
**then show** *?thesis*  
**unfolding** *mapi-branch-def by simp*  
**next**  
**case** *False*  
**let**  $?f = \langle \text{bridge } k j xs \rangle$   
  
**have**  $\langle \text{descendants } k i (((\neg (\text{@ } i' p)) \# ps, a) \# \text{branch}) xs \rangle$   
**using** *DiaN(6) descendants-no-head by fast*  
**then have**  $\langle A \vdash \text{mapi-branch } ?f (((\neg (\text{@ } i' p)) \# ps, a) \# \text{branch}) \rangle$   
**using** *DiaN(5-) by blast*  
**moreover have**  $\langle (\text{length } \text{branch}, \text{length } ps) \notin xs \rangle$   
**using** *DiaN(6) descendants-oob-head by fast*  
**ultimately have**  $\langle A \vdash ((\neg (\text{@ } i' p)) \# \text{mapi } (?f (\text{length } \text{branch})) ps, a) \#$   
 $\text{mapi-branch } ?f \text{branch} \rangle$   
**unfolding** *mapi-branch-def by simp*  
**moreover have**  $\langle \exists l l'. ?f l l' (\neg (\diamond p)) \text{ at } a \text{ in } \text{mapi-branch } ?f ((ps, a) \# \text{branch}) \rangle$   
**using** *DiaN(1) at-in-mapi-branch by fast*  
**then have**  $\langle (\neg (\diamond p)) \text{ at } a \text{ in } (\text{mapi } (?f (\text{length } \text{branch})) ps, a) \# \text{mapi-branch } ?f \text{branch} \rangle$   
**unfolding** *mapi-branch-def using p[where xs= $\langle xs \rangle$ ] by simp*  
**moreover have**  $\langle (\text{mapi } (?f v) rs, a) \in. \text{mapi-branch } ?f ((ps, a) \# \text{branch}) \rangle$   
**using** *v rev-nth-mapi-branch by fast*  
**then have**  $\langle (\text{mapi } (?f v) rs, a) \in$   
 $\text{set } ((\text{mapi } (?f (\text{length } \text{branch})) ps, a) \# \text{mapi-branch } ?f \text{branch}) \rangle$   
**unfolding** *mapi-branch-def by simp*

```

moreover have ⟨ $?f v v' (\diamond \text{Nom } i')$  on  $(\text{mapi } (?f v) rs, a)$ ⟩
  using  $v'$  rev-nth-mapi-block by fast
then have ⟨ $(\diamond \text{Nom } i')$  on  $(\text{mapi } (?f v) rs, a)$ ⟩
  using False by simp
ultimately have ⟨ $A \vdash (\text{mapi } (?f (\text{length } \text{branch})) ps, a) \# \text{mapi-branch } ?f$ 
branch⟩
  by  $(\text{meson } \text{DiaN}')$ 
then show ?thesis
  unfolding mapi-branch-def by simp
qed
next
case  $(\text{SatP } a p b ps \text{branch } n)$ 
let  $?f = (\text{bridge } k j xs)$ 
have  $p: \langle ?f l l' (@ a p) = (@ a p) \rangle$  for  $l l'$ 
  by simp

have ⟨descendants  $k i ((p \# ps, a) \# \text{branch}) xs$ ⟩
  using  $\text{SatP}(5)$  descendants-no-head by fast
then have ⟨ $A \vdash \text{mapi-branch } ?f ((p \# ps, a) \# \text{branch})$ ⟩
  using  $\text{SatP}(4-)$  by blast
moreover have ⟨ $(\text{length } \text{branch}, \text{length } ps) \notin xs$ ⟩
  using  $\text{SatP}(5)$  descendants-oob-head by fast
ultimately have ⟨ $A \vdash (p \# \text{mapi } (?f (\text{length } \text{branch})) ps, a) \# \text{mapi-branch } ?f$ 
branch⟩
  unfolding mapi-branch-def by simp
moreover have ⟨ $\exists l l'. ?f l l' (@ a p)$  at  $b$  in  $\text{mapi-branch } ?f ((ps, a) \# \text{branch})$ ⟩
  using  $\text{SatP}(1)$  at-in-mapi-branch by fast
then have ⟨ $(@ a p)$  at  $b$  in  $(\text{mapi } (?f (\text{length } \text{branch})) ps, a) \# \text{mapi-branch } ?f$ 
branch⟩
  unfolding mapi-branch-def using  $p$  by simp
ultimately have ⟨ $A \vdash (\text{mapi } (?f (\text{length } \text{branch})) ps, a) \# \text{mapi-branch } ?f$ 
branch⟩
  using  $\text{SatP}'$  by fast
then show ?case
  unfolding mapi-branch-def by simp
next
case  $(\text{SatN } a p b ps \text{branch } n)$ 
obtain  $qs$  where  $qs: \langle (qs, b) \in. (ps, a) \# \text{branch} \rangle \langle (\neg (@ a p)) \text{ on } (qs, b) \rangle$ 
  using  $\text{SatN}(1)$  by fast
obtain  $v$  where  $v: \langle ((ps, a) \# \text{branch}) !. v = \text{Some } (qs, b) \rangle$ 
  using  $qs(1)$  rev-nth-mem by fast
obtain  $v'$  where  $v': \langle qs !. v' = \text{Some } (\neg (@ a p)) \rangle$ 
  using  $qs(2)$  rev-nth-on by fast

show ?case
proof  $(\text{cases } \langle (v, v') \in xs \rangle)$ 
  case True
    then have  $\langle a = k \rangle$ 
    using  $\text{SatN}(5)$  v v' descendants-types by fast

```

```

let ?f = ⟨bridge k j xs⟩
let ?branch = ⟨(¬ p) # ps, a⟩ # branch
have p: ⟨?f v v' (¬ (@ k p)) = (¬ (@ j p))⟩
  using True by simp

obtain rs' where
  ⟨?branch !. v = Some (rs', b)⟩
  ⟨rs' !. v' = Some (¬ (@ k p))⟩
  using v v' ⟨a = k⟩ index-Cons by fast
have ⟨descendants k i ?branch xs⟩
  using SatN(5) descendants-no-head by fast
then have ⟨A ⊢ mapi-branch ?f ?branch⟩
  using ⟨a = k⟩ SatN(4-) by blast
moreover have ⟨length branch, length ps⟩ ∉ xs
  using SatN(5) descendants-oob-head by fast
ultimately have ⟨A ⊢ ((¬ p) # mapi (?f (length branch)) ps, a) # mapi-branch
  ?f branch⟩
  unfolding mapi-branch-def using ⟨a = k⟩ by simp
  moreover have ⟨set (((¬ p) # mapi (?f (length branch)) ps, a) # mapi-branch
  ?f branch) ⊆
    set (((¬ p) # mapi (?f (length branch)) ps, a) # ([¬ p], j) # mapi-branch
  ?f branch)⟩
    by auto
  ultimately have *:
    ⟨A ⊢ ((¬ p) # mapi (?f (length branch)) ps, a) # ([¬ p], j) # mapi-branch
  ?f branch⟩
    using inf fin STA-struct by fastforce

have k: ⟨Nom k at j in mapi-branch ?f ((ps, a) # branch)⟩
  using SatN(6) nom-at-in-bridge unfolding mapi-branch-def by fastforce

have ⟨(mapi (?f v) qs, b) ∈. mapi-branch ?f ((ps, a) # branch)⟩
  using v rev-nth-mapi-branch by fast
moreover have ⟨?f v v' (¬ (@ k p)) on (mapi (?f v) qs, b)⟩
  using v' ⟨a = k⟩ rev-nth-mapi-block by fast
then have ⟨(¬ (@ j p)) on (mapi (?f v) qs, b)⟩
  using p by simp
ultimately have satn: ⟨(¬ (@ j p)) at b in mapi-branch ?f ((ps, a) # branch)⟩
  by blast

have ⟨j ∈ branch-nominals (mapi-branch ?f ((ps, a) # branch))⟩
  unfolding branch-nominals-def using k by fastforce
then have ?thesis if ⟨A ⊢ ([], j) # mapi-branch ?f ((ps, a) # branch)⟩
  using that GoTo by fast
  moreover have ⟨(¬ (@ j p)) at b in ([], j) # mapi-branch ?f ((ps, a) #
  branch)⟩
    using satn by auto
  ultimately have ?thesis if ⟨A ⊢ ([¬ p], j) # mapi-branch ?f ((ps, a) #

```

*branch*)  
**using** *that SatN'* **by** *fast*  
**then have** *?thesis* **if**  $\langle A \vdash ([\neg p], j) \# \text{mapi-branch } ?f ((ps, a) \# \text{branch}) \rangle$   
**using** *that SatN'* **by** *fast*  
**then have** *?thesis* **if**  
 $\langle A \vdash ([\neg p], j) \# (\text{mapi } (?f \text{ (length branch)}) ps, a) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**using** *that unfolding mapi-branch-def* **by** *simp*  
**moreover have**  $\langle \text{set } ((\text{mapi } (?f \text{ (length branch)}) ps, a) \# ([\neg p], j) \# \text{mapi-branch } ?f \text{ branch}) \subseteq$   
 $\text{set } (([\neg p], j) \# (\text{mapi } (?f \text{ (length branch)}) ps, a) \# \text{mapi-branch } ?f \text{ branch}) \rangle$   
**by** *auto*  
**ultimately have** *?thesis* **if**  
 $\langle A \vdash (\text{mapi } (?f \text{ (length branch)}) ps, a) \# ([\neg p], j) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**using** *that inf fin STA-struct* **by** *blast*  
**moreover have**  
 $\langle \text{Nom } k \text{ at } j \text{ in } (\text{mapi } (?f \text{ (length branch)}) ps, a) \# ([\neg p], j) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**using** *k unfolding mapi-branch-def* **by** *auto*  
**moreover have**  
 $\langle (\neg p) \text{ at } j \text{ in } (\text{mapi } (?f \text{ (length branch)}) ps, a) \# ([\neg p], j) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**by** *fastforce*  
**ultimately have** *?thesis* **if**  
 $\langle A \vdash ((\neg p) \# \text{mapi } (?f \text{ (length branch)}) ps, a) \# ([\neg p], j) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**using** *that a = k* **by** *(meson Nom' fm.distinct(21) fm.simps(18))*  
**then show** *?thesis*  
**using** *\** **by** *blast*  
**next**  
**case** *False*  
**let** *?f = bridge k j xs*  
  
**have**  $\langle \text{descendants } k \text{ } i \text{ } (((\neg p) \# ps, a) \# \text{branch}) \text{ } xs \rangle$   
**using** *SatN(5) descendants-no-head* **by** *fast*  
**then have**  $\langle A \vdash \text{mapi-branch } (\text{bridge } k \text{ } j \text{ } xs) \text{ } (((\neg p) \# ps, a) \# \text{branch}) \rangle$   
**using** *SatN(4-)* **by** *blast*  
**moreover have**  $\langle \text{length branch, length } ps \notin xs \rangle$   
**using** *SatN(5) descendants-oob-head* **by** *fast*  
**ultimately have**  $\langle A \vdash ((\neg p) \# \text{mapi } (?f \text{ (length branch)}) ps, a) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**unfolding** *mapi-branch-def* **by** *simp*  
**moreover have**  $\langle (\text{mapi } (?f \text{ } v) \text{ } qs, b) \in . \text{mapi-branch } ?f ((ps, a) \# \text{branch}) \rangle$   
**using** *v rev-nth-mapi-branch* **by** *fast*  
**then have**  $\langle (\text{mapi } (?f \text{ } v) \text{ } qs, b) \in$   
 $\text{set } ((\text{mapi } (?f \text{ (length branch)}) ps, a) \# \text{mapi-branch } ?f \text{ branch}) \rangle$   
**unfolding** *mapi-branch-def* **by** *simp*  
**moreover have**  $\langle ?f \text{ } v \text{ } v' (\neg (@ \text{ } a \text{ } p)) \text{ on } (\text{mapi } (?f \text{ } v) \text{ } qs, b) \rangle$   
**using** *v' rev-nth-mapi-block* **by** *fast*  
**then have**  $\langle (\neg (@ \text{ } a \text{ } p)) \text{ on } (\text{mapi } (?f \text{ } v) \text{ } qs, b) \rangle$

```

    using False by simp
    ultimately have ⟨A ⊢ (mapi (?f (length branch)) ps, a) # mapi-branch ?f
branch⟩
    by (meson SatN')
    then show ?thesis
    unfolding mapi-branch-def by simp
qed
next
case (GoTo i' n ps a branch)
let ?f = ⟨bridge k j xs⟩

have ⟨descendants k i (([], i') # (ps, a) # branch) xs⟩
    using GoTo(4) descendants-branch[where extra=⟨[-]⟩] by simp
then have ⟨A ⊢ mapi-branch ?f (([], i') # (ps, a) # branch)⟩
    using GoTo(3, 5-) by auto
then have ⟨A ⊢ ([], i') # mapi-branch ?f ((ps, a) # branch)⟩
    unfolding mapi-branch-def by simp
moreover have
    ⟨branch-nominals (mapi-branch ?f ((ps, a) # branch)) = branch-nominals ((ps,
a) # branch)⟩
    using GoTo(5-) nominals-mapi-branch-bridge[where j=j and k=k and
branch=⟨(ps, a) # branch⟩]
    by auto
then have ⟨i' ∈ branch-nominals (mapi-branch (bridge k j xs) ((ps, a) # branch))⟩
    using GoTo(1) by blast
ultimately show ?case
    using STA.GoTo by fast
next
case (Nom p b ps a branch n)
show ?case
proof (cases ⟨∃j. p = Nom j⟩)
case True
let ?f = ⟨bridge k j xs⟩

have ⟨descendants k i ((p # ps, a) # branch) xs⟩
    using Nom(7) descendants-block[where ps'=⟨[p]⟩] by simp
then have ⟨A ⊢ mapi-branch ?f ((p # ps, a) # branch)⟩
    using Nom(6-) by blast
moreover have ⟨?f (length branch) (length ps) p = p⟩
    using True by auto
ultimately have ⟨A ⊢ (p # mapi (?f (length branch)) ps, a) # mapi-branch
?f branch⟩
    unfolding mapi-branch-def by simp
moreover have ⟨p at b in mapi-branch ?f ((ps, a) # branch)⟩
    using Nom(1) True nom-at-in-bridge by fast
then have ⟨p at b in (mapi (?f (length branch)) ps, a) # mapi-branch ?f
branch⟩
    unfolding mapi-branch-def by simp
moreover have ⟨Nom a at b in mapi-branch ?f ((ps, a) # branch)⟩

