

# A Sequent Calculus for First-Order Logic

Asta Halkjær From

September 13, 2023

## Abstract

This work formalizes soundness and completeness of a one-sided sequent calculus for first-order logic. The completeness is shown via a translation from a complete semantic tableau calculus, the proof of which is based on the First-Order Logic According to Fitting theory. The calculi and proof techniques are taken from Ben-Ari's Mathematical Logic for Computer Science [1].

## Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Common Notation</b>                     | <b>2</b>  |
| <b>2</b> | <b>Tableau Calculus</b>                    | <b>2</b>  |
| 2.1      | Soundness . . . . .                        | 3         |
| 2.2      | Completeness for Closed Formulas . . . . . | 4         |
| 2.3      | Open Formulas . . . . .                    | 8         |
| 2.4      | Completeness . . . . .                     | 19        |
| <b>3</b> | <b>Sequent Calculus</b>                    | <b>20</b> |
| 3.1      | Soundness . . . . .                        | 21        |
| 3.2      | Tableau Calculus Equivalence . . . . .     | 22        |
| 3.3      | Completeness . . . . .                     | 23        |
| <b>4</b> | <b>Completeness Revisited</b>              | <b>24</b> |

# 1 Common Notation

**theory** *Common* **imports** *FOL-Fitting.FOL-Fitting* **begin**

**notation** *FF* ( $\langle \perp \rangle$ )

**notation** *TT* ( $\langle \top \rangle$ )

**end**

# 2 Tableau Calculus

**theory** *Tableau* **imports** *Common* **begin**

**inductive** *TC* ::  $\langle ('a, 'b) \text{ form list} \Rightarrow \text{bool} \rangle$  ( $\langle \neg \rightarrow 0 \rangle$ ) **where**

*Basic*:  $\langle \neg \text{Pred } i \ l \ \# \ \text{Neg } (\text{Pred } i \ l) \ \# \ G \rangle$

| *BasicFF*:  $\langle \neg \perp \ \# \ G \rangle$

| *BasicNegTT*:  $\langle \neg \text{Neg } \top \ \# \ G \rangle$

| *AlphaNegNeg*:  $\langle \neg A \ \# \ G \Longrightarrow \neg \text{Neg } (\text{Neg } A) \ \# \ G \rangle$

| *AlphaAnd*:  $\langle \neg A \ \# \ B \ \# \ G \Longrightarrow \neg \text{And } A \ B \ \# \ G \rangle$

| *AlphaNegOr*:  $\langle \neg \text{Neg } A \ \# \ \text{Neg } B \ \# \ G \Longrightarrow \neg \text{Neg } (\text{Or } A \ B) \ \# \ G \rangle$

| *AlphaNegImpl*:  $\langle \neg A \ \# \ \text{Neg } B \ \# \ G \Longrightarrow \neg \text{Neg } (\text{Impl } A \ B) \ \# \ G \rangle$

| *BetaNegAnd*:  $\langle \neg \text{Neg } A \ \# \ G \Longrightarrow \neg \text{Neg } B \ \# \ G \Longrightarrow \neg \text{Neg } (\text{And } A \ B) \ \# \ G \rangle$

| *BetaOr*:  $\langle \neg A \ \# \ G \Longrightarrow \neg B \ \# \ G \Longrightarrow \neg \text{Or } A \ B \ \# \ G \rangle$

| *BetaImpl*:  $\langle \neg \text{Neg } A \ \# \ G \Longrightarrow \neg B \ \# \ G \Longrightarrow \neg \text{Impl } A \ B \ \# \ G \rangle$

| *GammaForall*:  $\langle \neg \text{subst } A \ t \ 0 \ \# \ G \Longrightarrow \neg \text{Forall } A \ \# \ G \rangle$

| *GammaNegExists*:  $\langle \neg \text{Neg } (\text{subst } A \ t \ 0) \ \# \ G \Longrightarrow \neg \text{Neg } (\text{Exists } A) \ \# \ G \rangle$

| *DeltaExists*:  $\langle \neg \text{subst } A \ (\text{App } n \ []) \ 0 \ \# \ G \Longrightarrow \text{news } n \ (A \ \# \ G) \Longrightarrow \neg \text{Exists } A \ \# \ G \rangle$

| *DeltaNegForall*:  $\langle \neg \text{Neg } (\text{subst } A \ (\text{App } n \ []) \ 0) \ \# \ G \Longrightarrow \text{news } n \ (A \ \# \ G) \Longrightarrow \neg \text{Neg } (\text{Forall } A) \ \# \ G \rangle$

| *Order*:  $\langle \neg G \Longrightarrow \text{set } G = \text{set } G' \Longrightarrow \neg G' \rangle$

**lemma** *Shift*:  $\langle \neg \text{rotate1 } G \Longrightarrow \neg G \rangle$

**by** (*simp add: Order*)

**lemma** *Swap*:  $\langle \neg B \ \# \ A \ \# \ G \Longrightarrow \neg A \ \# \ B \ \# \ G \rangle$

**by** (*simp add: Order insert-commute*)

**definition** *tableauproof* ::  $\langle ('a, 'b) \text{ form list} \Rightarrow ('a, 'b) \text{ form} \Rightarrow \text{bool} \rangle$  **where**

$\langle \text{tableauproof } ps \ p \equiv (\neg \text{Neg } p \ \# \ ps) \rangle$

**theorem** *tableauNotAA*:  $\langle \neg [\text{Neg } (\text{Pred } "A" \ []), \text{Pred } "A" \ []] \rangle$

**by** (*rule Shift, simp*) (*rule Basic*)

**theorem** *AndAnd*:

$\langle \neg [\text{And } (\text{Pred } "A" \ []) \ (\text{Pred } "B" \ []), \text{Neg } (\text{And } (\text{Pred } "B" \ []) \ (\text{Pred } "A" \ []))] \rangle$

**apply** (*rule AlphaAnd*)

**apply** (*rule Shift, rule Shift, simp*)

**apply** (*rule BetaNegAnd*)

```

apply (rule Shift, rule Shift, simp)
apply (rule Basic)
apply (rule Swap)
apply (rule Basic)
done

```

## 2.1 Soundness

**lemma** *TC-soundness*:

```

⟨¬ G ⟹ ∃ p ∈ set G. ¬ eval e f g p⟩
proof (induct G arbitrary: f rule: TC.induct)
case (DeltaExists A n G)
show ?case
proof (rule ccontr)
  assume ⟨¬ (∃ p ∈ set (Exists A # G). ¬ eval e f g p)⟩
  then have *: ⟨∀ p ∈ set (Exists A # G). eval e f g p⟩
    by simp

  then obtain x where ⟨eval (shift e 0 x) (f(n := λw. x)) g A⟩
  using ⟨news n (A # G)⟩ by auto
  then have **: ⟨eval e (f(n := λw. x)) g (subst A (App n []) 0)⟩
    by simp

  have ⟨∃ p ∈ set (subst A (App n []) 0 # G). ¬ eval e (f(n := λw. x)) g p⟩
  using DeltaExists by fast
  then consider
    ⟨¬ eval e (f(n := λw. x)) g (subst A (App n []) 0)⟩ |
    ⟨∃ p ∈ set G. ¬ eval e (f(n := λw. x)) g p⟩
  by auto
  then show False
  proof cases
    case 1
      then show ?thesis
      using ** ..
    next
      case 2
      then obtain p where ⟨¬ eval e (f(n := λw. x)) g p⟩ ⟨p ∈ set G⟩
      by blast
      then have ⟨¬ eval e f g p⟩
      using ⟨news n (A # G)⟩ by (metis Ball-set set-subset-Cons subsetCE
upd-lemma)
      then show ?thesis
      using * ⟨p ∈ set G⟩ by simp
    qed
  qed
next
case (DeltaNegForall A n G)
show ?case
proof (rule ccontr)

