

A Sequent Calculus for First-Order Logic

Asta Halkjær From

October 13, 2025

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

1	Common Notation	2
2	Tableau Calculus	2
2.1	Soundness	3
2.2	Completeness for Closed Formulas	3
2.3	Open Formulas	3
2.4	Completeness	6
3	Sequent Calculus	6
3.1	Soundness	7
3.2	Tableau Calculus Equivalence	7
3.3	Completeness	7
4	Completeness Revisited	7

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

lemma *Swap*: $\langle \neg B \ \# \ A \ \# \ G \Longrightarrow \neg A \ \# \ B \ \# \ G \rangle$
 $\langle \text{proof} \rangle$

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

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

2.1 Soundness

lemma *TC-soundness*:

⟨ $\vdash G \implies \exists p \in \text{set } G. \neg \text{eval } e \text{ f } g \text{ } p$ ⟩
 ⟨*proof*⟩

theorem *tableau-soundness*:

⟨*tableauproof* $ps \text{ } p \implies \text{list-all } (\text{eval } e \text{ f } g) \text{ } ps \implies \text{eval } e \text{ f } g \text{ } p$ ⟩
 ⟨*proof*⟩

2.2 Completeness for Closed Formulas

theorem *infinite-nonempty*: ⟨*infinite* $A \implies \exists x. x \in A$ ⟩
 ⟨*proof*⟩

theorem *TCd-consistency*:

assumes *inf-param*: ⟨*infinite* ($UNIV :: 'a \text{ set}$)⟩
shows ⟨*consistency* $\{S :: ('a, 'b) \text{ form set}. \exists G. S = \text{set } G \wedge \neg (\vdash G)\}$ ⟩
 ⟨*proof*⟩

theorem *tableau-completeness'*:

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

2.3 Open Formulas

lemma *TC-psubst*:

fixes $f :: \langle 'a \Rightarrow 'a \rangle$
assumes *inf-params*: ⟨*infinite* ($UNIV :: 'a \text{ set}$)⟩
shows ⟨ $\vdash G \implies \vdash \text{map } (psubst \text{ } f) \text{ } G$ ⟩
 ⟨*proof*⟩

lemma *subcs-map*: ⟨*subcs* $c \text{ } s \text{ } G = \text{map } (subc \text{ } c \text{ } s) \text{ } G$ ⟩
 ⟨*proof*⟩

lemma *TC-subcs*:

fixes $G :: \langle ('a, 'b) \text{ form list} \rangle$
assumes *inf-params*: ⟨*infinite* ($UNIV :: 'a \text{ set}$)⟩
shows ⟨ $\vdash G \implies \vdash \text{subcs } c \text{ } s \text{ } G$ ⟩
 ⟨*proof*⟩

lemma *TC-map-subc*:

fixes $G :: \langle ('a, 'b) \text{ form list} \rangle$
assumes *inf-params*: ⟨*infinite* ($UNIV :: 'a \text{ set}$)⟩
shows ⟨ $\vdash G \implies \vdash \text{map } (subc \text{ } c \text{ } s) \text{ } G$ ⟩
 ⟨*proof*⟩

lemma *ex-all-closed*: $\langle \exists m. \text{list-all } (\text{closed } m) \ G \rangle$
 $\langle \text{proof} \rangle$

primrec *sub-consts* :: $\langle 'a \text{ list} \Rightarrow ('a, 'b) \text{ form} \Rightarrow ('a, 'b) \text{ form} \rangle$ **where**
 $\langle \text{sub-consts } [] \ p = p \rangle$
 $| \langle \text{sub-consts } (c \# cs) \ p = \text{sub-consts } cs \ (\text{subst } p \ (\text{App } c \ [])) \ (\text{length } cs) \rangle$

lemma *valid-sub-consts*:
assumes $\langle \forall (e :: \text{nat} \Rightarrow 'a) \ f \ g. \text{eval } e \ f \ g \ p \rangle$
shows $\langle \text{eval } (e :: \text{nat} \Rightarrow 'a) \ f \ g \ (\text{sub-consts } cs \ p) \rangle$
 $\langle \text{proof} \rangle$