```



```

    using Nom(2) True nom-at-in-bridge by fast
    then have ⟨Nom a at b in (mapi (?f (length branch)) ps, a) # mapi-branch
    ?f branch⟩
      unfolding mapi-branch-def by simp
      ultimately have ⟨A ⊢ (mapi (?f (length branch)) ps, a) # mapi-branch ?f
      branch⟩
        by (meson Nom' Nom.hyps(3))
      then show ?thesis
        unfolding mapi-branch-def by simp
next
case False
obtain qs where qs: ⟨(qs, b) ∈. (ps, a) # branch⟩ ⟨p on (qs, b)⟩
  using Nom(1) by blast
obtain v where v: ⟨((ps, a) # branch) !. v = Some (qs, b)⟩
  using qs(1) rev-nth-mem by fast
obtain v' where v': ⟨qs !. v' = Some p⟩
  using qs(2) False rev-nth-on by fast

show ?thesis
proof (cases ⟨(v, v') ∈ xs⟩)
case True
  let ?xs = ⟨{(length branch, length ps)} ∪ xs⟩
  let ?f = ⟨bridge k j ?xs⟩

  let ?p = ⟨bridge' k j p⟩
  have p: ⟨?f v v' p = ?p⟩
    using True by simp

  consider (dia) ⟨p = (◇ Nom k)⟩ | (satn) q where ⟨p = (¬ (@ k q))⟩ | (old)
  ⟨?p = p⟩
  by (meson Nom.prem(1) True descendants-types v v')
  then have A: ⟨∀ i. ?p = Nom i ∨ ?p = (◇ Nom i) ⟶ i ∈ A⟩
    using Nom(3) ⟨j ∈ A⟩ by cases simp-all

obtain qs' where
  ⟨((p # ps, a) # branch) !. v = Some (qs', b)⟩
  ⟨qs' !. v' = Some p⟩
  using v v' index-Cons by fast
moreover have ⟨Nom a at b in (p # ps, a) # branch⟩
  using Nom(2) by fastforce
moreover have ⟨descendants k i ((p # ps, a) # branch) xs⟩
  using Nom(7) descendants-block[where ps' = ⟨p⟩] by simp
moreover have
  ⟨((p # ps, a) # branch) !. length branch = Some (p # ps, a)⟩
  ⟨(p # ps) !. length ps = Some p⟩
  by simp-all
ultimately have ⟨descendants k i ((p # ps, a) # branch) ?xs⟩
  using True Copied by fast
then have ⟨A ⊢ mapi-branch ?f ((p # ps, a) # branch)⟩

```

**using** *Nom(6-)* **by** *blast*  
**then have**  $\langle A \vdash (?p \# \text{mapi } (?f \text{ (length branch)) } ps, a) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**unfolding** *mapi-branch-def* **by** *simp*

**moreover have**  $\langle (\text{mapi } (?f v) qs, b) \in. \text{mapi-branch } ?f ((ps, a) \# \text{branch}) \rangle$   
**using** *v rev-nth-mapi-branch* **by** *fast*  
**then have**  $\langle (\text{mapi } (?f v) qs, b) \in. (\text{mapi } (?f \text{ (length branch)) } ps, a) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**unfolding** *mapi-branch-def* **by** *simp*  
**moreover have**  $\langle ?f v v' p \text{ on } (\text{mapi } (?f v) qs, b) \rangle$   
**using** *v v' rev-nth-mapi-block* **by** *fast*  
**then have**  $\langle ?p \text{ on } (\text{mapi } (?f v) qs, b) \rangle$   
**using** *p* **by** *simp*

**moreover have**  $\langle \text{Nom } a \text{ at } b \text{ in } \text{mapi-branch } ?f ((ps, a) \# \text{branch}) \rangle$   
**using** *Nom(2) nom-at-in-bridge* **by** *fast*  
**then have**  $\langle \text{Nom } a \text{ at } b \text{ in } (\text{mapi } (?f \text{ (length branch)) } ps, a) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**unfolding** *mapi-branch-def* **by** *simp*  
**ultimately have**  $\langle A \vdash (\text{mapi } (?f \text{ (length branch)) } ps, a) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**using** *A* **by** (*meson* *Nom' Nom(3)*)  
**then have**  $\langle A \vdash (\text{mapi } (\text{bridge } k j xs \text{ (length branch)) } ps, a) \# \text{mapi-branch } (\text{bridge } k j xs) \text{ branch} \rangle$   
**using** *mapi-branch-head-add-oob* [**where** *branch=branch* **and** *ps=ps*]  
**unfolding** *mapi-branch-def* **by** *simp*  
**then show** *?thesis*  
**unfolding** *mapi-branch-def* **by** *simp*

**next**  
**case** *False*  
**let** *?f = (bridge k j xs)*

**have**  $\langle \text{descendants } k i ((p \# ps, a) \# \text{branch}) xs \rangle$   
**using** *Nom(7) descendants-no-head* **by** *fast*  
**then have**  $\langle A \vdash \text{mapi-branch } ?f ((p \# ps, a) \# \text{branch}) \rangle$   
**using** *Nom(6-)* **by** *blast*  
**moreover have**  $\langle (\text{length branch}, \text{length } ps) \notin xs \rangle$   
**using** *Nom(7) descendants-oob-head* **by** *fast*  
**ultimately have**  $\langle A \vdash (p \# \text{mapi } (?f \text{ (length branch)) } ps, a) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**unfolding** *mapi-branch-def* **by** *simp*

**moreover have**  $\langle (\text{mapi } (?f v) qs, b) \in. \text{mapi-branch } ?f ((ps, a) \# \text{branch}) \rangle$   
**using** *v rev-nth-mapi-branch* **by** *fast*  
**then have**  $\langle (\text{mapi } (?f v) qs, b) \in. (\text{mapi } (?f \text{ (length branch)) } ps, a) \# \text{mapi-branch } ?f \text{ branch} \rangle$   
**unfolding** *mapi-branch-def* **by** *simp*  
**moreover have**  $\langle ?f v v' p \text{ on } (\text{mapi } (?f v) qs, b) \rangle$

**using**  $v v'$  *rev-nth-mapi-block* **by** *fast*  
**then have**  $\langle p \text{ on } (\text{mapi } (?f v) \text{ } qs, b) \rangle$   
**using** *False* **by** *simp*  
**moreover have**  $\langle \text{Nom } a \text{ at } b \text{ in } \text{mapi-branch } ?f ((ps, a) \# \text{branch}) \rangle$   
**using** *Nom(2)* *nom-at-in-bridge* **by** *fast*  
**then have**  $\langle \text{Nom } a \text{ at } b \text{ in } (\text{mapi } (?f (\text{length } \text{branch})) ps, a) \# \text{mapi-branch } ?f \text{branch} \rangle$   
**unfolding** *mapi-branch-def* **by** *simp*  
**ultimately have**  $\langle A \vdash (\text{mapi } (?f (\text{length } \text{branch})) ps, a) \# \text{mapi-branch } ?f \text{branch} \rangle$   
**by** (*meson* *Nom'* *Nom(3)*)  
**then show** *?thesis*  
**unfolding** *mapi-branch-def* **by** *simp*  
**qed**  
**qed**  
**qed**

**lemma** *STA-bridge*:

**fixes**  $i :: 'b$   
**assumes**  
 $\langle \text{inf}: \langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle \text{ and} \rangle$   
 $\langle A \vdash \text{branch} \rangle \langle \text{descendants } k \text{ } i \text{ } \text{branch } xs \rangle$   
 $\langle \text{Nom } k \text{ at } j \text{ in } \text{branch} \rangle$   
 $\langle \text{finite } A \rangle \langle j \in A \rangle$   
**shows**  $\langle A \vdash \text{mapi-branch } (\text{bridge } k \text{ } j \text{ } xs) \text{branch} \rangle$   
**proof** –  
**have**  $\langle A \vdash ([], j) \# \text{branch} \rangle$   
**using** *assms(2, 5–6)* *inf STA-struct*[**where**  $\text{branch}' = \langle ([], j) \# \text{branch} \rangle$ ] **by** *auto*  
**moreover have**  $\langle \text{descendants } k \text{ } i \text{ } ([], j) \# \text{branch} \text{ } xs \rangle$   
**using** *assms(3)* *descendants-branch*[**where**  $\text{extra} = \langle [-] \rangle$ ] **by** *fastforce*  
**ultimately have**  $\langle A \vdash \text{mapi-branch } (\text{bridge } k \text{ } j \text{ } xs) ([], j) \# \text{branch} \rangle$   
**using** *STA-bridge'* *inf* *assms(3–)* **by** *fast*  
**then have**  $*$ :  $\langle A \vdash ([], j) \# \text{mapi-branch } (\text{bridge } k \text{ } j \text{ } xs) \text{branch} \rangle$   
**unfolding** *mapi-branch-def* **by** *simp*  
**have**  $\langle \text{branch-nominals } (\text{mapi-branch } (\text{bridge } k \text{ } j \text{ } xs) \text{branch}) = \text{branch-nominals } \text{branch} \rangle$   
**using** *nominals-mapi-branch-bridge* *assms(4–)* **by** *fast*  
**moreover have**  $\langle j \in \text{branch-nominals } \text{branch} \rangle$   
**using** *assms(4)* **unfolding** *branch-nominals-def* **by** *fastforce*  
**ultimately have**  $\langle j \in \text{branch-nominals } (\text{mapi-branch } (\text{bridge } k \text{ } j \text{ } xs) \text{branch}) \rangle$   
**by** *simp*  
**then show** *?thesis*  
**using**  $*$  *GoTo* **by** *fast*  
**qed**

## 10.4 Derivation

**theorem** *Bridge*:

**fixes**  $i :: 'b$   
**assumes**  $inf: \langle infinite (UNIV :: 'b set) \rangle$  **and**  $fin: \langle finite A \rangle$  **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**  $\langle A \vdash (ps, i) \# branch \rangle$   
**proof** –  
**let**  $?xs = \langle \{(length\ branch, length\ ps)\} \rangle$   
  
**have**  $\langle descendants\ k\ i\ (((\Diamond\ Nom\ k) \# ps, i) \# branch)\ ?xs \rangle$   
**using** *Initial by force*  
**moreover have**  $\langle Nom\ k\ at\ j\ in\ ((\Diamond\ Nom\ k) \# ps, i) \# branch \rangle$   
**using** *assms(4) by fastforce*  
**ultimately have**  $\langle A \vdash mapi\text{-}branch\ (bridge\ k\ j\ ?xs)\ (((\Diamond\ Nom\ k) \# ps, i) \# branch) \rangle$   
**using** *STA-bridge inf fin assms(3, 6) by fast*  
**then have**  $\langle A \vdash ((\Diamond\ Nom\ j) \# mapi\ (bridge\ k\ j\ ?xs\ (length\ branch))\ ps, i) \# mapi\text{-}branch\ (bridge\ k\ j\ ?xs)\ branch \rangle$   
**unfolding** *mapi-branch-def by simp*  
**moreover have**  $\langle mapi\text{-}branch\ (bridge\ k\ j\ \{(length\ branch, length\ ps)\})\ branch = mapi\text{-}branch\ (bridge\ k\ j\ \{\})\ branch \rangle$   
**using** *mapi-branch-add-oob[where xs={}] by fastforce*  
**moreover have**  $\langle mapi\ (bridge\ k\ j\ ?xs\ (length\ branch))\ ps = mapi\ (bridge\ k\ j\ \{\})\ (length\ branch)\ ps \rangle$   
**using** *mapi-block-add-oob[where xs={}] and ps=ps by simp*  
**ultimately have**  $\langle A \vdash ((\Diamond\ Nom\ j) \# ps, i) \# branch \rangle$   
**using** *mapi-block-id[where ps=ps] mapi-branch-id[where branch=branch] by simp*  
**then show** *?thesis*  
**using** *Dup assms(5) by (metis new-def)*  
**qed**

## 11 Completeness

### 11.1 Hintikka

**abbreviation**  $at\text{-}in\text{-}set :: \langle ('a, 'b)\ fm \Rightarrow 'b \Rightarrow ('a, 'b)\ block\ set \Rightarrow bool \rangle$  **where**  
 $\langle at\text{-}in\text{-}set\ p\ a\ S \equiv \exists ps. (ps, a) \in S \wedge p\ on\ (ps, a) \rangle$

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

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**  $A :: \langle 'b\ set \rangle$  **and**  $H :: \langle ('a, 'b)\ block\ set \rangle$  **assumes**  
 $ProP: \langle Nom\ j\ at\ i\ in'\ H \Longrightarrow Pro\ x\ at\ j\ in'\ H \Longrightarrow \neg (\neg Pro\ x)\ at\ i\ in'\ H \rangle$  **and**  
 $NomP: \langle Nom\ a\ at\ i\ in'\ H \Longrightarrow \neg (\neg Nom\ a)\ at\ i\ in'\ H \rangle$  **and**  
 $NegN: \langle (\neg \neg p)\ at\ i\ in'\ H \Longrightarrow p\ at\ i\ in'\ H \rangle$  **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* ::  $\langle ('a, 'b) \text{ block set} \Rightarrow 'b \Rightarrow 'b \Rightarrow \text{bool} \rangle$  **where**  
 $\langle \text{hequiv } H \ i \ j \equiv \text{Nom } j \text{ at } i \text{ in}' H \rangle$

**abbreviation** *hequiv-rel* ::  $\langle 'b \text{ set} \Rightarrow ('a, 'b) \text{ block set} \Rightarrow ('b \times 'b) \text{ set} \rangle$  **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* ::  $\langle ('a, 'b) \text{ block set} \Rightarrow 'b \text{ set} \rangle$  **where**  
 $\langle \text{names } H \equiv \{i \mid \exists ps \ i. (ps, i) \in H\} \rangle$

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

**lemma** *hequiv-refl'*:  $\langle (ps, i) \in H \implies \text{hequiv } H \ i \ i \rangle$   
**using** *hequiv-refl unfolding names-def* **by** *fastforce*

**lemma** *hequiv-sym'*:  
**assumes**  $\langle \text{Hintikka } A \ H \rangle \langle i \in A \rangle \langle \text{hequiv } H \ i \ j \rangle$   
**shows**  $\langle \text{hequiv } H \ j \ i \rangle$

**proof** –

**have**  $\langle i \in A \longrightarrow \text{Nom } i \text{ at } i \text{ in}' H \longrightarrow \text{Nom } j \text{ at } i \text{ in}' H \longrightarrow \text{Nom } i \text{ at } j \text{ in}' H \rangle$   
**for**  $i \ j$

**using** *assms(1) Hintikka.Nom* **by** *fast*

**then show** *?thesis*

**using** *assms(2–) unfolding hequiv-def* **by** *auto*

**qed**

**lemma** *hequiv-sym*:  $\langle \text{Hintikka } A \ H \implies i \in A \implies j \in A \implies \text{hequiv } H \ i \ j \longleftrightarrow$   
 $\text{hequiv } H \ j \ i \rangle$   
**by**  $(\text{meson } \text{hequiv-sym}' )$

**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** –

**have**  $\langle \text{hequiv } H \ j \ i \rangle$   
**by**  $(\text{meson } \text{assms}(1-2, 4) \ \text{hequiv-sym})$   
**moreover have**  $\langle k \in A \longrightarrow \text{Nom } k \text{ at } j \text{ in } H \longrightarrow \text{Nom } i \text{ at } j \text{ in } H \longrightarrow \text{Nom } k$   
 $\text{at } i \text{ in } H \rangle$  **for**  $i \ k \ j$   
**using**  $\text{assms}(1) \ \text{Hintikka.Nom}$  **by** *fast*  
**ultimately show** *?thesis*  
**using**  $\text{assms}(3-)$  **unfolding** *hequiv-def* **by** *blast*  
**qed**

**lemma** *hequiv-names*:  $\langle \text{hequiv } H \ i \ j \implies i \in \text{names } H \rangle$   
**unfolding** *hequiv-def names-def* **by** *blast*

**lemma** *hequiv-names-rel*:  
**assumes**  $\langle \text{Hintikka } A \ H \rangle$   
**shows**  $\langle \text{hequiv-rel } A \ H \subseteq \text{names } H \times \text{names } H \rangle$   
**using**  $\text{assms}$  *hequiv-names hequiv-sym* **by** *fast*