```

```

assume  $\langle \neg (\exists p \in \text{set } (\text{Neg } (\text{Forall } A) \# G). \neg \text{eval } e f g p) \rangle$ 
then have *:  $\langle \forall p \in \text{set } (\text{Neg } (\text{Forall } A) \# G). \text{eval } e f g p \rangle$ 
  by simp

then obtain x where  $\langle \text{eval } (\text{shift } e 0 x) (f(n := \lambda w. x)) g (\text{Neg } A) \rangle$ 
  using  $\langle \text{news } n (A \# G) \rangle$  by auto
then have **:  $\langle \text{eval } e (f(n := \lambda w. x)) g (\text{Neg } (\text{subst } A (\text{App } n [])) 0) \rangle$ 
  by simp

have  $\langle \exists p \in \text{set } (\text{Neg } (\text{subst } A (\text{App } n [])) 0) \# G). \neg \text{eval } e (f(n := \lambda w. x)) g$ 
p  $\rangle$ 
  using DeltaNegForall by fast
then consider
   $\langle \neg \text{eval } e (f(n := \lambda w. x)) g (\text{Neg } (\text{subst } A (\text{App } n [])) 0) \rangle$  |
   $\langle \exists p \in \text{set } G. \neg \text{eval } e (f(n := \lambda w. x)) g p \rangle$ 
  by auto
then show False
proof cases
  case 1
  then show ?thesis
    using ** ..
  next
  case 2
  then obtain p where  $\langle \neg \text{eval } e (f(n := \lambda w. x)) g p \rangle$   $\langle p \in \text{set } G \rangle$ 
  by blast
  then have  $\langle \neg \text{eval } e f g p \rangle$ 
    using  $\langle \text{news } n (A \# G) \rangle$  by (metis Ball-set set-subset-Cons subsetCE
upd-lemma)
  then show ?thesis
    using *  $\langle p \in \text{set } G \rangle$  by simp
  qed
qed
qed auto

```

**theorem** *tableau-soundness*:

```

 $\langle \text{tableauproof } ps p \implies \text{list-all } (\text{eval } e f g) ps \implies \text{eval } e f g p \rangle$ 
using TC-soundness unfolding tableauproof-def list-all-def by fastforce

```

## 2.2 Completeness for Closed Formulas

**theorem** *infinite-nonempty*:  $\langle \text{infinite } A \implies \exists x. x \in A \rangle$

**by** (*simp add: ex-in-conv infinite-imp-nonempty*)

**theorem** *TCd-consistency*:

**assumes** *inf-param*:  $\langle \text{infinite } (\text{UNIV}::'a \text{ set}) \rangle$

**shows**  $\langle \text{consistency } \{S::('a, 'b) \text{ form set}. \exists G. S = \text{set } G \wedge \neg (\vdash G)\} \rangle$

**unfolding** *consistency-def*

**proof** (*intro conjI allI impI notI*)

**fix** *S* ::  $\langle ('a, 'b) \text{ form set} \rangle$

```

assume  $\langle S \in \{set\ G \mid G. \neg (\neg G)\} \rangle$  (is  $\langle S \in ?C \rangle$ )
then obtain  $G :: \langle 'a, 'b \rangle$  form list
  where *:  $\langle S = set\ G \rangle$  and  $\langle \neg (\neg G) \rangle$ 
  by blast

{ fix  $p\ ts$ 
  assume  $\langle Pred\ p\ ts \in S \wedge Neg\ (Pred\ p\ ts) \in S \rangle$ 
  then show False
  using * Basic Order  $\langle \neg (\neg G) \rangle$  by fastforce }

{ assume  $\langle \perp \in S \rangle$ 
  then show False
  using * BasicFF Order  $\langle \neg (\neg G) \rangle$  by fastforce }

{ assume  $\langle Neg\ \top \in S \rangle$ 
  then show False
  using * BasicNegTT Order  $\langle \neg (\neg G) \rangle$  by fastforce }

{ fix  $Z$ 
  assume  $\langle Neg\ (Neg\ Z) \in S \rangle$ 
  then have  $\langle \neg (\neg Z \# G) \rangle$ 
    using * AlphaNegNeg Order  $\langle \neg (\neg G) \rangle$ 
    by (metis insert-absorb list.set(2))
  moreover have  $\langle S \cup \{Z\} = set\ (Z \# G) \rangle$ 
    using * by simp
  ultimately show  $\langle S \cup \{Z\} \in ?C \rangle$ 
  by blast }

{ fix  $A\ B$ 
  assume  $\langle And\ A\ B \in S \rangle$ 
  then have  $\langle \neg (\neg A \# B \# G) \rangle$ 
    using * AlphaAnd Order  $\langle \neg (\neg G) \rangle$ 
    by (metis insert-absorb list.set(2))
  moreover have  $\langle S \cup \{A, B\} = set\ (A \# B \# G) \rangle$ 
    using * by simp
  ultimately show  $\langle S \cup \{A, B\} \in ?C \rangle$ 
  by blast }

{ fix  $A\ B$ 
  assume  $\langle Neg\ (Or\ A\ B) \in S \rangle$ 
  then have  $\langle \neg (\neg Neg\ A \# Neg\ B \# G) \rangle$ 
    using * AlphaNegOr Order  $\langle \neg (\neg G) \rangle$ 
    by (metis insert-absorb list.set(2))
  moreover have  $\langle S \cup \{Neg\ A, Neg\ B\} = set\ (Neg\ A \# Neg\ B \# G) \rangle$ 
    using * by simp
  ultimately show  $\langle S \cup \{Neg\ A, Neg\ B\} \in ?C \rangle$ 
  by blast }

{ fix  $A\ B$ 

```

```

assume  $\langle \text{Neg } (\text{Impl } A B) \in S \rangle$ 
then have  $\langle \neg (\neg A \# \text{Neg } B \# G) \rangle$ 
  using * AlphaNegImpl Order  $\langle \neg (\neg G) \rangle$ 
  by (metis insert-absorb list.set(2))
moreover have  $\langle \{A, \text{Neg } B\} \cup S = \text{set } (A \# \text{Neg } B \# G) \rangle$ 
  using * by simp
ultimately show  $\langle S \cup \{A, \text{Neg } B\} \in ?C \rangle$ 
  by blast }

{ fix A B
  assume  $\langle \text{Or } A B \in S \rangle$ 
  then have  $\langle \neg (\neg A \# G) \vee \neg (\neg B \# G) \rangle$ 
    using * BetaOr Order  $\langle \neg (\neg G) \rangle$ 
    by (metis insert-absorb list.set(2))
  then show  $\langle S \cup \{A\} \in ?C \vee S \cup \{B\} \in ?C \rangle$ 
    using * by auto }

{ fix A B
  assume  $\langle \text{Neg } (\text{And } A B) \in S \rangle$ 
  then have  $\langle \neg (\neg \text{Neg } A \# G) \vee \neg (\neg \text{Neg } B \# G) \rangle$ 
    using * BetaNegAnd Order  $\langle \neg (\neg G) \rangle$ 
    by (metis insert-absorb list.set(2))
  then show  $\langle S \cup \{\text{Neg } A\} \in ?C \vee S \cup \{\text{Neg } B\} \in ?C \rangle$ 
    using * by auto }

{ fix A B
  assume  $\langle \text{Impl } A B \in S \rangle$ 
  then have  $\langle \neg (\neg \text{Neg } A \# G) \vee \neg (\neg B \# G) \rangle$ 
    using * BetaImpl Order  $\langle \neg (\neg G) \rangle$ 
    by (metis insert-absorb list.set(2))
  then show  $\langle S \cup \{\text{Neg } A\} \in ?C \vee S \cup \{B\} \in ?C \rangle$ 
    using * by auto }

{ fix P and t :: 'a term
  assume  $\langle \text{Forall } P \in S \rangle$ 
  then have  $\langle \neg (\neg \text{subst } P t 0 \# G) \rangle$ 
    using * GammaForall Order  $\langle \neg (\neg G) \rangle$ 
    by (metis insert-absorb list.set(2))
  moreover have  $\langle S \cup \{\text{subst } P t 0\} = \text{set } (\text{subst } P t 0 \# G) \rangle$ 
    using * by simp
  ultimately show  $\langle S \cup \{\text{subst } P t 0\} \in ?C \rangle$ 
    by blast }

{ fix P and t :: 'a term
  assume  $\langle \text{Neg } (\text{Exists } P) \in S \rangle$ 
  then have  $\langle \neg (\neg \text{Neg } (\text{subst } P t 0) \# G) \rangle$ 
    using * GammaNegExists Order  $\langle \neg (\neg G) \rangle$ 
    by (metis insert-absorb list.set(2))
  moreover have  $\langle S \cup \{\text{Neg } (\text{subst } P t 0)\} = \text{set } (\text{Neg } (\text{subst } P t 0) \# G) \rangle$ 

