# Verified Sequent-Solver

By Peter Lammich

June 9, 2016

## Contents

**1 Propositional Logic And Sequents** **1**
1.1 Syntax . . . . . 1
1.2 Semantics . . . . . 2

**2 Semantic Foundation of Sequent Calculus** **2**

**3 Syntactic Sequence Calculus Implementation** **3**

**4 Correctness and Completeness** **4**

**5 Test Cases** **4**

**theory** *Sequent-Solver*
**imports** *Main* ~~/*src*/*HOL*/*Library*/*Code-Char*
**begin**

# 1 Propositional Logic And Sequents

## 1.1 Syntax

**type-synonym** *name* = *String.literal*

**datatype** *frm* =
  *Atom name* ($ - [*181*] *180*)
| *Bot* (⊥)
| *Not frm* (*not* - [*171*] *170*)
| *And frm frm* (**infixl** *and* *160*)
| *Or frm frm* (**infixl** *or* *150*)
| *Impl frm frm* (**infixr** *imp* *140*)

**datatype** *sequent* = *SEQ frm set frm set* (**infix** ⇒ *30*)

### 1.2 Semantics

**type-synonym** *valuation* = *name* ⇒ *bool*

**context fixes** *A* :: *valuation*
**begin**
**fun** *eval* :: *frm* ⇒ *bool* **where**
*eval* ($*x*) ⟷ *A x*
| *eval* (⊥) ⟷ *False*
| *eval* (*not F*) ⟷ (¬*eval F*)
| *eval* (*F1 or F2*) ⟷ *eval F1* ∨ *eval F2*
| *eval* (*F1 and F2*) ⟷ *eval F1* ∧ *eval F2*
| *eval* (*F1 imp F2*) ⟷ (*eval F1* ⟶ *eval F2*)
**end**

Validity of a formula

**definition** *valid* (⊨ - [*100*] *100*) **where** *valid F* ≡ ∀ *A*. *eval A F*

Semantics of a sequent

**fun** *holds* **where**
*holds* (Γ⇒Δ) ⟷ (∀ *A*. (∀ *F*∈Γ. *eval A F*) ⟶ (∃ *F*∈Δ. *eval A F*))
**declare** *holds.simps*[*simp del*]

**lemma** *valid-by-sequent*: ⊨*F* ⟷ *holds* ({} ⇒ {*F*})
**by** (*auto simp*: *holds.simps valid-def*)

# 2 Semantic Foundation of Sequent Calculus

**context**
**notes** [*simp*] = *holds.simps*
**begin**
**notation** *insert* (**infixr** · *65*)

The standard axioms as semantic equivalences:

**lemma** *axiom1*: *holds* (*F*·Γ ⇒ *F*·Δ) **by** *simp*
**lemma** *axiom2*: *holds* (⊥·Γ ⇒ Δ) **by** *simp*
**lemma** *notL*: *holds* (*not F*·Γ ⇒ Δ) ⟷ *holds* (Γ ⇒ *F*·Δ) **by** *auto*
**lemma** *notR*: *holds* (Γ ⇒ *not F*·Δ) ⟷ *holds* (*F*·Γ ⇒ Δ) **by** *auto*
**lemma** *andL*: *holds* (*F and G*·Γ ⇒ Δ) ⟷ *holds* (*F*·*G*·Γ ⇒ Δ) **by** *auto*
**lemma** *andR*:
*holds* (Γ ⇒ *F and G*·Δ) ⟷ *holds* (Γ ⇒ *F*·Δ) ∧ *holds* (Γ ⇒ *G*·Δ) **by** *auto*
**lemma** *orL*:
*holds* (*F or G*·Γ ⇒ Δ) ⟷ *holds* (*F*·Γ ⇒ Δ) ∧ *holds* (*G*·Γ ⇒ Δ) **by** *auto*
**lemma** *orR*: *holds* (Γ ⇒ *F or G*·Δ) ⟷ *holds* (Γ ⇒ *F*·*G*·Δ) **by** *auto*
**lemma** *impL*:
*holds* (*F imp G*·Γ ⇒ Δ) ⟷ *holds* (Γ ⇒ *F*·Δ) ∧ *holds* (*G*·Γ ⇒ Δ) **by** *auto*
**lemma** *impR*: *holds* (Γ ⇒ *F imp G*·Δ) ⟷ *holds* (*F*·Γ ⇒ *G*·Δ) **by** *auto*

**lemmas** *standard-axioms* = *axiom2 notL notR andL andR orL orR impL impR*

Two auxiliary equivalences, required for our implementation

**lemma** *holds-atoms-only*:
*holds* (*Atom*‘*set P* ⇒ *Atom*‘*set Q*) ⟷ (*set P* ∩ *set Q* ≠ {})
**by** *clarsimp fast*
**lemma** *holds-bot-conl*: *holds* (Γ ⇒ ⊥·Δ) ⟷ *holds* (Γ ⇒ Δ) **by** *auto*

**lemmas** *auxiliary-axioms* = *holds-atoms-only holds-bot-conl*
**end**

# 3 Syntactic Sequence Calculus Implementation

In a sequent's implementation, we separate formulas and atoms

**datatype** *seqi* = *SEQ frm list name list frm list name list*

This function chooses the first formula of the premise/conclusion as principal formula, preferring non-branching rules.