**lemma** *hequiv-refl-rel*:  
**assumes**  $\langle \text{Hintikka } A \ H \rangle$   
**shows**  $\langle \text{refl-on } (\text{names } H \cap A) \ (\text{hequiv-rel } A \ H) \rangle$   
**unfolding** *refl-on-def* **using**  $\text{assms}$  *hequiv-refl hequiv-names-rel* **by** *fast*

**lemma** *hequiv-sym-rel*:  $\langle \text{Hintikka } A \ H \implies \text{sym } (\text{hequiv-rel } A \ H) \rangle$   
**unfolding** *sym-def* **using** *hequiv-sym* **by** *fast*

**lemma** *hequiv-trans-rel*:  $\langle \text{Hintikka } B \ A \implies \text{trans } (\text{hequiv-rel } B \ A) \rangle$   
**unfolding** *trans-def* **using** *hequiv-trans* **by** *fast*

**lemma** *hequiv-rel*:  $\langle \text{Hintikka } A \ H \implies \text{equiv } (\text{names } H \cap A) \ (\text{hequiv-rel } A \ H) \rangle$   
**using** *hequiv-refl-rel hequiv-sym-rel hequiv-trans-rel* **by**  $(\text{rule } \text{equivI})$

**lemma** *nominal-in-names*:  
**assumes**  $\langle \text{Hintikka } A \ H \rangle \ \langle \exists \text{ block} \in H. \ i \in \text{block-nominals } \text{block} \rangle$   
**shows**  $\langle i \in \text{names } H \rangle$   
**using**  $\text{assms}$  *Hintikka.GoTo* **unfolding** *names-def* **by** *fastforce*

### 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  $\diamond \text{Nom } j$  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**  $assign :: \langle 'b \text{ set} \Rightarrow ('a, 'b) \text{ block set} \Rightarrow 'b \Rightarrow 'b \text{ set} \rangle$  **where**  
 $\langle assign \ A \ H \ i \equiv \text{if } \exists a. a \in A \wedge \text{Nom } a \text{ at } i \text{ in } H$   
 $\text{ then } \text{proj } (\text{hequiv-rel } A \ H) \ (\text{SOME } a. a \in A \wedge \text{Nom } a \text{ at } i \text{ in } H)$   
 $\text{ else } \{i\} \rangle$

**definition**  $reach :: \langle 'b \text{ set} \Rightarrow ('a, 'b) \text{ block set} \Rightarrow 'b \text{ set} \Rightarrow 'b \text{ set set} \rangle$  **where**  
 $\langle reach \ A \ H \ is \equiv \{assign \ A \ H \ j \mid i \ j. i \in is \wedge (\Diamond \text{Nom } j) \text{ at } i \text{ in } H\} \rangle$

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

**lemma**  $ex\text{-assignment}$ :

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

**proof**  $(\text{cases } \langle \exists b. b \in A \wedge \text{Nom } b \text{ at } i \text{ in } H \rangle)$

**case**  $True$

**let**  $?b = \langle \text{SOME } b. b \in A \wedge \text{Nom } b \text{ at } i \text{ in } H \rangle$

**have**  $*$ :  $\langle ?b \in A \wedge \text{Nom } ?b \text{ at } i \text{ in } H \rangle$

**using**  $someI\text{-ex } True$  .

**moreover from this have**  $\langle \text{hequiv } H \ ?b \ ?b \rangle$

**using**  $assms \text{ block-nominals nominal-in-names hequiv-refl}$

**by**  $(metis \ (no\text{-types, lifting}) \ \text{nominals.simps}(2) \ \text{singleton}I)$

**ultimately show**  $?thesis$

**unfolding**  $assign\text{-def } proj\text{-def}$  **by**  $auto$

**next**

**case**  $False$

**then show**  $?thesis$

**unfolding**  $assign\text{-def}$  **by**  $auto$

**qed**

**lemma**  $ur\text{-closure}$ :

**assumes**  $\langle Hintikka \ A \ H \rangle \langle p \text{ at } i \text{ in } H \rangle \langle \forall a. p = \text{Nom } a \vee p = (\Diamond \text{Nom } a) \longrightarrow a \in A \rangle$

**shows**  $\langle \forall a \in assign \ A \ H \ i. p \text{ at } a \text{ in } H \rangle$

**proof**  $(\text{cases } \langle \exists b. b \in A \wedge \text{Nom } b \text{ at } i \text{ in } H \rangle)$

**case**  $True$

**let**  $?b = \langle \text{SOME } b. b \in A \wedge \text{Nom } b \text{ at } i \text{ in } H \rangle$

**have**  $*$ :  $\langle ?b \in A \wedge \text{Nom } ?b \text{ at } i \text{ in } H \rangle$

**using**  $someI\text{-ex } True$  .

**then have**  $\langle p \text{ at } ?b \text{ in } H \rangle$

**using**  $assms$  **by**  $(meson \ Hintikka.Nom)$

**then have**  $\langle p \text{ at } a \text{ in } H \rangle$  **if**  $\langle \text{hequiv } H \ ?b \ a \rangle$  **for**  $a$

**using**  $that \ assms(1, 3)$  **unfolding**  $hequiv\text{-def}$  **by**  $(meson \ Hintikka.Nom)$

**moreover have**  $\langle assign \ A \ H \ i = \text{proj } (\text{hequiv-rel } A \ H) \ ?b \rangle$

**unfolding**  $assign\text{-def}$  **using**  $True$  **by**  $simp$

**ultimately show**  $?thesis$

**unfolding**  $proj\text{-def}$  **by**  $blast$

**next**

**case** *False*  
**then show** *?thesis*  
    **unfolding** *assign-def* **using** *assms* **by** *auto*  
**qed**

**lemma** *ur-closure'*:  
    **assumes**  $\langle \text{Hintikka } A \ H \rangle \langle p \text{ at } i \text{ in}' H \rangle \langle \forall a. p = \text{Nom } a \vee p = (\diamond \text{Nom } a) \longrightarrow a \in A \rangle$   
    **shows**  $\langle \exists a \in \text{assign } A \ H \ i. p \text{ at } a \text{ in}' H \rangle$   
**proof** –  
    **obtain** *a* **where**  $\langle a \in \text{assign } A \ H \ i \rangle$   
    **using** *assms(1)* *ex-assignment* **by** *fast*  
    **then show** *?thesis*  
    **using** *assms ur-closure[where i=i]* **by** *blast*  
**qed**

**lemma** *mem-hequiv-rel*:  $\langle a \in \text{proj } (\text{hequiv-rel } A \ H) \ b \implies a \in A \rangle$   
    **unfolding** *proj-def* **by** *blast*

**lemma** *hequiv-proj*:  
    **assumes**  $\langle \text{Hintikka } A \ H \rangle$   
     $\langle \text{Nom } a \text{ at } i \text{ in}' H \rangle \langle a \in A \rangle \langle \text{Nom } b \text{ at } i \text{ in}' H \rangle \langle b \in A \rangle$   
    **shows**  $\langle \text{proj } (\text{hequiv-rel } A \ H) \ a = \text{proj } (\text{hequiv-rel } A \ H) \ b \rangle$   
**proof** –  
    **have**  $\langle \text{equiv } (\text{names } H \cap A) \ (\text{hequiv-rel } A \ H) \rangle$   
    **using** *assms(1)* *hequiv-rel* **by** *fast*  
    **moreover have**  $\langle \{a, b\} \subseteq \text{names } H \cap A \rangle$   
    **using** *assms(1–5)* *nominal-in-names* **by** *fastforce*  
    **moreover have**  $\langle \text{Nom } b \text{ at } a \text{ in}' H \rangle$   
    **using** *assms(1–2, 4–5)* *Hintikka.Nom* **by** *fast*  
    **then have**  $\langle \text{hequiv } H \ a \ b \rangle$   
    **unfolding** *hequiv-def* **by** *simp*  
    **ultimately show** *?thesis*  
    **by** (*simp add: proj-iff*)  
**qed**

**lemma** *hequiv-proj-opening*:  
    **assumes**  $\langle \text{Hintikka } A \ H \rangle \langle \text{Nom } a \text{ at } i \text{ in}' H \rangle \langle a \in A \rangle \langle i \in A \rangle$   
    **shows**  $\langle \text{proj } (\text{hequiv-rel } A \ H) \ a = \text{proj } (\text{hequiv-rel } A \ H) \ i \rangle$   
    **using** *hequiv-proj assms* **by** *fastforce*

**lemma** *assign-proj-refl*:  
    **assumes**  $\langle \text{Hintikka } A \ H \rangle \langle \text{Nom } i \text{ at } i \text{ in}' H \rangle \langle i \in A \rangle$   
    **shows**  $\langle \text{assign } A \ H \ i = \text{proj } (\text{hequiv-rel } A \ H) \ i \rangle$   
**proof** –  
    **let** *?a* =  $\langle \text{SOME } a. a \in A \wedge \text{Nom } a \text{ at } i \text{ in}' H \rangle$   
    **have**  $\langle \exists a. a \in A \wedge \text{Nom } a \text{ at } i \text{ in}' H \rangle$   
    **using** *assms(2–3)* **by** *fast*  
    **with** *someI-ex* **have**  $\ast: \langle ?a \in A \wedge \text{Nom } ?a \text{ at } i \text{ in}' H \rangle .$



```

then have ⟨assign A H i = proj (hequiv-rel A H) ?a⟩
  unfolding assign-def by auto
then show ?thesis
  unfolding assign-def
  using hequiv-proj * assms by fast
qed

lemma assign-named:
  assumes ⟨Hintikka A H⟩ ⟨i ∈ proj (hequiv-rel A H) a⟩
  shows ⟨i ∈ names H⟩
  using assms unfolding proj-def by simp (meson hequiv-names hequiv-sym')

lemma assign-unique:
  assumes ⟨Hintikka A H⟩ ⟨a ∈ assign A H i⟩
  shows ⟨assign A H a = assign A H i⟩
proof (cases ⟨∃ b. b ∈ A ∧ Nom b at i in' H⟩)
case True
let ?b = ⟨SOME b. b ∈ A ∧ Nom b at i in' H⟩
have *: ⟨?b ∈ A ∧ Nom ?b at i in' H⟩
  using someI-ex True .

have **: ⟨assign A H i = proj (hequiv-rel A H) ?b⟩
  unfolding assign-def using True by simp
moreover from this have ⟨Nom a at a in' H⟩
  using assms assign-named unfolding names-def by fastforce
ultimately have ⟨assign A H a = proj (hequiv-rel A H) a⟩
  using assms assign-proj-refl mem-hequiv-rel by fast
with ** show ?thesis
  unfolding proj-def using assms
  by simp (meson hequiv-sym' hequiv-trans)
next
case False
then have ⟨assign A H i = {i}⟩
  unfolding assign-def by auto
then have ⟨a = i⟩
  using assms(2) by simp
then show ?thesis
  by simp
qed

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
  using assms Hintikka.ProP ur-closure by fastforce

lemma Hintikka-model:
  assumes ⟨Hintikka A H⟩

```

**shows**  
 $\langle p \text{ at } i \text{ in}' H \implies \text{nominals } p \subseteq A \implies$   
 $\text{Model } (\text{reach } A H) (\text{val } H), \text{ assign } A H, \text{ assign } A H i \models p \rangle$   
 $\langle (\neg p) \text{ at } i \text{ in}' H \implies \text{nominals } p \subseteq A \implies$   
 $\neg \text{Model } (\text{reach } A H) (\text{val } H), \text{ assign } A H, \text{ assign } A H i \models p \rangle$

**proof** (*induct p arbitrary: i*)  
**fix**  $i$   
**case** ( $\text{Pro } x$ )  
**assume**  $\langle \text{Pro } x \text{ at } i \text{ in}' H \rangle$   
**then show**  $\langle \text{Model } (\text{reach } A H) (\text{val } H), \text{ assign } A H, \text{ assign } A H i \models \text{Pro } x \rangle$   
**using**  $\text{assms}(1)$  *wr-closure'* **unfolding** *val-def* **by** *fastforce*

**next**  
**fix**  $i$   
**case** ( $\text{Pro } x$ )  
**assume**  $\langle (\neg \text{Pro } x) \text{ at } i \text{ in}' H \rangle$   
**then have**  $\langle \nexists a. a \in \text{assign } A H i \wedge \text{Pro } x \text{ at } a \text{ in}' H \rangle$   
**using**  $\text{assms}(1)$  *assign-val* **unfolding** *names-def* **by** *fast*  
**then have**  $\langle \neg \text{val } H (\text{assign } A H i) x \rangle$   
**unfolding** *proj-def val-def hequiv-def* **by** *simp*  
**then show**  $\langle \neg \text{Model } (\text{reach } A H) (\text{val } H), \text{ assign } A H, \text{ assign } A H i \models \text{Pro } x \rangle$   
**by** *simp*

**next**  
**fix**  $i$   
**case** ( $\text{Nom } a$ )  
**assume**  $\ast: \langle \text{Nom } a \text{ at } i \text{ in}' H \rangle \langle \text{nominals } (\text{Nom } a) \subseteq A \rangle$

**let**  $?b = \langle \text{SOME } b. b \in A \wedge \text{Nom } b \text{ at } i \text{ in}' H \rangle$   
**let**  $?c = \langle \text{SOME } b. b \in A \wedge \text{Nom } b \text{ at } a \text{ in}' H \rangle$

**have**  $\langle a \in A \rangle$   
**using**  $\ast(2)$  **by** *simp*  
**then have**  $\langle \exists b. b \in A \wedge \text{Nom } b \text{ at } i \text{ in}' H \rangle$   
**using**  $\ast$  **by** *fast*  
**with** *someI-ex* **have**  $b: \langle ?b \in A \wedge \text{Nom } ?b \text{ at } i \text{ in}' H \rangle .$   
**then have**  $\langle \text{assign } A H i = \text{proj } (\text{hequiv-rel } A H) ?b \rangle$   
**unfolding** *assign-def* **by** *auto*  
**also have**  $\langle \text{proj } (\text{hequiv-rel } A H) ?b = \text{proj } (\text{hequiv-rel } A H) a \rangle$   
**using** *hequiv-proj assms(1)*  $b \ast \langle a \in A \rangle$  **by** *fast*

**also have**  $\langle \text{Nom } a \text{ at } a \text{ in}' H \rangle$   
**using**  $\ast \langle a \in A \rangle$  *assms(1) Hintikka.Nom* **by** *fast*  
**then have**  $\langle \exists c. c \in A \wedge \text{Nom } c \text{ at } a \text{ in}' H \rangle$   
**using**  $\langle a \in A \rangle$  **by** *blast*  
**with** *someI-ex* **have**  $c: \langle ?c \in A \wedge \text{Nom } ?c \text{ at } a \text{ in}' H \rangle .$   
**then have**  $\langle \text{assign } A H a = \text{proj } (\text{hequiv-rel } A H) ?c \rangle$   
**unfolding** *assign-def* **by** *auto*  
**then have**  $\langle \text{proj } (\text{hequiv-rel } A H) a = \text{assign } A H a \rangle$   
**using** *hequiv-proj-opening assms(1)*  $\langle a \in A \rangle c$  **by** *fast*