```

```

    using * by simp
    ultimately show  $\langle S \cup \{Neg (subst P t 0)\} \in ?C \rangle$ 
    by blast }

{ fix P
  assume  $\langle Exists P \in S \rangle$ 
  have  $\langle finite ((\bigcup p \in set G. params p) \cup params P) \rangle$ 
  by simp
  then have  $\langle infinite (- ((\bigcup p \in set G. params p) \cup params P)) \rangle$ 
  using inf-param Diff-infinite-finite finite-compl infinite-UNIV-listI by blast
  then obtain x where **:  $\langle x \in - ((\bigcup p \in set G. params p) \cup params P) \rangle$ 
  using infinite-imp-nonempty by blast
  then have  $\langle news x (P \# G) \rangle$ 
  using Ball-set-list-all by auto
  then have  $\langle \neg (\neg subst P (App x []) 0 \# G) \rangle$ 
  using *  $\langle Exists P \in S \rangle$  Order DeltaExists  $\langle \neg (\neg G) \rangle$ 
  by (metis insert-absorb list.set(2))
  moreover have  $\langle S \cup \{subst P (App x []) 0\} = set (subst P (App x []) 0 \# G) \rangle$ 
  using * by simp
  ultimately show  $\langle \exists x. S \cup \{subst P (App x []) 0\} \in ?C \rangle$ 
  by blast }

{ fix P
  assume  $\langle Neg (Forall P) \in S \rangle$ 
  have  $\langle finite ((\bigcup p \in set G. params p) \cup params P) \rangle$ 
  by simp
  then have  $\langle infinite (- ((\bigcup p \in set G. params p) \cup params P)) \rangle$ 
  using inf-param Diff-infinite-finite finite-compl infinite-UNIV-listI by blast
  then obtain x where **:  $\langle x \in - ((\bigcup p \in set G. params p) \cup params P) \rangle$ 
  using infinite-imp-nonempty by blast
  then have  $\langle news x (P \# G) \rangle$ 
  using Ball-set-list-all by auto
  then have  $\langle \neg (\neg Neg (subst P (App x []) 0) \# G) \rangle$ 
  using *  $\langle Neg (Forall P) \in S \rangle$  Order DeltaNegForall  $\langle \neg (\neg G) \rangle$ 
  by (metis insert-absorb list.set(2))
  moreover have  $\langle S \cup \{Neg (subst P (App x []) 0)\} = set (Neg (subst P (App x []) 0) \# G) \rangle$ 
  using * by simp
  ultimately show  $\langle \exists x. S \cup \{Neg (subst P (App x []) 0)\} \in ?C \rangle$ 
  by blast }
qed

theorem tableau-completeness':
  fixes p ::  $\langle (nat, nat) form \rangle$ 
  assumes  $\langle closed 0 p \rangle$ 
  and  $\langle list-all (closed 0) ps \rangle$ 
  and mod:  $\langle \forall (e :: nat \Rightarrow nat hterm) f g. list-all (eval e f g) ps \longrightarrow eval e f g p \rangle$ 
  shows  $\langle \text{tableauproof } ps p \rangle$ 

```

```

proof (rule ccontr)
  fix e
  assume  $\langle \neg \text{tableauproof } ps \ p \rangle$ 

  let ?S =  $\langle \text{set } (Neg \ p \ \# \ ps) \rangle$ 
  let ?C =  $\langle \{ \text{set } (G \ :: \ (nat, \ nat) \ \text{form } list) \mid G. \ \neg \ (\neg \ G) \} \rangle$ 
  let ?f = HApp
  let ?g =  $\langle (\lambda a \ ts. \ \text{Pred } a \ (\text{terms-of-hterms } ts) \in \text{Extend } ?S$ 
     $\ (\text{mk-finite-char } (\text{mk-alt-consistency } (\text{close } ?C))) \ \text{from-nat}) \rangle$ 

  from  $\langle \text{list-all } (\text{closed } 0) \ ps \rangle$ 
  have  $\langle \forall p \in \text{set } ps. \ \text{closed } 0 \ p \rangle$ 
  by (simp add: list-all-iff)

  { fix x
    assume  $\langle x \in ?S \rangle$ 
    moreover have  $\langle \text{consistency } ?C \rangle$ 
    using TCd-consistency by blast
    moreover have  $\langle ?S \in ?C \rangle$ 
    using  $\langle \neg \text{tableauproof } ps \ p \rangle$  unfolding tableauproof-def by blast
    moreover have  $\langle \text{infinite } (\neg \ (\bigcup p \in ?S. \ \text{params } p)) \rangle$ 
    by (simp add: Compl-eq-Diff-UNIV infinite-UNIV-listI)
    moreover note  $\langle \text{closed } 0 \ p \rangle \langle \forall p \in \text{set } ps. \ \text{closed } 0 \ p \rangle \langle x \in ?S \rangle$ 
    then have  $\langle \text{closed } 0 \ x \rangle$  by auto
    ultimately have  $\langle \text{eval } e \ ?f \ ?g \ x \rangle$ 
    using model-existence by blast }
  then have  $\langle \text{list-all } (\text{eval } e \ ?f \ ?g) \ (Neg \ p \ \# \ ps) \rangle$ 
  by (simp add: list-all-iff)
  moreover have  $\langle \text{eval } e \ ?f \ ?g \ (Neg \ p) \rangle$ 
  using calculation by simp
  moreover have  $\langle \text{list-all } (\text{eval } e \ ?f \ ?g) \ ps \rangle$ 
  using calculation by simp
  then have  $\langle \text{eval } e \ ?f \ ?g \ p \rangle$ 
  using mod by blast
  ultimately show False by simp
qed

```

## 2.3 Open Formulas

```

lemma TC-psubst:
  fixes f ::  $\langle 'a \Rightarrow 'a \rangle$ 
  assumes inf-params:  $\langle \text{infinite } (UNIV \ :: \ 'a \ \text{set}) \rangle$ 
  shows  $\langle \neg \ G \Longrightarrow \neg \ \text{map } (\text{psubst } f) \ G \rangle$ 
proof (induct G arbitrary: f rule: TC.induct)
  case (DeltaExists A n G)
  let ?params =  $\langle \text{params } A \cup (\bigcup p \in \text{set } G. \ \text{params } p) \rangle$ 

  have  $\langle \text{finite } ?params \rangle$ 
  by simp

```



```

then obtain fresh where *:  $\langle \text{fresh} \notin ?\text{params} \cup \{n\} \cup \text{image } f ?\text{params} \rangle$ 
  using ex-new-if-finite inf-params
  by (metis finite.emptyI finite.insertI finite-UnI finite-imageI)

let ?f =  $\langle f(n := \text{fresh}) \rangle$ 

have  $\langle \text{news } n (A \# G) \rangle$ 
  using DeltaExists by blast
then have  $\langle \text{new fresh } (p\text{subst } ?f A) \rangle \langle \text{news fresh } (map (p\text{subst } ?f) G) \rangle$ 
  using * new-psubst-image news-psubst by (fastforce simp add: image-Un)+
then have G:  $\langle map (p\text{subst } ?f) G = map (p\text{subst } f) G \rangle$ 
  using DeltaExists
  by (metis (mono-tags, lifting) Ball-set insertCI list.set(2) map-eq-conv psubst-upd)

have  $\langle \neg p\text{subst } ?f (subst A (App n []) 0) \# map (p\text{subst } ?f) G \rangle$ 
  using DeltaExists by (metis list.simps(9))
then have  $\langle \neg subst (p\text{subst } ?f A) (App \text{fresh} []) 0 \# map (p\text{subst } ?f) G \rangle$ 
  by simp
moreover have  $\langle \text{news fresh } (map (p\text{subst } ?f) (A \# G)) \rangle$ 
  using  $\langle \text{new fresh } (p\text{subst } ?f A) \rangle \langle \text{news fresh } (map (p\text{subst } ?f) G) \rangle$  by simp
then have  $\langle \text{news fresh } (p\text{subst } ?f A \# map (p\text{subst } ?f) G) \rangle$ 
  by simp
ultimately have  $\langle \neg map (p\text{subst } ?f) (Exists A \# G) \rangle$ 
  using TC.DeltaExists by fastforce
then show ?case
  using DeltaExists G by simp
next
case (DeltaNegForall A n G)
let ?params =  $\langle \text{params } A \cup (\bigcup p \in \text{set } G. \text{params } p) \rangle$ 

have  $\langle \text{finite } ?\text{params} \rangle$ 
  by simp
then obtain fresh where *:  $\langle \text{fresh} \notin ?\text{params} \cup \{n\} \cup \text{image } f ?\text{params} \rangle$ 
  using ex-new-if-finite inf-params
  by (metis finite.emptyI finite.insertI finite-UnI finite-imageI)

let ?f =  $\langle f(n := \text{fresh}) \rangle$ 

have  $\langle \text{news } n (A \# G) \rangle$ 
  using DeltaNegForall by blast
then have  $\langle \text{new fresh } (p\text{subst } ?f A) \rangle \langle \text{news fresh } (map (p\text{subst } ?f) G) \rangle$ 
  using * new-psubst-image news-psubst by (fastforce simp add: image-Un)+
then have G:  $\langle map (p\text{subst } ?f) G = map (p\text{subst } f) G \rangle$ 
  using DeltaNegForall
  by (metis (mono-tags, lifting) Ball-set insertCI list.set(2) map-eq-conv psubst-upd)

have  $\langle \neg p\text{subst } ?f (Neg (subst A (App n []) 0)) \# map (p\text{subst } ?f) G \rangle$ 
  using DeltaNegForall by (metis list.simps(9))
then have  $\langle \neg Neg (subst (p\text{subst } ?f A) (App \text{fresh} []) 0) \# map (p\text{subst } ?f) G \rangle$ 