lemma *closed-sub' [simp]*:
assumes $\langle k \leq m \rangle$ **shows**
 $\langle \text{closedt } (\text{Suc } m) \ t = \text{closedt } m \ (\text{substt } t \ (\text{App } c \ [])) \ k \rangle$
 $\langle \text{closedts } (\text{Suc } m) \ l = \text{closedts } m \ (\text{substts } l \ (\text{App } c \ [])) \ k \rangle$
 $\langle \text{proof} \rangle$

lemma *closed-sub*: $\langle k \leq m \implies \text{closed } (\text{Suc } m) \ p = \text{closed } m \ (\text{subst } p \ (\text{App } c \ [])) \ k \rangle$
 $\langle \text{proof} \rangle$

lemma *closed-sub-consts*: $\langle \text{length } cs = k \implies \text{closed } m \ (\text{sub-consts } cs \ p) = \text{closed } (m + k) \ p \rangle$
 $\langle \text{proof} \rangle$

lemma *map-sub-consts-Nil*: $\langle \text{map } (\text{sub-consts } []) \ G = G \rangle$
 $\langle \text{proof} \rangle$

primrec *conjoin* :: $\langle ('a, 'b) \text{ form list} \Rightarrow ('a, 'b) \text{ form} \rangle$ **where**
 $\langle \text{conjoin } [] = \text{Neg } \perp \rangle$
 $| \langle \text{conjoin } (p \# ps) = \text{And } p \ (\text{conjoin } ps) \rangle$

lemma *eval-conjoin*: $\langle \text{list-all } (\text{eval } e \ f \ g) \ G = \text{eval } e \ f \ g \ (\text{conjoin } G) \rangle$
 $\langle \text{proof} \rangle$

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

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

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

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

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

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

lemma *TC-vars-for-consts*:
fixes $G :: \langle ('a, 'b) \text{ form list} \rangle$
assumes $\langle \text{infinite } (\text{UNIV} :: 'a \text{ set}) \rangle$
shows $\langle \neg G \Longrightarrow \neg \text{map } (\lambda p. \text{vars-for-consts } p \ cs) \ G \rangle$
 $\langle \text{proof} \rangle$

lemma *vars-for-consts-sub-consts*:
 $\langle \text{closed } (\text{length } cs) \ p \Longrightarrow \text{list-all } (\lambda c. \text{new } c \ p) \ cs \Longrightarrow \text{distinct } cs \Longrightarrow \text{vars-for-consts } (\text{sub-consts } cs \ p) \ cs = p \rangle$
 $\langle \text{proof} \rangle$

lemma *all-vars-for-consts-sub-consts*:
 $\langle \text{list-all } (\text{closed } (\text{length } cs)) \ G \Longrightarrow \text{list-all } (\lambda c. \text{list-all } (\text{new } c) \ G) \ cs \Longrightarrow \text{distinct } cs \Longrightarrow \text{map } (\lambda p. \text{vars-for-consts } p \ cs) \ (\text{map } (\text{sub-consts } cs) \ G) = G \rangle$
 $\langle \text{proof} \rangle$

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

lemma *all-fresh-constants*:
fixes $G :: \langle ('a, 'b) \text{ form list} \rangle$
assumes $\langle \text{infinite } (\text{UNIV} :: 'a \text{ set}) \rangle$
shows $\langle \exists cs. \text{length } cs = m \wedge \text{list-all } (\lambda c. \text{list-all } (\text{new } c) \ G) \ cs \wedge \text{distinct } cs \rangle$
 $\langle \text{proof} \rangle$

lemma *sub-consts-Neg*: $\langle \text{sub-consts } cs \ (\text{Neg } p) = \text{Neg } (\text{sub-consts } cs \ p) \rangle$

⟨proof⟩

2.4 Completeness

theorem *tableau-completeness*:

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

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

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

⟨proof⟩

corollary

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

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

shows $\langle \neg [Neg\ p] \rangle$

⟨proof⟩

end

3 Sequent Calculus

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

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

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

| *BasicNegFF*: $\langle \vdash Neg\ \perp \# G \rangle$

| *BasicTT*: $\langle \vdash \top \# G \rangle$

| *AlphaNegNeg*: $\langle \vdash A \# G \Longrightarrow \vdash Neg\ (Neg\ A) \# G \rangle$

| *AlphaNegAnd*: $\langle \vdash Neg\ A \# Neg\ B \# G \Longrightarrow \vdash Neg\ (And\ A\ B) \# G \rangle$

| *AlphaOr*: $\langle \vdash A \# B \# G \Longrightarrow \vdash Or\ A\ B \# G \rangle$

| *AlphaImpl*: $\langle \vdash Neg\ A \# B \# G \Longrightarrow \vdash Impl\ A\ B \# G \rangle$

| *BetaAnd*: $\langle \vdash A \# G \Longrightarrow \vdash B \# G \Longrightarrow \vdash And\ A\ B \# G \rangle$

| *BetaNegOr*: $\langle \vdash Neg\ A \# G \Longrightarrow \vdash Neg\ B \# G \Longrightarrow \vdash Neg\ (Or\ A\ B) \# G \rangle$

| *BetaNegImpl*: $\langle \vdash A \# G \Longrightarrow \vdash Neg\ B \# G \Longrightarrow \vdash Neg\ (Impl\ A\ B) \# G \rangle$

| *GammaExists*: $\langle \vdash \text{subst } A\ t\ 0 \# G \Longrightarrow \vdash \text{Exists } A \# G \rangle$

| *GammaNegForall*: $\langle \vdash Neg\ (\text{subst } A\ t\ 0) \# G \Longrightarrow \vdash Neg\ (\text{Forall } A) \# G \rangle$

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

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

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

⟨proof⟩

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

⟨proof⟩

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

⟨proof⟩

lemma $\vdash [And (Pred "A" []) (Pred "B" []), Neg (And (Pred "B" []) (Pred "A" []))]$
 $\langle proof \rangle$

3.1 Soundness

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

3.2 Tableau Calculus Equivalence

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

lemma *compl*: $\langle compl\ p = Neg\ p \vee (\exists q. compl\ p = q \wedge p = Neg\ q) \rangle$
 $\langle proof \rangle$

lemma *new-compl*: $\langle new\ n\ p \implies new\ n\ (compl\ p) \rangle$
 $\langle proof \rangle$

lemma *news-compl*: $\langle news\ n\ G \implies news\ n\ (map\ compl\ G) \rangle$
 $\langle proof \rangle$

theorem *TC-SC*: $\langle \vdash G \implies \vdash map\ compl\ G \rangle$
 $\langle proof \rangle$

3.3 Completeness

theorem *SC-completeness*:
fixes $p :: \langle (nat, nat)\ form \rangle$
assumes $\langle \forall (e :: nat \Rightarrow nat\ hterm)\ f\ g. list-all\ (eval\ e\ f\ g)\ ps \longrightarrow eval\ e\ f\ g\ p \rangle$
shows $\langle \vdash p \# map\ compl\ ps \rangle$
 $\langle proof \rangle$

corollary
fixes $p :: \langle (nat, nat)\ form \rangle$
assumes $\langle \forall (e :: nat \Rightarrow nat\ hterm)\ f\ g. eval\ e\ f\ g\ p \rangle$
shows $\langle \vdash [p] \rangle$
 $\langle proof \rangle$

end

theory *Sequent2* **imports** *Sequent* **begin**

4 Completeness Revisited

lemma $\langle \exists p. q = compl\ p \rangle$
 $\langle proof \rangle$

definition *compl'* **where**

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

lemma *comp'-sem*:

$\langle \text{eval } e \text{ f } g (\text{compl}' p) \longleftrightarrow \neg \text{eval } e \text{ f } g p \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

theorem *SC-completeness'*:

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

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

shows $\langle \vdash p \# ps \rangle$

$\langle \text{proof} \rangle$

corollary

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

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

shows $\langle \vdash ps \rangle$

$\langle \text{proof} \rangle$

end

References

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