**finally have**  $\langle \text{assign } A \ H \ i = \text{assign } A \ H \ a \rangle$  .  
**then show**  $\langle \text{Model } (\text{reach } A \ H) \ (\text{val } H), \text{assign } A \ H, \text{assign } A \ H \ i \models \text{Nom } a \rangle$   
**by simp**  
**next**  
**fix**  $i$   
**case**  $(\text{Nom } a)$   
**assume**  $*$ :  $\langle (\neg \text{Nom } a) \text{ at } i \text{ in}' H \rangle \langle \text{nominals } (\text{Nom } a) \subseteq A \rangle$   
**then have**  $\langle a \in A \rangle$   
**by simp**  
  
**have**  $\langle \text{hequiv } H \ a \ a \rangle$   
**using**  $\text{hequiv-refl} * \text{nominal-in-names assms}(1)$  **by fastforce**  
**obtain**  $j$  **where**  $j$ :  $\langle j \in \text{assign } A \ H \ i \rangle \langle (\neg \text{Nom } a) \text{ at } j \text{ in}' H \rangle$   
**using**  $\text{ur-closure}' \text{ assms}(1) *$  **by fastforce**  
**then have**  $\langle \neg \text{Nom } a \text{ at } j \text{ in}' H \rangle$   
**using**  $\text{assms}(1) \text{ Hintikka.NomP}$  **by fast**  
  
**moreover have**  $\langle \forall b \in \text{assign } A \ H \ a. \text{Nom } a \text{ at } b \text{ in}' H \rangle$   
**using**  $\text{assms } \langle a \in A \rangle \langle \text{hequiv } H \ a \ a \rangle \text{ur-closure}$  **unfolding**  $\text{hequiv-def}$  **by fast**  
**ultimately have**  $\langle \text{assign } A \ H \ a \neq \text{assign } A \ H \ i \rangle$   
**using**  $j$  **by blast**  
**then show**  $\langle \neg \text{Model } (\text{reach } A \ H) \ (\text{val } H), \text{assign } A \ H, \text{assign } A \ H \ i \models \text{Nom } a \rangle$   
**by simp**  
**next**  
**fix**  $i$   
**case**  $(\text{Neg } p)$   
**moreover assume**  $\langle (\neg p) \text{ at } i \text{ in}' H \rangle \langle \text{nominals } (\neg p) \subseteq A \rangle$   
**ultimately show**  $\langle \text{Model } (\text{reach } A \ H) \ (\text{val } H), \text{assign } A \ H, \text{assign } A \ H \ i \models \neg$   
 $p \rangle$   
**by simp**  
**next**  
**fix**  $i$   
**case**  $(\text{Neg } p)$   
**moreover assume**  $*$ :  $\langle (\neg \neg p) \text{ at } i \text{ in}' H \rangle$   
**then have**  $\langle p \text{ at } i \text{ in}' H \rangle$   
**using**  $\text{assms}(1) \text{ Hintikka.NegN}$  **by fast**  
**moreover assume**  $\langle \text{nominals } (\neg p) \subseteq A \rangle$   
**moreover from**  $\text{this} *$  **have**  $\langle \forall a. p = (\diamond \text{Nom } a) \longrightarrow a \in A \rangle$   
**by auto**  
**ultimately show**  $\langle \neg \text{Model } (\text{reach } A \ H) \ (\text{val } H), \text{assign } A \ H, \text{assign } A \ H \ i \models$   
 $\neg p \rangle$   
**using**  $\text{assms}(1)$  **by auto**  
**next**  
**fix**  $i$   
**case**  $(\text{Dis } p \ q)$   
**moreover assume**  $*$ :  $\langle (p \vee q) \text{ at } i \text{ in}' H \rangle$   
**then have**  $\langle p \text{ at } i \text{ in}' H \vee q \text{ at } i \text{ in}' H \rangle$   
**using**  $\text{assms}(1) \text{ Hintikka.DisP}$  **by fast**  
**moreover assume**  $\langle \text{nominals } (p \vee q) \subseteq A \rangle$

```

moreover from this * have  $\langle \forall a. p = (\diamond \text{Nom } a) \longrightarrow a \in A \rangle \langle \forall a. q = (\diamond \text{Nom } a) \longrightarrow a \in A \rangle$ 
  by auto
ultimately show  $\langle \text{Model } (\text{reach } A \ H) \ (\text{val } H), \ \text{assign } A \ H, \ \text{assign } A \ H \ i \models (p \vee q) \rangle$ 
  by simpmetis
next
  fix i
  case  $(\text{Dis } p \ q)$ 
  moreover assume *:  $\langle (\neg (p \vee q)) \ \text{at } i \ \text{in}' \ H \rangle$ 
  then have  $\langle (\neg p) \ \text{at } i \ \text{in}' \ H \rangle \langle (\neg q) \ \text{at } i \ \text{in}' \ H \rangle$ 
    using assms(1) Hintikka.DisN by fast+
  moreover assume  $\langle \text{nominals } (p \vee q) \subseteq A \rangle$ 
  moreover from this * have  $\langle \forall a. p = (\diamond \text{Nom } a) \longrightarrow a \in A \rangle \langle \forall a. q = (\diamond \text{Nom } a) \longrightarrow a \in A \rangle$ 
    by auto
  ultimately show  $\langle \neg \text{Model } (\text{reach } A \ H) \ (\text{val } H), \ \text{assign } A \ H, \ \text{assign } A \ H \ i \models (p \vee q) \rangle$ 
    by auto
  next
  fix i
  case  $(\text{Dia } p)$ 
  assume *:  $\langle (\diamond p) \ \text{at } i \ \text{in}' \ H \rangle \langle \text{nominals } (\diamond p) \subseteq A \rangle$ 
  with * have p:  $\langle \forall a. p = (\diamond \text{Nom } a) \longrightarrow a \in A \rangle$ 
    by auto

show  $\langle \text{Model } (\text{reach } A \ H) \ (\text{val } H), \ \text{assign } A \ H, \ \text{assign } A \ H \ i \models \diamond p \rangle$ 
proof  $(\text{cases } \langle \exists j. p = \text{Nom } j \rangle)$ 
  case True
    then obtain j where j:  $\langle p = \text{Nom } j \rangle \langle j \in A \rangle$ 
      using *(2) by auto
    then obtain a where a:  $\langle a \in \text{assign } A \ H \ i \rangle \langle (\diamond \text{Nom } j) \ \text{at } a \ \text{in}' \ H \rangle$ 
      using ur-closure' assms(1)  $\langle (\diamond p) \ \text{at } i \ \text{in}' \ H \rangle$  by fast

    from j have  $\langle (\diamond \text{Nom } j) \ \text{at } i \ \text{in}' \ H \rangle$ 
      using *(1) by simp
    then have  $\langle (\diamond \text{Nom } j) \ \text{at } a \ \text{in}' \ H \rangle$ 
      using ur-closure assms(1) a(2) by fast
    then have  $\langle \text{assign } A \ H \ j \in \text{reach } A \ H \ (\text{assign } A \ H \ i) \rangle$ 
      unfolding reach-def using a(1) by fast
    then show ?thesis
      using j(1) by simp
  next
  case False
    then obtain a where a:  $\langle a \in \text{assign } A \ H \ i \rangle \langle (\diamond p) \ \text{at } a \ \text{in}' \ H \rangle$ 
      using ur-closure' assms(1)  $\langle (\diamond p) \ \text{at } i \ \text{in}' \ H \rangle$  by fast
    then have  $\langle \exists j. (\diamond \text{Nom } j) \ \text{at } a \ \text{in}' \ H \wedge (@ j p) \ \text{at } a \ \text{in}' \ H \rangle$ 
      using False assms  $\langle (\diamond p) \ \text{at } i \ \text{in}' \ H \rangle$  by  $(\text{meson } \text{Hintikka.DiaP})$ 
    then obtain j where j:  $\langle (\diamond \text{Nom } j) \ \text{at } a \ \text{in}' \ H \rangle \langle (@ j p) \ \text{at } a \ \text{in}' \ H \rangle$ 

```

```

    by blast

  from  $j(2)$  have  $\langle p \text{ at } j \text{ in}' H \rangle$ 
    using  $assms(1)$  Hintikka.SatP by fast
  then have  $\langle Model \text{ (reach } A \ H) \text{ (val } H), assign \ A \ H, assign \ A \ H \ j \models p \rangle$ 
    using  $Dia \ p \ *(2)$  by simp
  moreover have  $\langle assign \ A \ H \ j \in reach \ A \ H \text{ (assign } \ A \ H \ i) \rangle$ 
    unfolding reach-def using  $a(1) \ j(1)$  by blast
  ultimately show ?thesis
    by auto
qed
next
fix  $i$ 
case ( $Dia \ p$ )
assume *:  $\langle (\neg (\Diamond \ p)) \text{ at } i \text{ in}' H \rangle \langle nominals \ (\Diamond \ p) \subseteq A \rangle$ 
then obtain  $a$  where  $a$ :  $\langle a \in assign \ A \ H \ i \rangle \langle (\neg (\Diamond \ p)) \text{ at } a \text{ in}' H \rangle$ 
  using ur-closure' assms(1) by fast
{
  fix  $j \ b$ 
  assume  $\langle (\Diamond \ Nom \ j) \text{ at } b \text{ in}' H \rangle \langle b \in assign \ A \ H \ a \rangle$ 
  moreover have  $\langle (\neg (\Diamond \ p)) \text{ at } b \text{ in}' H \rangle$ 
    using  $a(2)$   $assms(1)$  calculation(2) ur-closure by fast
  ultimately have  $\langle (\neg (\@ \ j \ p)) \text{ at } b \text{ in}' H \rangle$ 
    using  $assms(1)$  Hintikka.DiaN by fast
  then have  $\langle (\neg \ p) \text{ at } j \text{ in}' H \rangle$ 
    using  $assms(1)$  Hintikka.SatN by fast
  then have  $\langle \neg \ Model \text{ (reach } A \ H) \text{ (val } H), assign \ A \ H, assign \ A \ H \ j \models p \rangle$ 
    using  $Dia \ *(2)$  by simp
}
then have  $\langle \neg \ Model \text{ (reach } A \ H) \text{ (val } H), assign \ A \ H, assign \ A \ H \ a \models \Diamond \ p \rangle$ 
  unfolding reach-def by auto
moreover have  $\langle assign \ A \ H \ a = assign \ A \ H \ i \rangle$ 
  using  $assms(1)$  a assign-unique by fast
ultimately show  $\langle \neg \ Model \text{ (reach } A \ H) \text{ (val } H), assign \ A \ H, assign \ A \ H \ i \models$ 
 $\Diamond \ p \rangle$ 
  by simp
next
fix  $i$ 
case ( $Sat \ j \ p$ )
assume  $\langle (\@ \ j \ p) \text{ at } i \text{ in}' H \rangle \langle nominals \ (\@ \ j \ p) \subseteq A \rangle$ 
moreover from this have  $\langle \forall a. p = (\Diamond \ Nom \ a) \longrightarrow a \in A \rangle$ 
  by auto
moreover have  $\langle p \text{ at } j \text{ in}' H \rangle$  if  $\langle \exists a. (\@ \ j \ p) \text{ at } a \text{ in}' H \rangle$ 
  using that assms(1) Hintikka.SatP by fast
ultimately show  $\langle Model \text{ (reach } A \ H) \text{ (val } H), assign \ A \ H, assign \ A \ H \ i \models \@$ 
 $j \ p \rangle$ 
  using Sat by auto
next
fix  $i$ 

```

**case**  $\langle \text{Sat } j \ p \rangle$   
**assume**  $\langle \neg (\text{@ } j \ p) \text{ at } i \text{ in } H \rangle \langle \text{nominals } (\text{@ } j \ p) \subseteq A \rangle$   
**moreover from this have**  $\langle \forall a. p = (\diamond \text{Nom } a) \longrightarrow a \in A \rangle$   
**by auto**  
**moreover have**  $\langle \neg p \text{ at } j \text{ in } H \rangle$  **if**  $\langle \exists a. (\neg (\text{@ } j \ p)) \text{ at } a \text{ in } H \rangle$   
**using that assms**(1) *Hintikka.SatN* **by fast**  
**ultimately show**  $\langle \neg \text{Model } (\text{reach } A \ H) \ (val \ H), \text{ assign } A \ H, \text{ assign } A \ H \ i \models$   
 $\text{@ } j \ p \rangle$   
**using Sat by fastforce**  
**qed**

## 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** ::  $\langle 'b \text{ set} \Rightarrow ('a, 'b) \text{ block set} \Rightarrow \text{bool} \rangle$  **where**  
 $\langle \text{consistent } A \ S \equiv \nexists S'. \text{ set } S' \subseteq S \wedge A \vdash S' \rangle$

**instance fm** ::  $\langle \text{countable}, \text{countable} \rangle \text{countable}$   
**by countable-datatype**

**definition proper-dia** ::  $\langle ('a, 'b) \text{ fm} \Rightarrow ('a, 'b) \text{ fm option} \rangle$  **where**  
 $\langle \text{proper-dia } p \equiv \text{case } p \text{ of } (\diamond p) \Rightarrow (\text{if } \nexists a. p = \text{Nom } a \text{ then } \text{Some } p \text{ else } \text{None}) \mid$   
 $- \Rightarrow \text{None} \rangle$

**lemma proper-dia**:  $\langle \text{proper-dia } p = \text{Some } q \Longrightarrow p = (\diamond q) \wedge (\nexists a. q = \text{Nom } a) \rangle$   
**unfolding proper-dia-def by**  $\langle \text{cases } p \rangle$   $\langle \text{simp-all}, \text{metis option.discI option.inject} \rangle$

The following function witnesses each  $\diamond p$  in a fresh world.