```

```

    by simp
  moreover have ⟨news fresh (map (psubst ?f) (A # G))⟩
    using ⟨new fresh (psubst ?f A)⟩ ⟨news fresh (map (psubst ?f) G)⟩ by simp
  then have ⟨news fresh (psubst ?f A # map (psubst ?f) G)⟩
    by simp
  ultimately have ⟨¬ map (psubst ?f) (Neg (Forall A) # G)⟩
    using TC.DeltaNegForall by fastforce
  then show ?case
    using DeltaNegForall G by simp
next
  case (Order G G')
  then show ?case
    using Order TC.Order set-map by metis
qed (auto intro: TC.intros)

lemma subcs-map: ⟨subcs c s G = map (subc c s) G⟩
  by (induct G) simp-all

lemma TC-subcs:
  fixes G :: ⟨'a, 'b⟩ form list
  assumes inf-params: ⟨infinite (UNIV :: 'a set)⟩
  shows ⟨¬ G ⟹ ¬ subcs c s G⟩
proof (induct G arbitrary: c s rule: TC.induct)
  case (GammaForall A t G)
  let ?params = ⟨params A ∪ (⋃ p ∈ set G. params p) ∪ paramst s ∪ paramst t ∪ {c}⟩

  have ⟨finite ?params⟩
    by simp
  then obtain fresh where fresh: ⟨fresh ∉ ?params⟩
    using ex-new-if-finite inf-params by metis

  let ?f = ⟨id(c := fresh)⟩
  let ?g = ⟨id(fresh := c)⟩
  let ?s = ⟨psubst ?f s⟩

  have s: ⟨psubst ?g ?s = s⟩
    using fresh psubst-new-away' by simp
  have ⟨subc (?g c) (psubst ?g ?s) (psubst ?g (Forall A)) = subc c s (Forall A)⟩
    using fresh by simp
  then have A: ⟨psubst ?g (subc c ?s (Forall A)) = subc c s (Forall A)⟩
    using fun-upd-apply id-def subc-psubst UnCI fresh params.simps(8) by metis
  have ⟨∀ x ∈ (⋃ p ∈ set (Forall A # G). params p). x ≠ c ⟹ ?g x ≠ ?g c⟩
    using fresh by auto
  moreover have ⟨map (psubst ?g) (Forall A # G) = Forall A # G⟩
    using fresh by (induct G) simp-all
  ultimately have G: ⟨map (psubst ?g) (subcs c ?s (Forall A # G)) = subcs c s
(Forall A # G)⟩
    using s A by (simp add: subcs-psubst)

```

```

have ⟨new-term c ?s⟩
  using fresh psubst-new-free' by fast
then have ⟨¬ subc c ?s (subst A (subc-term c ?s t) 0) # subcs c ?s G⟩
  using GammaForall by (metis new-subc-put subcs.simps(2))
moreover have ⟨new-term c (subc-term c ?s t)⟩
  using ⟨new-term c ?s⟩ new-subc-same' by simp
ultimately have ⟨¬ subst (subc c (liftt ?s) A) (subc-term c ?s t) 0 # subcs c ?s
G⟩
  by simp
moreover have ⟨Forall (subc c (liftt ?s) A) ∈ set (subcs c ?s (Forall A # G))⟩
  by simp
ultimately have ⟨¬ subcs c ?s (Forall A # G)⟩
  using TC.GammaForall by simp
then have ⟨¬ map (psubst ?g) (subcs c ?s (Forall A # G))⟩
  using TC-psubst inf-params by blast
then show ⟨¬ subcs c s (Forall A # G)⟩
  using G by simp
next
  case (GammaNegExists A t G)
  let ?params = ⟨params A ∪ (⋃ p ∈ set G. params p) ∪ paramst s ∪ paramst t ∪
{c}⟩

  have ⟨finite ?params⟩
    by simp
  then obtain fresh where fresh: ⟨fresh ∉ ?params⟩
    using ex-new-if-finite inf-params by metis

  let ?f = ⟨id(c := fresh)⟩
  let ?g = ⟨id(fresh := c)⟩
  let ?s = ⟨psubstt ?f s⟩

  have s: ⟨psubstt ?g ?s = s⟩
    using fresh psubst-new-away' by simp
  have ⟨subc (?g c) (psubstt ?g ?s) (psubst ?g (Neg (Exists A))) = subc c s (Neg
(Exists A))⟩
    using fresh by simp
  then have A: ⟨psubst ?g (subc c ?s (Neg (Exists A))) = subc c s (Neg (Exists
A))⟩
    using fun-upd-apply id-def subc-psubst UnCI fresh params.simps(7,9) by metis

  have ⟨∀ x ∈ (⋃ p ∈ set (Neg (Exists A) # G). params p). x ≠ c ⟶ ?g x ≠ ?g
c⟩
    using fresh by auto
  moreover have ⟨map (psubst ?g) (Neg (Exists A) # G) = Neg (Exists A) # G⟩
    using fresh by (induct G) simp-all
  ultimately have G: ⟨map (psubst ?g) (subcs c ?s (Neg (Exists A) # G)) =
subcs c s (Neg (Exists A) # G)⟩
    using s A by (simp add: subcs-psubst)

```

```

have ⟨new-term c ?s⟩
  using fresh psubst-new-free' by fast
then have ⟨¬ Neg (subc c ?s (subst A (subc-term c ?s t) 0)) # subcs c ?s G⟩
  using GammaNegExists by (metis new-subc-put subc.simps(4) subcs.simps(2))
moreover have ⟨new-term c (subc-term c ?s t)⟩
  using ⟨new-term c ?s⟩ new-subc-same' by simp
ultimately have ⟨¬ Neg (subst (subc c (liftt ?s) A) (subc-term c ?s t) 0) #
subcs c ?s G⟩
  by simp
moreover have ⟨Neg (Exists (subc c (liftt ?s) A)) ∈ set (subcs c ?s (Neg (Exists
A) # G))⟩
  by simp
ultimately have ⟨¬ subcs c ?s (Neg (Exists A) # G)⟩
  using TC.GammaNegExists by simp
then have ⟨¬ map (psubst ?g) (subcs c ?s (Neg (Exists A) # G))⟩
  using TC-psubst inf-params by blast
then show ⟨¬ subcs c s (Neg (Exists A) # G)⟩
  using G by simp
next
case (DeltaExists A n G)
then show ?case
proof (cases ⟨c = n⟩)
  case True
  then have ⟨¬ Exists A # G⟩
    using DeltaExists TC.DeltaExists by metis
  moreover have ⟨new c A⟩ and ⟨news c G⟩
    using DeltaExists True by simp-all
  ultimately show ?thesis
    by (simp add: subcs-news)
next
case False
let ?params = ⟨params A ∪ (⋃ p ∈ set G. params p) ∪ paramst s ∪ {c} ∪ {n}⟩

have ⟨finite ?params⟩
  by simp
then obtain fresh where fresh: ⟨fresh ∉ ?params⟩
  using ex-new-if-finite inf-params by metis

let ?s = ⟨psubstt (id(n := fresh)) s⟩
let ?f = ⟨id(n := fresh, fresh := n)⟩

have f: ⟨∀ x ∈ ?params. x ≠ c ⟶ ?f x ≠ ?f c⟩
  using fresh by simp

have ⟨new-term n ?s⟩
  using fresh psubst-new-free' by fast
then have ⟨psubstt ?f ?s = psubstt (id(fresh := n)) ?s⟩
  by (metis fun-upd-twist psubstt-upd(1))