**function** (*sequential*) *holdsi* :: *seqi* ⇒ *bool* **where**
*holdsi* (*SEQ* [] P [] Q) ⟷ *set P* ∩ *set Q*≠{}
| *holdsi* (*SEQ* (⊥#Γ) *P* Δ *Q*) ⟷ *True*
| *holdsi* (*SEQ* Γ *P* (⊥#Δ) *Q*) ⟷ *holdsi* (*SEQ* Γ *P* Δ *Q*)

| *holdsi* (*SEQ* (*not F*#Γ) *P* Δ *Q*) ⟷ *holdsi* (*SEQ* Γ *P* (*F*#Δ) *Q*)
| *holdsi* (*SEQ* (*F and G*#Γ) *P* Δ *Q*) ⟷ *holdsi* (*SEQ* (*F*#*G*#Γ) *P* Δ *Q*)

| *holdsi* (*SEQ* Γ *P* (*not F*#Δ) *Q*) ⟷ *holdsi* (*SEQ* (*F*#Γ) *P* Δ *Q*)
| *holdsi* (*SEQ* Γ *P* (*F or G*#Δ) *Q*) ⟷ *holdsi* (*SEQ* Γ *P* (*F*#*G*#Δ) *Q*)
| *holdsi* (*SEQ* Γ *P* (*F imp G*#Δ) *Q*) ⟷ *holdsi* (*SEQ* (*F*#Γ) *P* (*G*#Δ) *Q*)

| *holdsi* (*SEQ* Γ *P* (*F and G*#Δ) *Q*)
⟷ *holdsi* (*SEQ* Γ *P* (*F*#Δ) *Q*) ∧ *holdsi* (*SEQ* Γ *P* (*G*#Δ) *Q*)

| *holdsi* (*SEQ* (*F or G*#Γ) *P* Δ *Q*)
⟷ *holdsi* (*SEQ* (*F*#Γ) *P* Δ *Q*) ∧ *holdsi* (*SEQ* (*G*#Γ) *P* Δ *Q*)
| *holdsi* (*SEQ* (*F imp G*#Γ) *P* Δ *Q*)
⟷ *holdsi* (*SEQ* Γ *P* (*F*#Δ) *Q*) ∧ *holdsi* (*SEQ* (*G*#Γ) *P* Δ *Q*)

| *holdsi* (*SEQ* ($*v*#Γ) *P* Δ *Q*) ⟷ *holdsi* (*SEQ* Γ (*v*#*P*) Δ *Q*)
| *holdsi* (*SEQ* Γ *P* ($*v*#Δ) *Q*) ⟷ *holdsi* (*SEQ* Γ *P* Δ (*v*#*Q*))
**by** *pat-completeness auto*

Termination is shown by a simple measure function:

**primrec** *seqi-measure* **where**
*seqi-measure* (*SEQ* Γ - Δ -) = *2*∗*listsum* (*map size* (Γ@Δ)) + *length* (Γ@Δ)
**termination** *holdsi*
**by** (*relation measure seqi-measure*) *simp-all*

# 4 Correctness and Completeness

This function maps a sequent's implementation to a sequent

**primrec** *seqi-abs* **where**
*seqi-abs* (*SEQ* Γ *P* Δ *Q*) = (*set* Γ ∪ *Atom*‘*set P* ⇒ *set* Δ ∪ *Atom*‘*set Q*)

The following theorem states both, correctness and completeness of our implementation

**theorem** *holdsi-holds*: *holdsi S* ⟷ *holds* (*seqi-abs S*)
**apply** (*induction S rule*: *holdsi.induct*)
**apply** (*simp-all add*: *standard-axioms auxiliary-axioms*)
**apply** (*simp-all add*:
*insert-commute*[*of* - *not F* **for** *F*]
*insert-commute*[*of* - *F and G* **for** *F G*]
*insert-commute*[*of* - *F or G* **for** *F G*]
*insert-commute*[*of* - *F imp G* **for** *F G*]
)
**apply** (*simp-all add*: *standard-axioms auxiliary-axioms*)
**apply** (*simp-all add*: *insert-commute*)
**done**

We immediately get an implementation for validity checking

**corollary** *valid-code*[*code*]: ⊨*F* ⟷ *holdsi* (*SEQ* [] [] [*F*] [])
**using** *valid-by-sequent holdsi-holds* **by** *auto*

Isabelle can generate code in your favorite functional language

**export-code** *holdsi valid* **checking** *SML OCaml? Haskell? Scala*

# 5 Test Cases

**no-notation** *Atom* ($ - [*181*] *180*)
**abbreviation** *mk-atom* ($ - [*181*] *180*) **where** *mk-atom s* ≡ *Atom* (*STR s*)

**value** ⊨ $''*A*'' *or not* $''*A*''
**value** ⊨ (($''*P*'' *imp* $''*Q*'') *imp* $''*P*'') *imp* $''*Q*''
**value** ⊨ (($''*Q*'' *imp* $''*P*'') *imp* $''*Q*'') *imp* $''*Q*''

**value** ⊨ $''*A*'' *and* ($''*B*'' *or* $''*C*'') *imp* ($''*A*'' *and* $''*B*'') *or* ($''*A*'' *and* $''*C*'')
**value** ⊨ *not* ($''*A*'' *and* $''*B*'') *imp* *not* $''*A*'' *and* *not* $''*B*''

**end**