**primrec witness-list** ::  $\langle ('a, 'b) \text{ fm list} \Rightarrow 'b \text{ set} \Rightarrow ('a, 'b) \text{ fm list} \rangle$  **where**  
 $\langle \text{witness-list } [] \ - = [] \rangle$   
 $\mid \langle \text{witness-list } (p \ # \ ps) \ \text{used} =$   
 $\quad \langle \text{case proper-dia } p \ \text{of}$   
 $\quad \quad \text{None} \Rightarrow \text{witness-list } ps \ \text{used}$   
 $\quad \mid \text{Some } q \Rightarrow$   
 $\quad \quad \text{let } i = \text{SOME } i. i \notin \text{used}$   
 $\quad \quad \text{in } (\text{@ } i \ q) \ # \ (\diamond \text{Nom } i) \ # \ \text{witness-list } ps \ (\{i\} \cup \text{used}) \rangle$

**primrec witness** ::  $\langle ('a, 'b) \text{ block} \Rightarrow 'b \text{ set} \Rightarrow ('a, 'b) \text{ block} \rangle$  **where**  
 $\langle \text{witness } (ps, a) \ \text{used} = (\text{witness-list } ps \ \text{used}, a) \rangle$

**lemma witness-list**:  
 $\langle \text{proper-dia } p = \text{Some } q \Longrightarrow \text{witness-list } (p \ # \ ps) \ \text{used} =$   
 $\quad \langle \text{let } i = \text{SOME } i. i \notin \text{used}$   
 $\quad \quad \text{in } (\text{@ } i \ q) \ # \ (\diamond \text{Nom } i) \ # \ \text{witness-list } ps \ (\{i\} \cup \text{used}) \rangle$   
**by simp**

**primrec** *extend* ::  
 $\langle 'b \text{ set} \Rightarrow ('a, 'b) \text{ block set} \Rightarrow (\text{nat} \Rightarrow ('a, 'b) \text{ block}) \Rightarrow \text{nat} \Rightarrow ('a, 'b) \text{ block set} \rangle$   
**where**  
 $\langle \text{extend } A \text{ } S \text{ } f \ 0 = S \rangle$   
 $| \langle \text{extend } A \text{ } S \text{ } f \ (\text{Suc } n) =$   
 $\quad (\text{if } \neg \text{consistent } A \ (\{f \ n\} \cup \text{extend } A \text{ } S \text{ } f \ n)$   
 $\quad \text{then } \text{extend } A \text{ } S \text{ } f \ n$   
 $\quad \text{else}$   
 $\quad \text{let } \text{used} = A \cup (\bigcup \text{block} \in \{f \ n\} \cup \text{extend } A \text{ } S \text{ } f \ n. \text{block-nominals } \text{block})$   
 $\quad \text{in } \{f \ n, \text{witness } (f \ n) \ \text{used}\} \cup \text{extend } A \text{ } S \text{ } f \ n \rangle$

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

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

**lemma** *extend-mem*:  $\langle S \subseteq \text{extend } A \text{ } S \text{ } f \ n \rangle$   
**by** (*induct n auto*)

**lemma** *Extend-mem*:  $\langle S \subseteq \text{Extend } A \text{ } S \text{ } f \rangle$   
**unfolding** *Extend-def* **using** *extend-mem* **by** *fast*

### 11.2.1 Consistency

**lemma** *split-list*:  
 $\langle \text{set } A \subseteq \{x\} \cup X \Longrightarrow x \in A \Longrightarrow \exists B. \text{set } (x \# B) = \text{set } A \wedge x \notin \text{set } B \rangle$   
**by** *simp (metis Diff-insert-absorb mk-disjoint-insert set-removeAll)*

**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**  $\langle \text{consistent } A \ (\{(ps, a)\} \cup S) \rangle$   
**unfolding** *consistent-def*

**proof**  
**assume**  $\langle \exists S'. \text{set } S' \subseteq \{(ps, a)\} \cup S \wedge A \vdash S' \rangle$   
**then obtain**  $S' \ n$  **where**  $\langle \text{set } S' \subseteq \{(ps, a)\} \cup S \rangle \langle (ps, a) \in S' \rangle \langle A, n \vdash S' \rangle$   
**using** *assms* **unfolding** *consistent-def* **by** *blast*  
**then obtain**  $S''$  **where**  $\langle \text{set } ((ps, a) \# S'') = \text{set } S' \rangle \langle (ps, a) \notin \text{set } S'' \rangle$   
**using** *split-list* **by** *metis*  
**then have**  $\langle A \vdash (ps, a) \# S'' \rangle$   
**using** *inf fin STA-struct*  $\langle A, n \vdash S' \rangle$  **by** *blast*  
**then have**  $\langle A \vdash (p \# ps, a) \# S'' \rangle$   
**using** *inf fin STA-struct-block* [**where**  $ps' = (p \# ps)$ ] **by** *fastforce*

**moreover have**  $\langle \text{set } ((p \# ps, a) \# S'') \subseteq \{(p \# ps, a)\} \cup S \rangle$   
**using**  $\langle (ps, a) \notin \text{set } S'' \rangle \langle \text{set } ((ps, a) \# S'') = \text{set } S' \rangle \langle \text{set } S' \subseteq \{(ps, a)\} \cup S \rangle$   
**by** *auto*  
**ultimately show** *False*  
**using** *cons* **unfolding** *consistent-def* **by** *blast*  
**qed**

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

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

**lemma** *finite-nominals-set*:  $\langle \text{finite } S \implies \text{finite } (\bigcup block \in S. block\text{-nominals } block) \rangle$   
**by** (*induct* *S* *rule*: *finite-induct*) (*simp-all* *add*: *finite-block-nominals*)

**lemma** *witness-list-used*:  
**fixes** *i* :: 'b  
**assumes** *inf*:  $\langle \text{infinite } (UNIV :: 'b \text{ set}) \rangle$  **and**  $\langle \text{finite used} \rangle$   $\langle i \notin \text{list-nominals } ps \rangle$   
**shows**  $\langle i \notin \text{list-nominals } (\text{witness-list } ps (\{i\} \cup \text{used})) \rangle$   
**using** *assms*(2-)

**proof** (*induct* *ps* *arbitrary*: *used*)

**case** (*Cons* *p* *ps*)

**then show** *?case*

**proof** (*cases*  $\langle \text{proper-dia } p \rangle$ )

**case** (*Some* *q*)

**let** *?j* =  $\langle \text{SOME } j. j \notin \{i\} \cup \text{used} \rangle$

**have**  $\langle \text{finite } (\{i\} \cup \text{used}) \rangle$

**using**  $\langle \text{finite used} \rangle$  **by** *simp*

**then have**  $\langle \exists j. j \notin \{i\} \cup \text{used} \rangle$

**using** *inf* *ex-new-if-finite* **by** *metis*

**then have** *j*:  $\langle ?j \notin \{i\} \cup \text{used} \rangle$

**using** *someI-ex* **by** *metis*

**have**  $\langle \text{witness-list } (p \# ps) (\{i\} \cup \text{used}) =$

$(@ ?j q) \# (\diamond \text{Nom } ?j) \# \text{witness-list } ps (\{?j\} \cup (\{i\} \cup \text{used})) \rangle$

**using** *Some* *witness-list* **by** *metis*

**then have** \*:  $\langle \text{list-nominals } (\text{witness-list } (p \# ps) (\{i\} \cup \text{used})) =$

$\{?j\} \cup \text{nominals } q \cup \text{list-nominals } (\text{witness-list } ps (\{?j\} \cup (\{i\} \cup \text{used})) \rangle$

**by** *simp*

**have**  $\langle \text{finite } (\{?j\} \cup \text{used}) \rangle$

**using**  $\langle \text{finite used} \rangle$  **by** *simp*

**moreover have**  $\langle i \notin \text{list-nominals } ps \rangle$

**using**  $\langle i \notin \text{list-nominals } (p \# ps) \rangle$  **by** *simp*

**ultimately have**  $\langle i \notin \text{list-nominals } (\text{witness-list } ps (\{i\} \cup (\{?j\} \cup \text{used})) \rangle$

**using** *Cons* **by** *metis*

**moreover have**  $\langle \{i\} \cup (\{?j\} \cup \text{used}) = \{?j\} \cup (\{i\} \cup \text{used}) \rangle$

**by** *blast*



**moreover have**  $\langle i \neq ?j \rangle$   
**using**  $j$  **by** *auto*  
**ultimately have**  $\langle i \in \text{list-nominals } (\text{witness-list } (p \# ps) (\{i\} \cup \text{used})) \longleftrightarrow$   
 $i \in \text{nominals } q \rangle$   
**using**  $*$  **by** *simp*  
**moreover have**  $\langle i \notin \text{nominals } q \rangle$   
**using** *Cons(3)* *Some proper-dia* **by** *fastforce*  
**ultimately show** *?thesis*  
**by** *blast*  
**qed** *simp*  
**qed** *simp*

**lemma** *witness-used:*

**fixes**  $i :: 'b$   
**assumes** *inf*:  $\langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$  **and**  
 $\langle \text{finite used} \rangle$   $\langle i \notin \text{block-nominals } \text{block} \rangle$   
**shows**  $\langle i \notin \text{block-nominals } (\text{witness } \text{block } (\{i\} \cup \text{used})) \rangle$   
**using** *assms witness-list-used* **by** (*induct block*) *fastforce*

**lemma** *consistent-witness-list:*

**fixes**  $a :: 'b$   
**assumes** *inf*:  $\langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$  **and**  $\langle \text{consistent } A \ S \rangle$   
 $\langle (ps, a) \in S \rangle$   $\langle \text{finite used} \rangle$   $\langle A \cup \bigcup (\text{block-nominals } ' S) \subseteq \text{used} \rangle$   
**shows**  $\langle \text{consistent } A \ (\{(witness-list \ ps \ used, a)\} \cup S) \rangle$   
**using** *assms(2-)*  
**proof** (*induct ps arbitrary: used S*)  
**case** *Nil*  
**then have**  $\langle \{(witness-list \ [] \ used, a)\} \cup S = S \rangle$   
**by** *auto*  
**moreover have**  $\langle \text{finite } \{ \} \rangle$   $\langle \{ \} \cap \text{used} = \{ \} \rangle$   
**by** *simp-all*  
**ultimately show** *?case*  
**using**  $\langle \text{consistent } A \ S \rangle$  **by** *simp*

**next**

**case** (*Cons p ps*)  
**have** *fin*:  $\langle \text{finite } A \rangle$   
**using** *assms(4-5)* *finite-subset* **by** *fast*  
**have**  $\langle \{(p \# ps, a)\} \cup S = S \rangle$   
**using**  $\langle (p \# ps, a) \in S \rangle$  **by** *blast*  
**then have**  $\langle \text{consistent } A \ (\{(p \# ps, a)\} \cup S) \rangle$   
**using**  $\langle \text{consistent } A \ S \rangle$  **by** *simp*  
**then have**  $\langle \text{consistent } A \ (\{(ps, a)\} \cup S) \rangle$   
**using** *inf fin consistent-drop-single* **by** *fast*  
**moreover have**  $\langle (ps, a) \in \{(ps, a)\} \cup S \rangle$   
**by** *simp*  
**moreover have**  $\langle A \cup \bigcup (\text{block-nominals } ' (\{(ps, a)\} \cup S)) \subseteq \text{extra} \cup \text{used} \rangle$  **for**  
*extra*  
**using**  $\langle (p \# ps, a) \in S \rangle$   $\langle A \cup \bigcup (\text{block-nominals } ' S) \subseteq \text{used} \rangle$  **by** *fastforce*  
**moreover have**  $\langle \text{finite } (\text{extra} \cup \text{used}) \rangle$  **if**  $\langle \text{finite } \text{extra} \rangle$  **for** *extra*

**using** *that*  $\langle \text{finite used} \rangle$  **by** *blast*  
**ultimately have** *cons*:  
 $\langle \text{consistent } A \ (\{(witness\text{-list } ps \ (extra \cup used), a)\} \cup (\{(ps, a)\} \cup S)) \rangle$   
**if**  $\langle \text{finite extra} \rangle$  **for** *extra*  
**using** *that* *Cons* **by** *metis*

**show** *?case*  
**proof** (*cases*  $\langle \text{proper-dia } p \rangle$ )  
**case** *None*  
**then have**  $\langle witness\text{-list } (p \# ps) \ used = witness\text{-list } ps \ used \rangle$   
**by** *auto*  
**moreover have**  $\langle \text{consistent } A \ (\{(witness\text{-list } ps \ used, a)\} \cup (\{(ps, a)\} \cup S)) \rangle$   
**using** *cons* [**where** *extra* =  $\{\}$ ] **by** *simp*  
**then have**  $\langle \text{consistent } A \ (\{(witness\text{-list } ps \ used, a)\} \cup S) \rangle$   
**using** *consistent-drop-block* [**where** *block* =  $\langle (ps, a) \rangle$ ] **by** *auto*  
**ultimately show** *?thesis*  
**by** *simp*

**next**  
**case** (*Some* *q*)  
**let** *?i* =  $\langle \text{SOME } i. i \notin used \rangle$   
**have**  $\langle \exists i. i \notin used \rangle$   
**using** *ex-new-if-finite inf*  $\langle \text{finite used} \rangle$  .  
**with** *someI-ex* **have**  $\langle ?i \notin used \rangle$  .  
**then have** *i*:  $\langle ?i \notin \bigcup (\text{block-nominals } 'S) \rangle$   
**using** *Cons* **by** *auto*  
**then have**  $\langle ?i \notin \text{block-nominals } (p \# ps, a) \rangle$   
**using** *Cons* **by** *blast*

**let** *?tail* =  $\langle witness\text{-list } ps \ (\{ ?i \} \cup used) \rangle$

**have**  $\langle \text{consistent } A \ (\{( ?tail, a \}) \cup (\{(ps, a)\} \cup S)) \rangle$   
**using** *cons* [**where** *extra* =  $\{ ?i \}$ ] **by** *blast*  
**then have** *\**:  $\langle \text{consistent } A \ (\{( ?tail, a \}) \cup S) \rangle$   
**using** *consistent-drop-block* [**where** *block* =  $\langle (ps, a) \rangle$ ] **by** *simp*

**have**  $\langle witness\text{-list } (p \# ps) \ used = (@ ?i q) \# (\diamond \text{Nom } ?i) \# ?tail \rangle$   
**using** *Some witness-list* **by** *metis*  
**moreover have**  $\langle \text{consistent } A \ (\{((@ ?i q) \# (\diamond \text{Nom } ?i) \# ?tail, a)\} \cup S) \rangle$   
**unfolding** *consistent-def*

**proof**  
**assume**  $\langle \exists S'. \text{set } S' \subseteq \{((@ ?i q) \# (\diamond \text{Nom } ?i) \# ?tail, a)\} \cup S \wedge A \vdash S' \rangle$   
**then obtain** *S' n* **where**  
 $\langle A, n \vdash S' \rangle$  **and** *S'*:  
 $\langle \text{set } S' \subseteq \{((@ ?i q) \# (\diamond \text{Nom } ?i) \# ?tail, a)\} \cup S \rangle$   
 $\langle ((@ ?i q) \# (\diamond \text{Nom } ?i) \# ?tail, a) \in S' \rangle$   
**using** *\** **unfolding** *consistent-def* **by** *blast*  
**then obtain** *S''* **where** *S''*:  
 $\langle \text{set } (\{((@ ?i q) \# (\diamond \text{Nom } ?i) \# ?tail, a) \# S''\}) = \text{set } S' \rangle$   
 $\langle ((@ ?i q) \# (\diamond \text{Nom } ?i) \# ?tail, a) \notin \text{set } S'' \rangle$

**using** *split-list*[**where**  $x = ((@ ?i q) \# (\diamond \text{Nom } ?i) \# ?tail, a)$ ] **by** *blast*  
**then have**  $\langle A \vdash ((@ ?i q) \# (\diamond \text{Nom } ?i) \# ?tail, a) \# S'' \rangle$   
**using** *inf*  $\langle \text{finite } A \rangle$  *STA-struct*  $\langle A, n \vdash S' \rangle$  **by** *blast*  
**moreover have**  $\langle \text{set } (((@ ?i q) \# (\diamond \text{Nom } ?i) \# ?tail, a) \# S'') \subseteq$   
 $\text{set } (((@ ?i q) \# (\diamond \text{Nom } ?i) \# ?tail, a) \# (p \# ps, a) \# S'') \rangle$   
**by** *auto*  
**ultimately have**  $** : \langle A \vdash ((@ ?i q) \# (\diamond \text{Nom } ?i) \# ?tail, a) \# (p \# ps,$   
 $a) \# S'' \rangle$   
**using** *inf*  $\langle \text{finite } A \rangle$  *STA-struct* **by** *blast*

**have**  $\langle ?i \notin \text{block-nominals } (?tail, a) \rangle$   
**using** *inf*  $\langle \text{finite used} \rangle$   $\langle ?i \notin \text{block-nominals } (p \# ps, a) \rangle$  *witness-used* **by**  
*fastforce*