```

```

then have psubst-s: ⟨psubstt ?f ?s = s⟩
  using fresh psubst-new-away' by simp

have ⟨?f c = c⟩ and ⟨new-term c (App fresh [])⟩
  using False fresh by auto

have ⟨psubst ?f (subc c ?s (subst A (App n []) 0)) =
  subc (?f c) (psubstt ?f ?s) (psubst ?f (subst A (App n []) 0))⟩
  by (simp add: subc-psubst)
also have ⟨... = subc c s (subst (psubst ?f A) (App fresh []) 0)⟩
  using ⟨?f c = c⟩ psubst-subst psubst-s by simp
also have ⟨... = subc c s (subst A (App fresh []) 0)⟩
  using DeltaExists fresh by simp
finally have psubst-A: ⟨psubst ?f (subc c ?s (subst A (App n []) 0)) =
  subst (subc c (liftt s) A) (App fresh []) 0⟩
  using ⟨new-term c (App fresh [])⟩ by simp

have ⟨news n G⟩
  using DeltaExists by simp
moreover have ⟨news fresh G⟩
  using fresh by (induct G) simp-all
ultimately have ⟨map (psubst ?f) G = G⟩
  by (induct G) simp-all
moreover have ⟨∀ x ∈ ⋃ p ∈ set G. params p. x ≠ c ⟶ ?f x ≠ ?f c⟩
  by auto
ultimately have psubst-G: ⟨map (psubst ?f) (subcs c ?s G) = subcs c s G⟩
  using ⟨?f c = c⟩ psubst-s by (simp add: subcs-psubst)

have ⟨¬ subc c ?s (subst A (App n []) 0) # subcs c ?s G⟩
  using DeltaExists by simp
then have ⟨¬ psubst ?f (subc c ?s (subst A (App n []) 0)) # map (psubst ?f)
(subcs c ?s G)⟩
  using TC-psubst inf-params DeltaExists.hyps(3) by fastforce
then have ⟨¬ psubst ?f (subc c ?s (subst A (App n []) 0)) # subcs c s G⟩
  using psubst-G by simp
then have sub-A: ⟨¬ subst (subc c (liftt s) A) (App fresh []) 0 # subcs c s G⟩
  using psubst-A by simp

have ⟨new-term fresh s⟩
  using fresh by simp
then have ⟨new-term fresh (liftt s)⟩
  by simp
then have ⟨new fresh (subc c (liftt s) A)⟩
  using fresh new-subc by simp
moreover have ⟨news fresh (subcs c s G)⟩
  using ⟨news fresh G⟩ ⟨new-term fresh s⟩ news-subcs by fast
ultimately show ⟨¬ subcs c s (Exists A # G)⟩
  using TC.DeltaExists sub-A by fastforce
qed

```

```

next
case (DeltaNegForall A n G)
then show ?case
proof (cases ⟨c = n⟩)
  case True
  then have ⟨¬ Neg (Forall A) ≠ G⟩
    using DeltaNegForall TC.DeltaNegForall by metis
  moreover have ⟨new c A⟩ and ⟨news c G⟩
    using DeltaNegForall True by simp-all
  ultimately show ?thesis
    by (simp add: subcs-news)
next
case False
let ?params = ⟨params A ∪ (⋃ p ∈ set G. params p) ∪ paramst s ∪ {c} ∪ {n}⟩

have ⟨finite ?params⟩
  by simp
then obtain fresh where fresh: ⟨fresh ∉ ?params⟩
  using ex-new-if-finite inf-params by metis

let ?s = ⟨psubst (id(n := fresh)) s⟩
let ?f = ⟨id(n := fresh, fresh := n)⟩

have f: ⟨∀ x ∈ ?params. x ≠ c ⟶ ?f x ≠ ?f c⟩
  using fresh by simp

have ⟨new-term n ?s⟩
  using fresh psubst-new-free' by fast
then have ⟨psubst ?f ?s = psubst (id(fresh := n)) ?s⟩
  using fun-upd-twist psubst-upd(1) by metis
then have psubst-s: ⟨psubst ?f ?s = s⟩
  using fresh psubst-new-away' by simp

have ⟨?f c = c⟩ and ⟨new-term c (App fresh [])⟩
  using False fresh by auto

have ⟨psubst ?f (subc c ?s (Neg (subst A (App n []) 0))) =
  subc (?f c) (psubst ?f ?s) (psubst ?f (Neg (subst A (App n []) 0)))⟩
  by (simp add: subc-psubst)
also have ⟨... = subc c s (Neg (subst (psubst ?f A)(App fresh []) 0))⟩
  using ⟨?f c = c⟩ psubst-subst psubst-s by simp
also have ⟨... = subc c s (Neg (subst A (App fresh []) 0))⟩
  using DeltaNegForall fresh by simp
finally have psubst-A: ⟨psubst ?f (subc c ?s (Neg (subst A (App n []) 0))) =
  Neg (subst (subc c (lift s) A) (App fresh []) 0)⟩
  using ⟨new-term c (App fresh [])⟩ by simp

have ⟨news n G⟩
  using DeltaNegForall by simp

```

```

moreover have ⟨news fresh G⟩
  using fresh by (induct G) simp-all
ultimately have ⟨map (psubst ?f) G = G⟩
  by (induct G) simp-all
moreover have ⟨∀ x ∈ ⋃ p ∈ set G. params p. x ≠ c → ?f x ≠ ?f c⟩
  by auto
ultimately have psubst-G: ⟨map (psubst ?f) (subcs c ?s G) = subcs c s G⟩
  using ⟨?f c = c⟩ psubst-s by (simp add: subcs-psubst)

have ⟨¬ subc c ?s (Neg (subst A (App n []) 0)) # subcs c ?s G⟩
  using DeltaNegForall by simp
then have ⟨¬ psubst ?f (subc c ?s (Neg (subst A (App n []) 0)))
  # map (psubst ?f) (subcs c ?s G)⟩
  using TC-psubst inf-params DeltaNegForall.hyps(3) by fastforce
then have ⟨¬ psubst ?f (subc c ?s (Neg (subst A (App n []) 0))) # subcs c s
G⟩
  using psubst-G by simp
then have sub-A: ⟨¬ Neg (subst (subc c (liftt s) A) (App fresh []) 0) # subcs
c s G⟩
  using psubst-A by simp

have ⟨new-term fresh s⟩
  using fresh by simp
then have ⟨new-term fresh (liftt s)⟩
  by simp
then have ⟨new fresh (subc c (liftt s) A)⟩
  using fresh new-subc by simp
moreover have ⟨news fresh (subcs c s G)⟩
  using ⟨news fresh G⟩ ⟨new-term fresh s⟩ news-subcs by fast
ultimately show ⟨¬ subcs c s (Neg (Forall A) # G)⟩
  using TC.DeltaNegForall sub-A by fastforce
qed
next
case (Order G G')
then show ?case
  using TC.Order set-map subcs-map by metis
qed (auto intro: TC.intros)

lemma TC-map-subc:
  fixes G :: ⟨('a, 'b) form list⟩
  assumes inf-params: ⟨infinite (UNIV :: 'a set)⟩
  shows ⟨¬ G ⇒ ¬ map (subc c s) G⟩
  using assms TC-subcs subcs-map by metis

lemma ex-all-closed: ⟨∃ m. list-all (closed m) G⟩
proof (induct G)
  case Nil
  then show ?case
    by simp

```

```

next
  case (Cons a G)
  then show ?case
    unfolding list-all-def
    using ex-closed closed-mono
    by (metis Ball-set list-all-simps(1) nat-le-linear)
qed

primrec sub-consts :: ⟨'a list ⇒ ('a, 'b) form ⇒ ('a, 'b) form⟩ where
  ⟨sub-consts [] p = p⟩
| ⟨sub-consts (c # cs) p = sub-consts cs (subst p (App c [])) (length cs)⟩

lemma valid-sub-consts:
  assumes ⟨∀ (e :: nat ⇒ 'a) f g. eval e f g p⟩
  shows ⟨eval (e :: nat ⇒ 'a) f g (sub-consts cs p)⟩
  using assms by (induct cs arbitrary: p) simp-all

lemma closed-sub' [simp]:
  assumes ⟨k ≤ m⟩ shows
    ⟨closedt (Suc m) t = closedt m (substt t (App c [])) k⟩
    ⟨closedts (Suc m) l = closedts m (substts l (App c [])) k⟩
  using assms by (induct t and l rule: closedt.induct closedts.induct) auto

lemma closed-sub: ⟨k ≤ m ⇒ closed (Suc m) p = closed m (subst p (App c [])) k⟩
  by (induct p arbitrary: m k) simp-all

lemma closed-sub-consts: ⟨length cs = k ⇒ closed m (sub-consts cs p) = closed (m + k) p⟩
proof (induct cs arbitrary: k p)
  case Nil
  then show ?case
    by simp
next
  case (Cons c cs)
  then show ?case
    using closed-sub by fastforce
qed

lemma map-sub-consts-Nil: ⟨map (sub-consts []) G = G⟩
  by (induct G) simp-all

primrec conjoin :: ⟨('a, 'b) form list ⇒ ('a, 'b) form⟩ where
  ⟨conjoin [] = Neg ⊥⟩
| ⟨conjoin (p # ps) = And p (conjoin ps)⟩

lemma eval-conjoin: ⟨list-all (eval e f g) G = eval e f g (conjoin G)⟩
  by (induct G) simp-all