**moreover have**  $\langle ?i \notin \text{branch-nominals } S'' \rangle$   
**unfolding** *branch-nominals-def* **using** *i S' S''* **by** *auto*  
**ultimately have**  $\langle ?i \notin \text{branch-nominals } ((?tail, a) \# (p \# ps, a) \# S'') \rangle$   
**using**  $\langle ?i \notin \text{block-nominals } (p \# ps, a) \rangle$  *unfolding* *branch-nominals-def*  
**by** *simp*  
**then have**  $\langle ?i \notin A \cup \text{branch-nominals } ((?tail, a) \# (p \# ps, a) \# S'') \rangle$   
**using**  $\langle ?i \notin \text{used} \rangle$  *Cons.prem*(4) **by** *blast*

**moreover have**  $\langle \nexists a. q = \text{Nom } a \rangle$   
**using** *Some proper-dia* **by** *blast*  
**moreover have**  $\langle (p \# ps, a) \in. (?tail, a) \# (p \# ps, a) \# S'' \rangle$   
**by** *simp*  
**moreover have**  $\langle p = (\diamond q) \rangle$   
**using** *Some proper-dia* **by** *blast*  
**then have**  $\langle (\diamond q) \text{ on } (p \# ps, a) \rangle$   
**by** *simp*  
**ultimately have**  $\langle A \vdash (?tail, a) \# (p \# ps, a) \# S'' \rangle$   
**using**  $** \langle \text{finite } A \rangle$  *DiaP''* **by** *fast*  
**moreover have**  $\langle \text{set } ((p \# ps, a) \# S'') \subseteq S \rangle$   
**using** *Cons*(3) *S' S''* **by** *auto*  
**ultimately show** *False*  
**using**  $*$  *unfolding* *consistent-def* **by** (*simp add: subset-Un-eq*)

**qed**  
**ultimately show** *?thesis*  
**by** *simp*

**qed**  
**qed**

**lemma** *consistent-witness*:  
**fixes** *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**  $\langle \text{consistent } A \ (\{\text{witness } \text{block } (A \cup \bigcup (\text{block-nominals } ' S))\} \cup S) \rangle$   
**using** *assms* *consistent-witness-list* **by** (*cases block*) *fastforce*

**lemma** *consistent-extend*:

```

fixes  $S :: \langle ('a, 'b) \text{ block set} \rangle$ 
assumes  $\text{inf} : \langle \text{infinite } (UNIV :: 'b \text{ set}) \rangle$  and  $\text{fin} : \langle \text{finite } A \rangle$  and
   $\langle \text{consistent } A \text{ (extend } A \text{ } S \text{ } f \text{ } n) \rangle \langle \text{finite } (\bigcup (\text{block-nominals } ' \text{ extend } A \text{ } S \text{ } f \text{ } n)) \rangle$ 
shows  $\langle \text{consistent } A \text{ (extend } A \text{ } S \text{ } f \text{ } (\text{Suc } n)) \rangle$ 
proof ( $\text{cases } \langle \text{consistent } A \text{ } (\{f \text{ } n\} \cup \text{extend } A \text{ } S \text{ } f \text{ } n) \rangle$ )
  case  $\text{True}$ 
  let  $?used = \langle A \cup (\bigcup \text{block} \in \{f \text{ } n\} \cup \text{extend } A \text{ } S \text{ } f \text{ } n. \text{block-nominals } \text{block}) \rangle$ 
  have  $*$ :  $\langle \text{extend } A \text{ } S \text{ } f \text{ } (n + 1) = \{f \text{ } n, \text{witness } (f \text{ } n) \text{ } ?used\} \cup \text{extend } A \text{ } S \text{ } f \text{ } n \rangle$ 
    using  $\text{True}$  by  $\text{simp}$ 

  have  $\langle \text{consistent } A \text{ } (\{f \text{ } n\} \cup \text{extend } A \text{ } S \text{ } f \text{ } n) \rangle$ 
    using  $\text{True}$  by  $\text{simp}$ 
  moreover have  $\langle \text{finite } ((\bigcup (\text{block-nominals } ' (\{f \text{ } n\} \cup \text{extend } A \text{ } S \text{ } f \text{ } n)))) \rangle$ 
    using  $\langle \text{finite } (\bigcup (\text{block-nominals } ' \text{ extend } A \text{ } S \text{ } f \text{ } n)) \rangle$   $\text{finite-nominals-set}$  by  $\text{force}$ 
  moreover have  $\langle f \text{ } n \in \{f \text{ } n\} \cup \text{extend } A \text{ } S \text{ } f \text{ } n \rangle$ 
    by  $\text{simp}$ 
  ultimately have  $\langle \text{consistent } A \text{ } (\{\text{witness } (f \text{ } n) \text{ } ?used\} \cup (\{f \text{ } n\} \cup \text{extend } A \text{ } S \text{ } f \text{ } n)) \rangle$ 
    using  $\text{inf fin consistent-witness}$  by  $\text{blast}$ 
  then show  $?thesis$ 
    using  $*$  by  $\text{simp}$ 
next
  case  $\text{False}$ 
  then show  $?thesis$ 
    using  $\text{assms}(3)$  by  $\text{simp}$ 
qed

```

```

lemma  $\text{finite-nominals-extend}$ :
  assumes  $\langle \text{finite } (\bigcup (\text{block-nominals } ' S)) \rangle$ 
  shows  $\langle \text{finite } (\bigcup (\text{block-nominals } ' \text{ extend } A \text{ } S \text{ } f \text{ } n)) \rangle$ 
  using  $\text{assms}$  by ( $\text{induct } n$ ) ( $\text{auto simp add: finite-block-nominals}$ )

```

```

lemma  $\text{consistent-extend}'$ :
  fixes  $S :: \langle ('a, 'b) \text{ block set} \rangle$ 
  assumes  $\langle \text{infinite } (UNIV :: 'b \text{ set}) \rangle \langle \text{finite } A \rangle \langle \text{consistent } A \text{ } S \rangle \langle \text{finite } (\bigcup (\text{block-nominals } ' S)) \rangle$ 
  shows  $\langle \text{consistent } A \text{ (extend } A \text{ } S \text{ } f \text{ } n) \rangle$ 
  using  $\text{assms}$ 
proof ( $\text{induct } n$ )
  case  $(\text{Suc } n)$ 
  then show  $?case$ 
    by ( $\text{metis consistent-extend finite-nominals-extend}$ )
qed  $\text{simp}$ 

```

```

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

```

**case**  $(\text{insert } x \ A)$   
**then obtain**  $m$  **where**  $\langle A \subseteq (\bigcup n \leq m. f \ n) \rangle$   
**by** *fast*  
**then have**  $\langle A \subseteq (\bigcup n \leq (m + k). f \ n) \rangle$  **for**  $k$   
**by** *fastforce*  
**moreover obtain**  $m'$  **where**  $\langle x \in f \ m' \rangle$   
**using** *insert(4)* **by** *blast*  
**ultimately have**  $\langle \{x\} \cup A \subseteq (\bigcup n \leq m + m'. f \ n) \rangle$   
**by** *auto*  
**then show** *?case*  
**by** *blast*  
**qed** *simp*

**lemma** *extend-bound*:  $\langle (\bigcup n \leq m. \text{extend } A \ S \ f \ n) = \text{extend } A \ S \ f \ m \rangle$   
**proof** *(induct m)*  
**case**  $(\text{Suc } m)$   
**have**  $\langle \bigcup (\text{extend } A \ S \ f \ ' \{.. \text{Suc } m\}) = \bigcup (\text{extend } A \ S \ f \ ' \{..m\}) \cup \text{extend } A \ S \ f \ (\text{Suc } m) \rangle$   
**using** *atMost-Suc* **by** *auto*  
**also have**  $\langle \dots = \text{extend } A \ S \ f \ m \cup \text{extend } A \ S \ f \ (\text{Suc } m) \rangle$   
**using** *Suc* **by** *blast*  
**also have**  $\langle \dots = \text{extend } A \ S \ f \ (\text{Suc } m) \rangle$   
**using** *extend-chain* **by** *blast*  
**finally show** *?case*  
**by** *simp*  
**qed** *simp*

**lemma** *consistent-Extend*:  
**fixes**  $S :: \langle ('a, 'b) \text{ block set} \rangle$   
**assumes** *inf*:  $\langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$  **and**  $\langle \text{finite } A \rangle$   
 $\langle \text{consistent } A \ S \rangle$   $\langle \text{finite } (\bigcup (\text{block-nominals } ' S)) \rangle$   
**shows**  $\langle \text{consistent } A \ (\text{Extend } A \ S \ f) \rangle$   
**unfolding** *Extend-def*  
**proof** *(rule ccontr)*  
**assume**  $\langle \neg \text{consistent } A \ (\bigcup (\text{range } (\text{extend } A \ S \ f))) \rangle$   
**then obtain**  $S' \ n$  **where**  $*$ :  
 $\langle A, n \vdash S' \rangle$   
 $\langle \text{set } S' \subseteq (\bigcup n. \text{extend } A \ S \ f \ n) \rangle$   
**unfolding** *consistent-def* **by** *blast*  
**moreover have**  $\langle \text{finite } (\text{set } S') \rangle$   
**by** *simp*  
**ultimately obtain**  $m$  **where**  $\langle \text{set } S' \subseteq (\bigcup n \leq m. \text{extend } A \ S \ f \ n) \rangle$   
**using** *UN-finite-bound* **by** *metis*  
**then have**  $\langle \text{set } S' \subseteq \text{extend } A \ S \ f \ m \rangle$   
**using** *extend-bound* **by** *blast*  
**moreover have**  $\langle \text{consistent } A \ (\text{extend } A \ S \ f \ m) \rangle$   
**using** *assms consistent-extend'* **by** *blast*  
**ultimately show** *False*  
**unfolding** *consistent-def* **using**  $*$  **by** *blast*

qed

### 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**  
 $\langle \text{maximal } A \ S \equiv \text{consistent } A \ S \wedge (\forall \text{block. block} \notin S \longrightarrow \neg \text{consistent } A \ (\{\text{block}\} \cup S)) \rangle$

**lemma** *extend-not-mem*:

$\langle f \ n \notin \text{extend } A \ S \ f \ (\text{Suc } n) \implies \neg \text{consistent } A \ (\{f \ n\} \cup \text{extend } A \ S \ f \ n) \rangle$   
**by** (*metis Un-insert-left extend.simps(2) insertI1*)

**lemma** *maximal-Extend*:

**fixes**  $S :: \langle ('a, 'b) \text{ block set} \rangle$   
**assumes** *inf*:  $\langle \text{infinite } (\text{UNIV} :: 'b \text{ set}) \rangle$  **and** *finite A*  
 $\langle \text{consistent } A \ S \rangle$  *finite*  $\langle \bigcup (\text{block-nominals } 'S) \rangle$  *surj f*  
**shows**  $\langle \text{maximal } A \ (\text{Extend } A \ S \ f) \rangle$   
**proof** (*rule ccontr*)  
**assume**  $\langle \neg \text{maximal } A \ (\text{Extend } A \ S \ f) \rangle$   
**then obtain** *block* **where**  
 $\langle \text{block} \notin \text{Extend } A \ S \ f \rangle$   $\langle \text{consistent } A \ (\{\text{block}\} \cup \text{Extend } A \ S \ f) \rangle$   
**unfolding** *maximal-def* **using** *assms consistent-Extend* **by** *metis*  
**obtain**  $n$  **where**  $n: \langle f \ n = \text{block} \rangle$   
**using**  $\langle \text{surj } f \rangle$  **unfolding** *surj-def* **by** *metis*  
**then have**  $\langle \text{block} \notin \text{extend } A \ S \ f \ (\text{Suc } n) \rangle$   
**using**  $\langle \text{block} \notin \text{Extend } A \ S \ f \rangle$  *extend-chain* **unfolding** *Extend-def* **by** *blast*  
**then have**  $\langle \neg \text{consistent } A \ (\{\text{block}\} \cup \text{extend } A \ S \ f \ n) \rangle$   
**using**  $n$  *extend-not-mem* **by** *blast*  
**moreover have**  $\langle \text{block} \notin \text{extend } A \ S \ f \ n \rangle$   
**using**  $\langle \text{block} \notin \text{extend } A \ S \ f \ (\text{Suc } n) \rangle$  *extend-chain* **by** *blast*  
**then have**  $\langle \{\text{block}\} \cup \text{extend } A \ S \ f \ n \subseteq \{\text{block}\} \cup \text{Extend } A \ S \ f \rangle$   
**unfolding** *Extend-def* **by** *blast*  
**ultimately have**  $\langle \neg \text{consistent } A \ (\{\text{block}\} \cup \text{Extend } A \ S \ f) \rangle$   
**using** *inconsistent-weaken* **by** *blast*  
**then show** *False*  
**using**  $\langle \text{consistent } A \ (\{\text{block}\} \cup \text{Extend } A \ S \ f) \rangle$  **by** *simp*  
qed

### 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**  
 $\langle \text{saturated } S \equiv \forall p \ i. (\diamond p) \text{ at } i \text{ in } S \longrightarrow (\exists a. p = \text{Nom } a) \longrightarrow$   
 $\langle \exists j. (@ j p) \text{ at } i \text{ in } S \wedge (\diamond \text{Nom } j) \text{ at } i \text{ in } S \rangle$

**lemma** *witness-list-append*:

$\langle \exists \text{extra. witness-list } (ps \text{ @ } qs) \text{ used} = \text{witness-list } ps \text{ used @ witness-list } qs \text{ (extra} \cup \text{used)} \rangle$   
**proof** (induct ps arbitrary: used)  
  **case** Nil  
  **then show** ?case  
  **by** (metis Un-absorb append-self-conv2 witness-list.simps(1))  
**next**  
  **case** (Cons p ps)  
  **show** ?case  
  **proof** (cases  $\langle \exists q. \text{proper-dia } p = \text{Some } q \rangle$ )  
  **case** True  
  **let** ?i = (SOME i. i  $\notin$  used)  
  **from** True **obtain** q **where** q:  $\langle \text{proper-dia } p = \text{Some } q \rangle$   
  **by** blast  
  **moreover have**  $\langle (p \# ps) \text{ @ } qs = p \# (ps \text{ @ } qs) \rangle$   
  **by** simp  
  **ultimately have**  
   $\langle \text{witness-list } ((p \# ps) \text{ @ } qs) \text{ used} = (\text{@ } ?i \text{ } q) \# (\diamond \text{Nom } ?i) \#$   
   $\text{witness-list } (ps \text{ @ } qs) (\{?i\} \cup \text{used}) \rangle$   
  **using** witness-list **by** metis  
  **then have**  
   $\langle \exists \text{extra. witness-list } ((p \# ps) \text{ @ } qs) \text{ used} = (\text{@ } ?i \text{ } q) \# (\diamond \text{Nom } ?i) \#$   
   $\text{witness-list } ps (\{?i\} \cup \text{used}) \text{ @ witness-list } qs (\text{extra} \cup (\{?i\} \cup \text{used})) \rangle$   
  **using** Cons **by** metis  
  **moreover have**  $\langle (\text{@ } ?i \text{ } q) \# (\diamond \text{Nom } ?i) \# \text{witness-list } ps (\{?i\} \cup \text{used}) =$   
   $\text{witness-list } (p \# ps) \text{ used} \rangle$   
  **using** q witness-list **by** metis  
  **ultimately have**  $\langle \exists \text{extra. witness-list } ((p \# ps) \text{ @ } qs) \text{ used} =$   
   $\text{witness-list } (p \# ps) \text{ used @ witness-list } qs (\text{extra} \cup (\{?i\} \cup \text{used})) \rangle$   
  **by** (metis append-Cons)  
  **then have**  $\langle \exists \text{extra. witness-list } ((p \# ps) \text{ @ } qs) \text{ used} =$   
   $\text{witness-list } (p \# ps) \text{ used @ witness-list } qs ((\{?i\} \cup \text{extra}) \cup \text{used}) \rangle$   
  **by** simp  
  **then show** ?thesis  
  **by** blast  
**qed** (simp add: Cons)  
**qed**

**lemma** ex-witness-list:

**assumes**  $\langle p \in. ps \rangle \langle \text{proper-dia } p = \text{Some } q \rangle$   
**shows**  $\langle \exists i. \{ \text{@ } i \text{ } q, \diamond \text{Nom } i \} \subseteq \text{set } (\text{witness-list } ps \text{ used}) \rangle$   
**using**  $\langle p \in. ps \rangle$   
**proof** (induct ps arbitrary: used)  
  **case** (Cons a ps)  
  **then show** ?case  
  **proof** (induct  $\langle a = p \rangle$ )  
  **case** True  
  **then have**  
   $\langle \exists i. \text{witness-list } (a \# ps) \text{ used} = (\text{@ } i \text{ } q) \# (\diamond \text{Nom } i) \#$

```

    witness-list ps ({i} ∪ used)
  using ⟨proper-dia p = Some q⟩ witness-list by metis
  then show ?case
    by auto
  next
  case False
  then have ⟨∃ i. {Ⓜ i q, Ⓢ Nom i} ⊆ set (witness-list ps (extra ∪ used))⟩ for
  extra
    by simp
  moreover have ⟨∃ extra. witness-list (a # ps) used =
    witness-list [a] used Ⓜ witness-list ps (extra ∪ used)⟩
    using witness-list-append[where ps=⟨[-]⟩] by simp
  ultimately show ?case
    by fastforce
  qed
qed simp

```

lemma saturated-Extend:

```

  fixes S :: ⟨'a, 'b⟩ block set
  assumes inf: ⟨infinite (UNIV :: 'b set)⟩ and fin: ⟨finite A⟩ and
    ⟨consistent A S⟩ ⟨finite (⋃ (block-nominals ' S))⟩ ⟨surj f⟩
  shows ⟨saturated (Extend A S f)⟩
  unfolding saturated-def
proof safe
  fix ps i p
  assume ⟨(ps, i) ∈ Extend A S f⟩ ⟨(Ⓢ p) on (ps, i)⟩ ⟨Ⓢ a. p = Nom a⟩
  obtain n where n: ⟨f n = (ps, i)⟩
    using ⟨surj f⟩ unfolding surj-def by metis

```

```

let ?used = ⟨A ∪ (⋃ block ∈ {f n} ∪ extend A S f n. block-nominals block)⟩

```

```

  have ⟨extend A S f n ⊆ Extend A S f⟩
    unfolding Extend-def by auto
  moreover have ⟨consistent A (Extend A S f)⟩
    using assms consistent-Extend by blast
  ultimately have ⟨consistent A ({(ps, i)} ∪ extend A S f n)⟩
    using ⟨(ps, i) ∈ Extend A S f⟩ inconsistent-weaken by blast
  then have ⟨extend A S f (Suc n) = {f n, witness (f n) ?used} ∪ extend A S f n⟩
    using n ⟨(Ⓢ p) on (ps, i)⟩ by auto
  then have ⟨witness (f n) ?used ∈ Extend A S f⟩
    unfolding Extend-def by blast
  then have *: ⟨(witness-list ps ?used, i) ∈ Extend A S f⟩
    using n by simp

```

```

  have ⟨(Ⓢ p) ∈. ps⟩
    using ⟨(Ⓢ p) on (ps, i)⟩ by simp
  moreover have ⟨proper-dia (Ⓢ p) = Some p⟩
    unfolding proper-dia-def using ⟨Ⓢ a. p = Nom a⟩ by simp
  ultimately have ⟨∃ j.

```



```

    (@ j p) on (witness-list ps ?used, i) ∧
    (◇ Nom j) on (witness-list ps ?used, i)
  using ex-witness-list by fastforce
  then show ∃ j.
    (∃ qs. (qs, i) ∈ Extend A S f ∧ (@ j p) on (qs, i)) ∧
    (∃ rs. (rs, i) ∈ Extend A S f ∧ (◇ Nom j) on (rs, i))
  using * by blast
qed

```

### 11.3 Smullyan-Fitting

**lemma** *Hintikka-Extend*:

```

  fixes S :: ⟨'a, 'b⟩ 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⟩
  unfolding Hintikka-def
  proof safe
    fix x i j ps qs rs
    assume
      ps: ⟨(ps, i) ∈ S⟩ ⟨Nom j on (ps, i)⟩ and
      qs: ⟨(qs, j) ∈ S⟩ ⟨Pro x on (qs, j)⟩ and
      rs: ⟨(rs, i) ∈ S⟩ ⟨¬ Pro x on (rs, i)⟩
    then have ⟨¬ A, n ⊢ [(qs, j), (ps, i), (rs, i)]⟩ for n
      using ⟨consistent A S⟩ unfolding consistent-def by simp
    moreover have ⟨A, n ⊢ [(¬ Pro x) # qs, j), (ps, i), (rs, i)]⟩ for n
      using qs(2) Close
    by (metis (no-types, lifting) list.set-intros(1) on.simps set-subset-Cons subsetD)
    then have ⟨A, n ⊢ [(qs, j), (ps, i), (rs, i)]⟩ for n
      using ps(2) rs(2)
    by (meson Nom' fm.distinct(21) fm.simps(18) list.set-intros(1) set-subset-Cons
      subsetD)
    ultimately show False
      by blast
  next
    fix a i ps qs
    assume
      ps: ⟨(ps, i) ∈ S⟩ ⟨Nom a on (ps, i)⟩ and
      qs: ⟨(qs, i) ∈ S⟩ ⟨¬ Nom a on (qs, i)⟩
    then have ⟨¬ A, n ⊢ [(ps, i), (qs, i)]⟩ for n
      using ⟨consistent A S⟩ unfolding consistent-def by simp
    moreover have ⟨A, n ⊢ [(ps, i), (qs, i)]⟩ for n
      using ps(2) qs(2) by (meson Close list.set-intros(1) set-subset-Cons subset-code(1))
    ultimately show False
      by blast
  next
    fix p i ps
    assume ps: ⟨(ps, i) ∈ S⟩ ⟨¬ ¬ p on (ps, i)⟩
    show ⟨p at i in' S⟩

```

```

proof (rule ccontr)
  assume  $\langle \neg p \text{ at } i \text{ in } S \rangle$ 
  then obtain  $S' n$  where
     $\langle A, n \vdash S' \rangle$  and  $S': \langle \text{set } S' \subseteq \{(p \# ps, i)\} \cup S \rangle$  and  $\langle (p \# ps, i) \in S' \rangle$ 
    using  $\langle \text{maximal } A S \rangle$  unfolding maximal-def consistent-def
    by (metis insert-is-Un list.set-intros(1) on.simps subset-insert)
  then obtain  $S''$  where  $S''$ :
     $\langle \text{set } ((p \# ps, i) \# S'') = \text{set } S' \rangle$   $\langle (p \# ps, i) \notin \text{set } S'' \rangle$ 
    using split-list[where  $x = (p \# ps, i)$ ] by blast
  then have  $\langle A \vdash (p \# ps, i) \# S'' \rangle$ 
    using inf fin STA-struct  $\langle A, n \vdash S' \rangle$  by blast
  then have  $\langle A \vdash (ps, i) \# S'' \rangle$ 
    using ps by (meson Neg' list.set-intros(1))
  moreover have  $\langle \text{set } ((ps, i) \# S'') \subseteq S \rangle$ 
    using  $S' S'' ps$  by auto
  ultimately show False
    using  $\langle \text{consistent } A S \rangle$  unfolding consistent-def by blast
qed
next
fix  $p q i ps$ 
assume  $ps: \langle (ps, i) \in S \rangle$   $\langle (p \vee q) \text{ on } (ps, i) \rangle$  and  $*$ :  $\langle \neg q \text{ at } i \text{ in } S \rangle$ 
show  $\langle p \text{ at } i \text{ in } S \rangle$ 
proof (rule ccontr)
  assume  $\langle \neg p \text{ at } i \text{ in } S \rangle$ 
  then obtain  $Sp' np$  where
     $\langle A, np \vdash Sp' \rangle$  and  $Sp': \langle \text{set } Sp' \subseteq \{(p \# ps, i)\} \cup S \rangle$  and  $\langle (p \# ps, i) \in Sp' \rangle$ 
    using  $\langle \text{maximal } A S \rangle$  unfolding maximal-def consistent-def
    by (metis insert-is-Un list.set-intros(1) on.simps subset-insert)
  then obtain  $Sp''$  where  $Sp''$ :
     $\langle \text{set } ((p \# ps, i) \# Sp'') = \text{set } Sp' \rangle$   $\langle (p \# ps, i) \notin \text{set } Sp'' \rangle$ 
    using split-list[where  $x = (p \# ps, i)$ ] by blast
  then have  $\langle A \vdash (p \# ps, i) \# Sp'' \rangle$ 
    using  $\langle A, np \vdash Sp' \rangle$  inf fin STA-struct by blast

obtain  $Sq' nq$  where
   $\langle A, nq \vdash Sq' \rangle$  and  $Sq': \langle \text{set } Sq' \subseteq \{(q \# ps, i)\} \cup S \rangle$  and  $\langle (q \# ps, i) \in Sq' \rangle$ 
  using  $*$   $\langle \text{maximal } A S \rangle$  unfolding maximal-def consistent-def
  by (metis insert-is-Un list.set-intros(1) on.simps subset-insert)
then obtain  $Sq''$  where  $Sq''$ :
   $\langle \text{set } ((q \# ps, i) \# Sq'') = \text{set } Sq' \rangle$   $\langle (q \# ps, i) \notin \text{set } Sq'' \rangle$ 
  using split-list[where  $x = (q \# ps, i)$ ] by blast
then have  $\langle A \vdash (q \# ps, i) \# Sq'' \rangle$ 
  using  $\langle A, nq \vdash Sq' \rangle$  inf fin STA-struct by blast

obtain  $S''$  where  $S''$ :  $\langle \text{set } S'' = \text{set } Sp'' \cup \text{set } Sq'' \rangle$ 
  by (meson set-union)
then have
   $\langle \text{set } ((p \# ps, i) \# Sp'') \subseteq \text{set } ((p \# ps, i) \# S'') \rangle$ 
   $\langle \text{set } ((q \# ps, i) \# Sq'') \subseteq \text{set } ((q \# ps, i) \# S'') \rangle$ 

```

```

    by auto
  then have  $\langle A \vdash (p \# ps, i) \# S'' \rangle \langle A \vdash (q \# ps, i) \# S'' \rangle$ 
    using  $\langle A \vdash (p \# ps, i) \# Sp'' \rangle \langle A \vdash (q \# ps, i) \# Sq'' \rangle$  inf fin STA-struct
by blast+
  then have  $\langle A \vdash (ps, i) \# S'' \rangle$ 
    using ps by (meson DisP'' list.set-intros(1))
  moreover have  $\langle \text{set } ((ps, i) \# S'') \subseteq S \rangle$ 
    using ps Sp' Sp'' Sq' Sq'' S'' by auto
  ultimately show False
    using  $\langle \text{consistent } A \ S \rangle$  unfolding consistent-def by blast
qed
next
fix p q i ps
assume ps:  $\langle (ps, i) \in S \rangle \langle (\neg (p \vee q)) \text{ on } (ps, i) \rangle$ 
show  $\langle (\neg p) \text{ at } i \text{ in } S \rangle$ 
proof (rule ccontr)
  assume  $\langle \neg (\neg p) \text{ at } i \text{ in } S \rangle$ 
  then obtain S' where
     $\langle A \vdash S' \rangle$  and
    S':  $\langle \text{set } S' \subseteq \{((\neg q) \# (\neg p) \# ps, i)\} \cup S \rangle$  and
     $\langle ((\neg q) \# (\neg p) \# ps, i) \in S' \rangle$ 
    using  $\langle \text{maximal } A \ S \rangle$  unfolding maximal-def consistent-def
  by (metis (mono-tags, lifting) insert-is-Un insert-subset list.simps(15) on.simps
    set-subset-Cons subset-insert)
  then obtain S'' where S'':
     $\langle \text{set } (((\neg q) \# (\neg p) \# ps, i) \# S'') = \text{set } S' \rangle$ 
     $\langle ((\neg q) \# (\neg p) \# ps, i) \notin \text{set } S'' \rangle$ 
    using split-list[where x= $((\neg q) \# (\neg p) \# ps, i)$ ] by blast
  then have  $\langle A \vdash ((\neg q) \# (\neg p) \# ps, i) \# S'' \rangle$ 
    using inf fin STA-struct  $\langle A \vdash S' \rangle$  by blast
  then have  $\langle A \vdash (ps, i) \# S'' \rangle$ 
    using ps by (meson DisN' list.set-intros(1))
  moreover have  $\langle \text{set } ((ps, i) \# S'') \subseteq S \rangle$ 
    using S' S'' ps by auto
  ultimately show False
    using  $\langle \text{consistent } A \ S \rangle$  unfolding consistent-def by blast
qed
next
fix p q i ps
assume ps:  $\langle (ps, i) \in S \rangle \langle (\neg (p \vee q)) \text{ on } (ps, i) \rangle$ 
show  $\langle (\neg q) \text{ at } i \text{ in } S \rangle$ 
proof (rule ccontr)
  assume  $\langle \neg (\neg q) \text{ at } i \text{ in } S \rangle$ 
  then obtain S' where
     $\langle A \vdash S' \rangle$  and
    S':  $\langle \text{set } S' \subseteq \{((\neg q) \# (\neg p) \# ps, i)\} \cup S \rangle$  and
     $\langle ((\neg q) \# (\neg p) \# ps, i) \in S' \rangle$ 
    using  $\langle \text{maximal } A \ S \rangle$  unfolding maximal-def consistent-def
  by (metis (mono-tags, lifting) insert-is-Un insert-subset list.simps(15) on.simps