```



**lemma** *valid-sub*:

**fixes**  $e :: \langle \text{nat} \Rightarrow 'a \rangle$

**assumes**  $\langle \forall (e :: \text{nat} \Rightarrow 'a) f g. \text{eval } e f g p \longrightarrow \text{eval } e f g q \rangle$

**shows**  $\langle \text{eval } e f g (\text{subst } p t m) \longrightarrow \text{eval } e f g (\text{subst } q t m) \rangle$

**using** *assms* **by** *simp*

**lemma** *eval-sub-consts*:

**fixes**  $e :: \langle \text{nat} \Rightarrow 'a \rangle$

**assumes**  $\langle \forall (e :: \text{nat} \Rightarrow 'a) f g. \text{eval } e f g p \longrightarrow \text{eval } e f g q \rangle$

**and**  $\langle \text{eval } e f g (\text{sub-consts } cs p) \rangle$

**shows**  $\langle \text{eval } e f g (\text{sub-consts } cs q) \rangle$

**using** *assms*

**proof** (*induct cs arbitrary: p q*)

**case** *Nil*

**then show** *?case*

**by** *simp*

**next**

**case** (*Cons c cs*)

**then show** *?case*

**by** (*metis sub-consts.simps(2) subst-lemma*)

**qed**

**lemma** *sub-consts-And* [*simp*]:  $\langle \text{sub-consts } cs (\text{And } p q) = \text{And } (\text{sub-consts } cs p) (\text{sub-consts } cs q) \rangle$

**by** (*induct cs arbitrary: p q simp-all*)

**lemma** *sub-consts-conjoin*:

$\langle \text{eval } e f g (\text{sub-consts } cs (\text{conjoin } G)) = \text{eval } e f g (\text{conjoin } (\text{map } (\text{sub-consts } cs) G)) \rangle$

**proof** (*induct G*)

**case** *Nil*

**then show** *?case*

**by** (*induct cs simp-all*)

**next**

**case** (*Cons p G*)

**then show** *?case*

**using** *sub-consts-And* **by** *simp*

**qed**

**lemma** *all-sub-consts-conjoin*:

$\langle \text{list-all } (\text{eval } e f g) (\text{map } (\text{sub-consts } cs) G) = \text{eval } e f g (\text{sub-consts } cs (\text{conjoin } G)) \rangle$

**by** (*induct G (simp-all add: valid-sub-consts)*)

**lemma** *valid-all-sub-consts*:

**fixes**  $e :: \langle \text{nat} \Rightarrow 'a \rangle$

**assumes**  $\langle \forall (e :: \text{nat} \Rightarrow 'a) f g. \text{list-all } (\text{eval } e f g) G \longrightarrow \text{eval } e f g p \rangle$

**shows**  $\langle \text{list-all } (\text{eval } e f g) (\text{map } (\text{sub-consts } cs) G) \longrightarrow \text{eval } e f g (\text{sub-consts } cs p) \rangle$

using *assms eval-conjoin eval-sub-consts all-sub-consts-conjoin by metis*

**lemma** *TC-vars-for-consts:*

fixes  $G :: \langle 'a, 'b \rangle$  form list

assumes  $\langle infinite (UNIV :: 'a \text{ set}) \rangle$

shows  $\langle \vdash G \implies \vdash map (\lambda p. vars\text{-for}\text{-consts } p \text{ } cs) G \rangle$

**proof** (induct cs)

case Nil

then show ?case

by simp

next

case (Cons c cs)

have  $\langle \vdash map (\lambda p. vars\text{-for}\text{-consts } p (c \# cs)) G =$

$\vdash map (\lambda p. subc c (Var (length cs)) (vars\text{-for}\text{-consts } p \text{ } cs)) G \rangle$

by simp

also have  $\langle \dots = \vdash map (subc c (Var (length cs)) o (\lambda p. vars\text{-for}\text{-consts } p \text{ } cs)) G \rangle$

unfolding comp-def by simp

also have  $\langle \dots = \vdash map (subc c (Var (length cs))) (map (\lambda p. vars\text{-for}\text{-consts } p \text{ } cs) G) \rangle$

by simp

finally show ?case

using Cons TC-map-subc assms by metis

qed

**lemma** *vars-for-consts-sub-consts:*

$\langle closed (length cs) p \implies list\text{-all } (\lambda c. new c p) cs \implies distinct cs \implies$

$vars\text{-for}\text{-consts (sub-consts cs } p) cs = p \rangle$

**proof** (induct cs arbitrary: p)

case (Cons c cs)

then show ?case

using subst-new-all closed-sub by force

qed simp

**lemma** *all-vars-for-consts-sub-consts:*

$\langle list\text{-all } (closed (length cs)) G \implies list\text{-all } (\lambda c. list\text{-all } (new c) G) cs \implies distinct cs \implies$

$map (\lambda p. vars\text{-for}\text{-consts } p \text{ } cs) (map (sub-consts cs) G) = G \rangle$

using vars-for-consts-sub-consts unfolding list-all-def

by (induct G) fastforce+

**lemma** *new-conjoin:*  $\langle new c (conjoin G) \implies list\text{-all } (new c) G \rangle$

by (induct G) simp-all

**lemma** *all-fresh-constants:*

fixes  $G :: \langle 'a, 'b \rangle$  form list

assumes  $\langle infinite (UNIV :: 'a \text{ set}) \rangle$

shows  $\langle \exists cs. length cs = m \wedge list\text{-all } (\lambda c. list\text{-all } (new c) G) cs \wedge distinct cs \rangle$

**proof** –

**obtain**  $cs$  **where**  $\langle \text{length } cs = m \rangle \langle \text{list-all } (\lambda c. \text{new } c \text{ (conjoin } G)) \text{ } cs \rangle \langle \text{distinct } cs \rangle$   
**using**  $assms$   $\text{fresh-constants}$  **by**  $blast$   
**then show**  $?thesis$   
**using**  $\text{new-conjoin}$  **unfolding**  $\text{list-all-def}$  **by**  $metis$   
**qed**

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

## 2.4 Completeness

**theorem**  $\text{tableau-completeness}$ :

**fixes**  $G :: \langle (\text{nat}, \text{nat}) \text{ form list} \rangle$

**assumes**  $\langle \forall (e :: \text{nat} \Rightarrow \text{nat hterm}) f g. \text{list-all (eval } e \text{ } f \text{ } g) \text{ } G \longrightarrow \text{eval } e \text{ } f \text{ } g \text{ } p \rangle$

**shows**  $\langle \text{tableauproof } G \text{ } p \rangle$

**proof** –

**obtain**  $m$  **where**  $*$ :  $\langle \text{list-all (closed } m) (p \# G) \rangle$

**using**  $\text{ex-all-closed}$  **by**  $blast$

**moreover obtain**  $cs$  **where**  $**$ :

$\langle \text{length } cs = m \rangle$

$\langle \text{distinct } cs \rangle$

$\langle \text{list-all } (\lambda c. \text{list-all (new } c) (p \# G)) \text{ } cs \rangle$

**using**  $\text{all-fresh-constants}$  **by**  $blast$

**ultimately have**  $\langle \text{closed } 0 \text{ (sub-consts } cs \text{ } p) \rangle$

**using**  $\text{closed-sub-consts}$  **by**  $\text{fastforce}$

**moreover have**  $\langle \text{list-all (closed } 0) (\text{map (sub-consts } cs) \text{ } G) \rangle$

**using**  $\text{closed-sub-consts} *$   $\langle \text{length } cs = m \rangle$  **by**  $(\text{induct } G) \text{ fastforce+}$

**moreover have**  $\langle \forall (e :: \text{nat} \Rightarrow \text{nat hterm}) f g. \text{list-all (eval } e \text{ } f \text{ } g) (\text{map (sub-consts } cs) \text{ } G) \longrightarrow$

$\text{eval } e \text{ } f \text{ } g \text{ (sub-consts } cs \text{ } p) \rangle$

**using**  $assms \text{ valid-all-sub-consts}$  **by**  $blast$

**ultimately have**  $\langle \neg \text{Neg (sub-consts } cs \text{ } p) \# \text{map (sub-consts } cs) \text{ } G \rangle$

**using**  $\text{tableau-completeness'}$  **unfolding**  $\text{tableauproof-def}$  **by**  $\text{simp}$

**then have**  $\langle \neg \text{map (sub-consts } cs) (\text{Neg } p \# G) \rangle$

**by**  $(\text{simp add: sub-consts-Neg})$

**then have**  $\langle \neg \text{map } (\lambda p. \text{vars-for-consts } p \text{ } cs) (\text{map (sub-consts } cs) (\text{Neg } p \# G)) \rangle$

**using**  $\text{TC-vars-for-consts}$  **by**  $blast$

**then show**  $?thesis$

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

**using**  $\text{all-vars-for-consts-sub-consts}$   $[\text{where } G = \langle \text{Neg } p \# G \rangle] *$   $**$  **by**  $\text{simp}$

**qed**

**corollary**

**fixes**  $p :: \langle (\text{nat}, \text{nat}) \text{ form} \rangle$

**assumes**  $\langle \forall (e :: \text{nat} \Rightarrow \text{nat hterm}) f g. \text{eval } e \text{ } f \text{ } g \text{ } p \rangle$

**shows**  $\langle \neg [\text{Neg } p] \rangle$

**using**  $assms \text{ tableau-completeness}$  **unfolding**  $\text{tableauproof-def}$  **by**  $\text{simp}$

end

### 3 Sequent Calculus

**theory** *Sequent* **imports** *Tableau* **begin**

**inductive** *SC* ::  $\langle ('a, 'b) \text{ form list} \Rightarrow \text{bool} \rangle (\langle \vdash \rightarrow 0 \rangle)$  **where**

*Basic*:  $\langle \vdash \text{Pred } i \ l \ \# \ \text{Neg } (\text{Pred } i \ l) \ \# \ G \rangle$   
| *BasicNegFF*:  $\langle \vdash \text{Neg } \perp \ \# \ G \rangle$   
| *BasicTT*:  $\langle \vdash \top \ \# \ G \rangle$   
| *AlphaNegNeg*:  $\langle \vdash A \ \# \ G \Longrightarrow \vdash \text{Neg } (\text{Neg } A) \ \# \ G \rangle$   
| *AlphaNegAnd*:  $\langle \vdash \text{Neg } A \ \# \ \text{Neg } B \ \# \ G \Longrightarrow \vdash \text{Neg } (\text{And } A \ B) \ \# \ G \rangle$   
| *AlphaOr*:  $\langle \vdash A \ \# \ B \ \# \ G \Longrightarrow \vdash \text{Or } A \ B \ \# \ G \rangle$   
| *AlphaImpl*:  $\langle \vdash \text{Neg } A \ \# \ B \ \# \ G \Longrightarrow \vdash \text{Impl } A \ B \ \# \ G \rangle$   
| *BetaAnd*:  $\langle \vdash A \ \# \ G \Longrightarrow \vdash B \ \# \ G \Longrightarrow \vdash \text{And } A \ B \ \# \ G \rangle$   
| *BetaNegOr*:  $\langle \vdash \text{Neg } A \ \# \ G \Longrightarrow \vdash \text{Neg } B \ \# \ G \Longrightarrow \vdash \text{Neg } (\text{Or } A \ B) \ \# \ G \rangle$   
| *BetaNegImpl*:  $\langle \vdash A \ \# \ G \Longrightarrow \vdash \text{Neg } B \ \# \ G \Longrightarrow \vdash \text{Neg } (\text{Impl } A \ B) \ \# \ G \rangle$   
| *GammaExists*:  $\langle \vdash \text{subst } A \ t \ 0 \ \# \ G \Longrightarrow \vdash \text{Exists } A \ \# \ G \rangle$   
| *GammaNegForall*:  $\langle \vdash \text{Neg } (\text{subst } A \ t \ 0) \ \# \ G \Longrightarrow \vdash \text{Neg } (\text{Forall } A) \ \# \ G \rangle$   
| *DeltaForall*:  $\langle \vdash \text{subst } A \ (\text{App } n \ []) \ 0 \ \# \ G \Longrightarrow \text{news } n \ (A \ \# \ G) \Longrightarrow \vdash \text{Forall } A \ \# \ G \rangle$   
| *DeltaNegExists*:  $\langle \vdash \text{Neg } (\text{subst } A \ (\text{App } n \ []) \ 0) \ \# \ G \Longrightarrow \text{news } n \ (A \ \# \ G) \Longrightarrow \vdash \text{Neg } (\text{Exists } A) \ \# \ G \rangle$   
| *Order*:  $\langle \vdash G \Longrightarrow \text{set } G = \text{set } G' \Longrightarrow \vdash G' \rangle$

**lemma** *Shift*:  $\langle \vdash \text{rotate1 } G \Longrightarrow \vdash G \rangle$

**by** (*simp add: Order*)

**lemma** *Swap*:  $\langle \vdash B \ \# \ A \ \# \ G \Longrightarrow \vdash A \ \# \ B \ \# \ G \rangle$

**by** (*simp add: Order insert-commute*)

**lemma**  $\langle \vdash [\text{Neg } (\text{Pred } "A" \ []), \text{Pred } "A" \ []] \rangle$

**by** (*rule Shift, simp*) (*rule Basic*)

**lemma**  $\langle \vdash [\text{And } (\text{Pred } "A" \ []) (\text{Pred } "B" \ []), \text{Neg } (\text{And } (\text{Pred } "B" \ []) (\text{Pred } "A" \ []))] \rangle$

**apply** (*rule BetaAnd*)

**apply** (*rule Swap*)

**apply** (*rule AlphaNegAnd*)

**apply** (*rule Shift, simp, rule Swap*)

**apply** (*rule Basic*)

**apply** (*rule Swap*)

**apply** (*rule AlphaNegAnd*)

**apply** (*rule Shift, rule Shift, simp*)

**apply** (*rule Basic*)

**done**

### 3.1 Soundness

**lemma** *SC-soundness*:  $\langle \vdash G \implies \exists p \in \text{set } G. \text{eval } e \text{ f } g \text{ p} \rangle$

**proof** (*induct*  $G$  *arbitrary*:  $f$  *rule*: *SC.induct*)

**case** (*DeltaForall*  $A$   $n$   $G$ )

**then consider**

$\langle \forall x. \text{eval } e \text{ (f(n := } \lambda w. x)) \text{ g (subst A (App n [])) 0)} \rangle |$

$\langle \exists x. \exists p \in \text{set } G. \text{eval } e \text{ (f(n := } \lambda w. x)) \text{ g p} \rangle$

**by** *fastforce*

**then show** *?case*

**proof** *cases*

**case** 1

**then have**  $\langle \forall x. \text{eval } (\text{shift } e \text{ 0 } x) \text{ (f(n := } \lambda w. x)) \text{ g } A \rangle$

**by** *simp*

**then have**  $\langle \forall x. \text{eval } (\text{shift } e \text{ 0 } x) \text{ f } g \text{ A} \rangle$

**using**  $\langle \text{news } n \text{ (A \# G)} \rangle$  **by** *simp*

**then show** *?thesis*

**by** *simp*

**next**

**case** 2

**then have**  $\langle \exists p \in \text{set } G. \text{eval } e \text{ f } g \text{ p} \rangle$

**using**  $\langle \text{news } n \text{ (A \# G)} \rangle$  **using** *Ball-set insert-iff list.set(2) upd-lemma* **by**

*metis*

**then show** *?thesis*

**by** *simp*

**qed**

**next**

**case** (*DeltaNegExists*  $A$   $n$   $G$ )

**then consider**

$\langle \forall x. \text{eval } e \text{ (f(n := } \lambda w. x)) \text{ g (Neg (subst A (App n [])) 0)} \rangle |$

$\langle \exists x. \exists p \in \text{set } G. \text{eval } e \text{ (f(n := } \lambda w. x)) \text{ g p} \rangle$

**by** *fastforce*

**then show** *?case*

**proof** *cases*

**case** 1

**then have**  $\langle \forall x. \text{eval } (\text{shift } e \text{ 0 } x) \text{ (f(n := } \lambda w. x)) \text{ g (Neg A)} \rangle$

**by** *simp*

**then have**  $\langle \forall x. \text{eval } (\text{shift } e \text{ 0 } x) \text{ f } g \text{ (Neg A)} \rangle$

**using**  $\langle \text{news } n \text{ (A \# G)} \rangle$  **by** *simp*

**then show** *?thesis*

**by** *simp*

**next**

**case** 2

**then have**  $\langle \exists p \in \text{set } G. \text{eval } e \text{ f } g \text{ p} \rangle$

**using**  $\langle \text{news } n \text{ (A \# G)} \rangle$  **using** *Ball-set insert-iff list.set(2) upd-lemma* **by**

*metis*

**then show** *?thesis*

**by** *simp*

**qed**

**qed** *auto*

### 3.2 Tableau Calculus Equivalence

**fun** *compl* ::  $\langle ('a, 'b) \text{ form} \Rightarrow ('a, 'b) \text{ form} \rangle$  **where**  
 $\langle \text{compl } (\text{Neg } p) = p \rangle$   
 $| \langle \text{compl } p = \text{Neg } p \rangle$

**lemma** *compl*:  $\langle \text{compl } p = \text{Neg } p \vee (\exists q. \text{compl } p = q \wedge p = \text{Neg } q) \rangle$   
**by** (*cases p rule: compl.cases*) *simp-all*