```

```

      set-subset-Cons subset-insert)
then obtain  $S''$  where  $S''$ :
  ⟨set ((( $\neg$   $q$ ) # ( $\neg$   $p$ ) #  $ps$ ,  $i$ ) #  $S''$ ) = set  $S'$ ⟩
  ⟨(( $\neg$   $q$ ) # ( $\neg$   $p$ ) #  $ps$ ,  $i$ )  $\notin$  set  $S''$ ⟩
  using split-list[where  $x = ((\neg q) \# (\neg p) \# ps, i)$ ] by blast
then have  $\langle A \vdash ((\neg q) \# (\neg p) \# ps, i) \# S'' \rangle$ 
  using inf fin STA-struct  $\langle A \vdash S' \rangle$  by blast
then have  $\langle A \vdash (ps, i) \# S'' \rangle$ 
  using  $ps$  by (meson DisN' list.set-intros(1))
moreover have  $\langle set ((ps, i) \# S'') \subseteq S \rangle$ 
  using  $S' S'' ps$  by auto
ultimately show False
  using  $\langle consistent A S \rangle$  unfolding consistent-def by blast
qed
next
  fix  $p i ps$ 
  assume  $\langle \nexists a. p = Nom a \rangle \langle (ps, i) \in S \rangle \langle (\diamond p) \text{ on } (ps, i) \rangle$ 
  then show  $\langle \exists j. (\diamond Nom j) \text{ at } i \text{ in}' S \wedge (@ j p) \text{ at } i \text{ in}' S \rangle$ 
  using  $\langle saturated S \rangle$  unfolding saturated-def by blast
next
  fix  $p i j ps qs$ 
  assume
     $ps: \langle (ps, i) \in S \rangle \langle (\neg (\diamond p)) \text{ on } (ps, i) \rangle$  and
     $qs: \langle (qs, i) \in S \rangle \langle (\diamond Nom j) \text{ on } (qs, i) \rangle$ 
  show  $\langle (\neg (@ j p)) \text{ at } i \text{ in}' S \rangle$ 
  proof (rule ccontr)
    assume  $\langle \neg (\neg (@ j p)) \text{ at } i \text{ in}' S \rangle$ 
    then obtain  $S' n$  where
       $\langle A, n \vdash S' \rangle$  and  $S': \langle set S' \subseteq \{([\neg (@ j p)], i)\} \cup S \rangle$  and  $\langle ([\neg (@ j p)], i) \in S' \rangle$ 
      using  $\langle maximal A S \rangle$  unfolding maximal-def consistent-def
      by (metis insert-is-Un list.set-intros(1) on.simps subset-insert)
    then obtain  $S''$  where  $S''$ :
       $\langle set (([\neg (@ j p)], i) \# S'') = set S' \rangle \langle ([\neg (@ j p)], i) \notin set S'' \rangle$ 
      using split-list[where  $x = ([\neg (@ j p)], i)$ ] by blast
    then have  $\langle A \vdash ([\neg (@ j p)], i) \# S'' \rangle$ 
      using inf fin STA-struct  $\langle A, n \vdash S' \rangle$  by blast
    then have  $\langle A \vdash ([\neg (@ j p)], i) \# (ps, i) \# (qs, i) \# S'' \rangle$ 
      using inf fin STA-struct[where  $branch' = ([\neg], -) \# (ps, i) \# (qs, i) \# S''$ ]
     $\langle A, n \vdash S' \rangle$ 
    by fastforce
    then have  $\langle A \vdash ([\neg], i) \# (ps, i) \# (qs, i) \# S'' \rangle$ 
    using  $ps(2) qs(2)$  by (meson DiaN' list.set-intros(1) set-subset-Cons subset-iff)
    moreover have  $\langle i \in branch-nominals ((ps, i) \# (qs, i) \# S'') \rangle$ 
    unfolding branch-nominals-def by simp
    ultimately have  $\langle A \vdash (ps, i) \# (qs, i) \# S'' \rangle$ 
    using GoTo by fast
    moreover have  $\langle set ((ps, i) \# (qs, i) \# S'') \subseteq S \rangle$ 
    using  $S' S'' ps qs$  by auto

```

```

ultimately show False
  using ⟨consistent A S⟩ unfolding consistent-def by blast
qed
next
fix p i ps a
assume ps: ⟨(ps, a) ∈ S⟩ ⟨(@ i p) on (ps, a)⟩
show ⟨p at i in' S⟩
proof (rule ccontr)
  assume ⟨¬ p at i in' S⟩
  then obtain S' n where
    ⟨A, n ⊢ S'⟩ and S': ⟨set S' ⊆ {[p], i} ∪ S⟩ and ⟨([p], i) ∈. S'⟩
  using ⟨maximal A S⟩ unfolding maximal-def consistent-def
  by (metis insert-is-Un list.set-intros(1) on.simps subset-insert)
  then obtain S'' where S'':
    ⟨set (([p], i) # S'') = set S'⟩ ⟨([p], i) ∉ set S''⟩
  using split-list[where x=⟨([p], i)⟩] by blast
  then have ⟨A ⊢ ([p], i) # S''⟩
  using inf fin STA-struct ⟨A, n ⊢ S'⟩ by blast
  moreover have ⟨set (([p], i) # S'') ⊆ set (([p], i) # (ps, a) # S'')⟩
  by auto
  ultimately have ⟨A ⊢ ([p], i) # (ps, a) # S''⟩
  using inf fin STA-struct ⟨A, n ⊢ S'⟩ by blast
  then have ⟨A ⊢ ([], i) # (ps, a) # S''⟩
  using ps by (metis SatP' insert-iff list.simps(15))
  moreover have ⟨i ∈ branch-nominals ((ps, a) # S'')⟩
  using ps unfolding branch-nominals-def by fastforce
  ultimately have ⟨A ⊢ (ps, a) # S''⟩
  using GoTo by fast
  moreover have ⟨set ((ps, a) # S'') ⊆ S⟩
  using S' S'' ps by auto
  ultimately show False
  using ⟨consistent A S⟩ unfolding consistent-def by blast
qed
next
fix p i ps a
assume ps: ⟨(ps, a) ∈ S⟩ ⟨(¬ (@ i p)) on (ps, a)⟩
show ⟨(¬ p) at i in' S⟩
proof (rule ccontr)
  assume ⟨¬ (¬ p) at i in' S⟩
  then obtain S' n where
    ⟨A, n ⊢ S'⟩ and S': ⟨set S' ⊆ {[¬ p], i} ∪ S⟩ and ⟨([¬ p], i) ∈. S'⟩
  using ⟨maximal A S⟩ unfolding maximal-def consistent-def
  by (metis insert-is-Un list.set-intros(1) on.simps subset-insert)
  then obtain S'' where S'':
    ⟨set (([¬ p], i) # S'') = set S'⟩ ⟨([¬ p], i) ∉ set S''⟩
  using split-list[where x=⟨([¬ p], i)⟩] by blast
  then have ⟨A ⊢ ([¬ p], i) # S''⟩
  using inf fin STA-struct ⟨A, n ⊢ S'⟩ by blast
  then have ⟨A ⊢ ([¬ p], i) # (ps, a) # S''⟩

```

```

    using inf fin STA-struct[where branch'= $\langle(\neg p), i\rangle \# - \# S''\rangle$ ]  $\langle A, n \vdash S'\rangle$ 
    by fastforce
  then have  $\langle A \vdash ([], i) \# (ps, a) \# S''\rangle$ 
    using ps by (metis SatN' insert-iff list.simps(15))
  moreover have  $\langle i \in \text{branch-nominals } ((ps, a) \# S'')\rangle$ 
    using ps unfolding branch-nominals-def by fastforce
  ultimately have  $\langle A \vdash (ps, a) \# S''\rangle$ 
    using GoTo by fast
  moreover have  $\langle \text{set } ((ps, a) \# S'') \subseteq S\rangle$ 
    using S' S'' ps by auto
  ultimately show False
    using  $\langle \text{consistent } A S\rangle$  unfolding consistent-def by blast
qed
next
fix p i ps a
assume i:  $\langle i \in \text{nominals } p\rangle$  and ps:  $\langle (ps, a) \in S\rangle$   $\langle p \text{ on } (ps, a)\rangle$ 
show  $\langle \exists qs. (qs, i) \in S\rangle$ 
proof (rule ccontr)
  assume  $\nexists qs. (qs, i) \in S$ 
  then obtain S' n where
     $\langle A, n \vdash S'\rangle$  and S':  $\langle \text{set } S' \subseteq \{([], i)\} \cup S\rangle$  and  $\langle ([], i) \in S'\rangle$ 
    using  $\langle \text{maximal } A S\rangle$  unfolding maximal-def consistent-def
    by (metis insert-is-Un subset-insert)
  then obtain S'' where S'':
     $\langle \text{set } (([], i) \# S'') = \text{set } S'\rangle$   $\langle ([], i) \notin \text{set } S''\rangle$ 
    using split-list[where x= $\langle ([], i)\rangle$ ] by blast
  then have  $\langle A \vdash ([], i) \# (ps, a) \# S''\rangle$ 
    using inf fin STA-struct[where branch'= $\langle([], i) \# (ps, a) \# S''\rangle$ ]  $\langle A, n \vdash S'\rangle$ 
    by fastforce
  moreover have  $\langle i \in \text{branch-nominals } ((ps, a) \# S'')\rangle$ 
    using i ps unfolding branch-nominals-def by auto
  ultimately have  $\langle A \vdash (ps, a) \# S''\rangle$ 
    using GoTo by fast
  moreover have  $\langle \text{set } ((ps, a) \# S'') \subseteq S\rangle$ 
    using S' S'' ps by auto
  ultimately show False
    using  $\langle \text{consistent } A S\rangle$  unfolding consistent-def by blast
qed
next
fix p i j ps qs
assume
  p:  $\langle \forall a. p = \text{Nom } a \vee p = (\Diamond \text{Nom } a) \longrightarrow a \in A\rangle$  and
  ps:  $\langle (ps, i) \in S\rangle$   $\langle p \text{ on } (ps, i)\rangle$  and
  qs:  $\langle (qs, i) \in S\rangle$   $\langle \text{Nom } j \text{ on } (qs, i)\rangle$ 

show  $\langle p \text{ at } j \text{ in } S\rangle$ 
proof (rule ccontr)
  assume  $\nexists rs. (rs, j) \in S \wedge p \text{ on } (rs, j)$ 
  then obtain S' n where

```

$\langle A, n \vdash S' \rangle$  and  $S'$ :  $\langle \text{set } S' \subseteq \{([p], j)\} \cup S \rangle$  and  $\langle ([p], j) \in S' \rangle$   
**using**  $\langle \text{maximal } A \ S \rangle$  **unfolding** *maximal-def consistent-def*  
**by** *(metis insert-is-Un list.set-intros(1) on.simps subset-insert)*  
**then obtain**  $S''$  **where**  $S''$ :  
 $\langle \text{set } (\{([p], j) \} \# S'') = \text{set } S' \rangle$   $\langle ([p], j) \notin \text{set } S'' \rangle$   
**using** *split-list[where x= $\langle ([p], j) \rangle$ ]* **by** *blast*  
**then have**  $\langle A \vdash ([p], j) \# S'' \rangle$   
**using** *inf fin STA-struct*  $\langle A, n \vdash S' \rangle$  **by** *blast*  
**then have**  $\langle A \vdash ([p], j) \# (ps, i) \# (qs, i) \# S'' \rangle$   
**using** *inf fin STA-struct[where branch'= $\langle ([-], -) \# (ps, i) \# (qs, i) \# S'' \rangle$ ]*  
 $\langle A, n \vdash S' \rangle$   
**by** *fastforce*  
**then have**  $\langle A \vdash ([], j) \# (ps, i) \# (qs, i) \# S'' \rangle$   
**using** *ps(2) qs(2) p* **by** *(meson Nom' in-mono list.set-intros(1) set-subset-Cons)*  
**moreover have**  $\langle j \in \text{branch-nominals } ((ps, i) \# (qs, i) \# S'') \rangle$   
**using** *qs(2) unfolding branch-nominals-def* **by** *fastforce*  
**ultimately have**  $\langle A \vdash (ps, i) \# (qs, i) \# S'' \rangle$   
**using** *GoTo* **by** *fast*  
**moreover have**  $\langle \text{set } ((ps, i) \# (qs, i) \# S'') \subseteq S \rangle$   
**using** *S' S'' ps qs* **by** *auto*  
**ultimately show** *False*  
**using**  $\langle \text{consistent } A \ S \rangle$  **unfolding** *consistent-def* **by** *blast*  
**qed**  
**qed**

## 11.4 Result

**theorem** *completeness*:

**fixes**  $p :: \langle 'a :: \text{countable}, 'b :: \text{countable} \rangle \text{ fm}$

**assumes**

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

*valid*:  $\langle \forall (M :: ('b \text{ set}, 'a) \text{ model}) \ g \ w. \ M, g, w \models p \rangle$

**shows**  $\langle \text{nominals } p, 1 \vdash [(\neg p), i] \rangle$

**proof** –

**let**  $?A = \langle \text{nominals } p \rangle$

**have**  $\langle ?A \vdash [(\neg p), i] \rangle$

**proof** *(rule ccontr)*

**assume**  $\langle \neg ?A \vdash [(\neg p), i] \rangle$

**moreover have**  $\langle \text{finite } ?A \rangle$

**using** *finite-nominals* **by** *blast*

**ultimately have**  $*$ :  $\langle \text{consistent } ?A \{[(\neg p), i]\} \rangle$

**unfolding** *consistent-def* **using** *STA-struct inf*

**by** *(metis empty-set list.simps(15))*

**let**  $?S = \langle \text{Extend } ?A \{[(\neg p), i]\} \text{ from-nat} \rangle$

**have**  $\langle \text{finite } \{[(\neg p), i]\} \rangle$

**by** *simp*

**then have** *fin*:  $\langle \text{finite } (\bigcup (\text{block-nominals } ' \{[(\neg p), i]\})) \rangle$

```

using finite-nominals-set by blast

have  $\langle \text{consistent } ?A \ ?S \rangle$ 
  using consistent-Extend inf * fin  $\langle \text{finite } ?A \rangle$  by blast
moreover have  $\langle \text{maximal } ?A \ ?S \rangle$ 
  using maximal-Extend inf * fin by fastforce
moreover have  $\langle \text{saturated } ?S \rangle$ 
  using saturated-Extend inf * fin by fastforce
ultimately have  $\langle \text{Hintikka } ?A \ ?S \rangle$ 
  using Hintikka-Extend inf  $\langle \text{finite } ?A \rangle$  by blast
moreover have  $\langle ([\neg p], i) \in ?S \rangle$ 
  using Extend-mem by blast
moreover have  $\langle (\neg p) \text{ on } ([\neg p], i) \rangle$ 
  by simp
ultimately have  $\langle \neg \text{Model } (\text{reach } ?A \ ?S) (\text{val } ?S), \text{assign } ?A \ ?S, \text{assign } ?A \ ?S \ i \models p \rangle$ 
  using Hintikka-model(2) by fast
then show False
  using valid by blast
qed
then show ?thesis
  using STA-one by fast
qed

```

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

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

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  $\langle i \notin \text{nominals } p \rangle$ 
shows  $\langle \text{valid } p \longleftrightarrow \text{nominals } p, 1 \vdash [([\neg p], i)] \rangle$ 
proof
  assume  $\langle \text{valid } p \rangle$ 
  then show  $\langle \text{nominals } p, 1 \vdash [([\neg p], i)] \rangle$ 
    using completeness by blast
next
  assume  $\langle \text{nominals } p, 1 \vdash [([\neg p], i)] \rangle$ 
  then show  $\langle \text{valid } p \rangle$ 
    using assms soundness-fresh by fast
qed

```

The restricted validity implies validity in general.



**theorem** *valid-semantics*:  
 $\langle \text{valid } p \longrightarrow M, g, w \models p \rangle$   
**proof**  
**assume**  $\langle \text{valid } p \rangle$   
**then have**  $\langle i \notin \text{nominals } p \implies \text{nominals } p \vdash [(\neg p), i] \rangle$  **for**  $i$   
**using** *main* **by** *blast*  
**moreover have**  $\langle \exists i. i \notin \text{nominals } p \rangle$   
**by** (*simp add: finite-nominals ex-new-if-finite*)  
**ultimately show**  $\langle M, g, w \models p \rangle$   
**using** *soundness-fresh* **by** *fast*  
**qed**  
**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.