**lemma** *new-compl*:  $\langle \text{new } n \ p \Longrightarrow \text{new } n \ (\text{compl } p) \rangle$   
**by** (*cases p rule: compl.cases*) *simp-all*

**lemma** *news-compl*:  $\langle \text{news } n \ G \Longrightarrow \text{news } n \ (\text{map } \text{compl } G) \rangle$   
**using** *new-compl* **by** (*induct G*) *fastforce+*

**theorem** *TC-SC*:  $\langle \vdash G \Longrightarrow \vdash \text{map } \text{compl } G \rangle$

**proof** (*induct G rule: TC.induct*)

**case** (*Basic i l G*)

**then show** *?case*

**using** *SC.Basic Swap* **by** *fastforce*

**next**

**case** (*AlphaNegNeg A G*)

**then show** *?case*

**using** *SC.AlphaNegNeg compl* **by** (*metis compl.simps(1) list.simps(9)*)

**next**

**case** (*AlphaAnd A B G*)

**then have**  $\vdash \text{compl } A \# \text{compl } B \# \text{map } \text{compl } G$

**by** *simp*

**then have**  $\vdash \text{Neg } A \# \text{Neg } B \# \text{map } \text{compl } G$

**using** *compl AlphaNegNeg Swap* **by** *metis*

**then show** *?case*

**using** *AlphaNegAnd* **by** *simp*

**next**

**case** (*AlphaNegImpl A B G*)

**then have**  $\vdash \text{compl } A \# B \# \text{map } \text{compl } G$

**by** *simp*

**then have**  $\vdash \text{Neg } A \# B \# \text{map } \text{compl } G$

**using** *compl AlphaNegNeg* **by** *metis*

**then show** *?case*

**using** *AlphaImpl* **by** *simp*

**next**

**case** (*BetaOr A G B*)

**then have**  $\langle \vdash \text{compl } A \# \text{map } \text{compl } G \rangle \langle \vdash \text{compl } B \# \text{map } \text{compl } G \rangle$

**by** *simp-all*

**then have**  $\langle \vdash \text{Neg } A \# \text{map } \text{compl } G \rangle \langle \vdash \text{Neg } B \# \text{map } \text{compl } G \rangle$

**using** *compl AlphaNegNeg* **by** *metis+*

**then show** *?case*

**using** *BetaNegOr* **by** *simp*

**next**

**case** (*BetaImpl A G B*)

```

then have  $\langle \vdash A \# \text{map compl } G \rangle \langle \vdash \text{compl } B \# \text{map compl } G \rangle$ 
  by simp-all
then have  $\langle \vdash A \# \text{map compl } G \rangle \langle \vdash \text{Neg } B \# \text{map compl } G \rangle$ 
  by  $-$  (assumption, metis compl AlphaNegNeg)
then have  $\langle \vdash \text{Neg } (\text{Impl } A \ B) \# \text{map compl } G \rangle$ 
  using BetaNegImpl by blast
then have  $\langle \vdash \text{compl } (\text{Impl } A \ B) \# \text{map compl } G \rangle$ 
  using  $\langle \vdash A \# \text{map compl } G \rangle$  compl by simp
then show ?case
  by simp
next
case (GammaForall A t G)
then have  $\langle \vdash \text{compl } (\text{subst } A \ t \ 0) \# \text{map compl } G \rangle$ 
  by simp
then have  $\langle \vdash \text{Neg } (\text{subst } A \ t \ 0) \# \text{map compl } G \rangle$ 
  using compl AlphaNegNeg by metis
then show ?case
  using GammaNegForall by simp
next
case (DeltaExists A n G)
then have  $\langle \vdash \text{compl } (\text{subst } A \ (\text{App } n \ []) \ 0) \# \text{map compl } G \rangle$ 
  by simp
then have  $\langle \vdash \text{Neg } (\text{subst } A \ (\text{App } n \ []) \ 0) \# \text{map compl } G \rangle$ 
  using compl AlphaNegNeg by metis
moreover have  $\langle \text{news } n \ (A \# \text{map compl } G) \rangle$ 
  using DeltaExists news-compl by fastforce
ultimately show ?case
  using DeltaNegExists by simp
next
case (DeltaNegForall A n G)
then have  $\langle \vdash \text{subst } A \ (\text{App } n \ []) \ 0 \# \text{map compl } G \rangle$ 
  by simp
moreover have  $\langle \text{news } n \ (A \# \text{map compl } G) \rangle$ 
  using DeltaNegForall news-compl by fastforce
ultimately show ?case
  using DeltaForall by simp
qed (simp-all add: SC.intros)

```

### 3.3 Completeness

**theorem** *SC-completeness*:

**fixes**  $p :: \langle (\text{nat}, \text{nat}) \text{ form} \rangle$

**assumes**  $\langle \forall (e :: \text{nat} \Rightarrow \text{nat hterm}) f g. \text{list-all } (\text{eval } e \ f \ g) \ ps \longrightarrow \text{eval } e \ f \ g \rangle$

**shows**  $\langle \vdash p \# \text{map compl } ps \rangle$

**proof**  $-$

**have**  $\langle \vdash \text{Neg } p \# ps \rangle$

**using** *assms tableau-completeness unfolding tableauproof-def* **by** *simp*

**then show** *?thesis*

**using** *TC-SC* **by** *fastforce*

qed

corollary

fixes  $p :: \langle (nat, nat) \text{ form} \rangle$   
assumes  $\langle \forall (e :: nat \Rightarrow nat \text{ hterm}) f g. \text{eval } e f g p \rangle$   
shows  $\langle \vdash [p] \rangle$   
using *assms SC-completeness list.map(1)* by *metis*

end

theory *Sequent2* imports *Sequent* begin

## 4 Completeness Revisited

lemma  $\langle \exists p. q = \text{compl } p \rangle$   
by (*metis compl.simps(1)*)

definition *compl'* where

$\langle \text{compl}' = (\lambda q. (\text{SOME } p. q = \text{compl } p)) \rangle$

lemma *comp'-sem*:

$\langle \text{eval } e f g (\text{compl}' p) \longleftrightarrow \neg \text{eval } e f g p \rangle$   
by (*smt compl'-def compl.simps(1) compl eval.simps(7) someI-ex*)

lemma *comp'-sem-list*:  $\langle \text{list-ex } (\lambda p. \neg \text{eval } e f g p) (\text{map } \text{compl}' ps) \longleftrightarrow \text{list-ex } (\text{eval } e f g) ps \rangle$

by (*induct ps*) (*use comp'-sem in auto*)

theorem *SC-completeness'*:

fixes  $ps :: \langle (nat, nat) \text{ form list} \rangle$   
assumes  $\langle \forall (e :: nat \Rightarrow nat \text{ hterm}) f g. \text{list-ex } (\text{eval } e f g) (p \# ps) \rangle$   
shows  $\langle \vdash p \# ps \rangle$

proof –

define *ps'* where  $\langle ps' = \text{map } \text{compl}' ps \rangle$

then have  $\langle ps = \text{map } \text{compl } ps' \rangle$

by (*induct ps arbitrary: ps'*) (*simp, smt compl'-def compl.simps(1) list.simps(9) someI-ex*)

from *assms* have  $\langle \forall (e :: nat \Rightarrow nat \text{ hterm}) f g. (\text{list-ex } (\text{eval } e f g) ps) \vee \text{eval } e f g p \rangle$

by *auto*

then have  $\langle \forall (e :: nat \Rightarrow nat \text{ hterm}) f g. (\text{list-ex } (\lambda p. \neg \text{eval } e f g p) ps') \vee \text{eval } e f g p \rangle$

unfolding *ps'-def* using *comp'-sem-list* by *blast*

then have  $\langle \forall (e :: nat \Rightarrow nat \text{ hterm}) f g. \text{list-all } (\text{eval } e f g) ps' \longrightarrow \text{eval } e f g p \rangle$

by (*metis Ball-set Bex-set*)

then have  $\langle \vdash p \# \text{map } \text{compl } ps' \rangle$

using *SC-completeness* by *blast*

then show *?thesis*

using  $\langle ps = \text{map } \text{compl } ps' \rangle$  by *auto*

qed



```
corollary  
  fixes ps :: ⟨(nat, nat) form list⟩  
  assumes ⟨ $\forall (e :: nat \Rightarrow nat \text{ hterm}) f g. \text{list-ex (eval e f g) ps}$ ⟩  
  shows ⟨ $\vdash ps$ ⟩  
  using assms SC-completeness' by (cases ps) auto  
  
end
```

## References

- [1] M. Ben-Ari. *Mathematical Logic for Computer Science, 3rd Edition*. Springer, 2012.