# Propositional Proof Systems

Julius Michaelis and Tobias Nipkow

October 26, 2026

**Abstract**

We present a formalization of Sequent Calculus, Natural Deduction, Hilbert Calculus, and Resolution using a deep embedding of propositional formulas. We provide proofs of many of the classical results, including Cut Elimination, Craig's Interpolation, proof transformation between all calculi, and soundness and completeness. Additionally, we formalize the Model Existence Theorem.

## Contents

**1 Formulas** **2**
- 1.1 Derived Connectives . . . 3
- 1.2 Semantics . . . 5
- 1.3 Substitutions . . . 8
- 1.4 Conjunctive Normal Forms . . . 8
  - 1.4.1 Going back: CNFs to formulas . . . 13
  - 1.4.2 Tseytin transformation . . . 14
- 1.5 Implication-only formulas . . . 18
- 1.6 Consistency . . . 19
- 1.7 Compactness . . . 23
- 1.8 Craig Interpolation using Semantics . . . 25

**2 Proof Systems** **26**
- 2.1 Sequent Calculus . . . 26
  - 2.1.1 Contraction . . . 29
  - 2.1.2 Cut . . . 29
  - 2.1.3 Mimicking the original . . . 34
  - 2.1.4 Soundness, Completeness . . . 35
- 2.2 Natural Deduction . . . 38
- 2.3 Hilbert Calculus . . . 41
- 2.4 Resolution . . . 44
  - 2.4.1 Completeness . . . 47

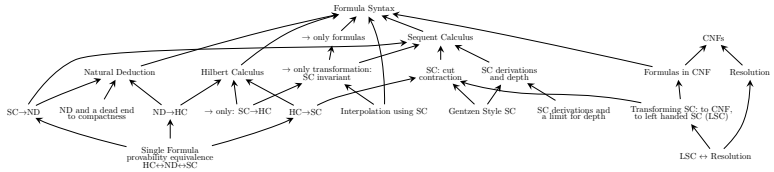


Figure 1: Overview of results considering Proof Transformation

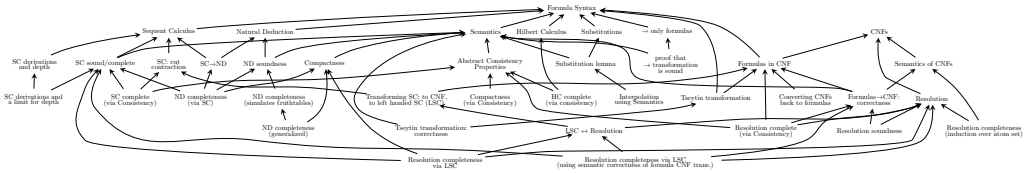


Figure 2: Overview of results considering Semantics

**3 Proof Transformation** **49**
3.1 HC to SC . . . . . 49
3.2 SC to ND . . . . . 49
3.3 ND to HC . . . . . 49
3.4 HC, SC, and ND . . . . . 50
3.5 Transforming SC proofs into proofs of CNFs . . . . . 50
3.6 Converting between Resolution and SC proofs . . . . . 53
3.7 An alternate proof of ND completeness . . . . . 55
3.8 SC and Implication-only formulas . . . . . 56
3.8.1 SC to HC . . . . . 56
3.8.2 Craig Interpolation . . . . . 57

The files of this entry are organized as a web of results that should allow loading only that part of the formalization that the user is interested in. Special care was taken not to mix proofs that require semanics and proofs that talk about transformation between proof systems. An overview of the different theory files and their dependencies can be found in figures 1 and 2.

# 1 Formulas

**theory** *Formulas*
**imports** *Main HOL−Library.Countable*
**begin**

**notation** *insert* (‹- ▷ -› [*56*,*55*] *55*)

**datatype** (*atoms*: *'a*) *formula* =
*Atom 'a*

| *Bot* (‹⊥›)
| *Not 'a formula* (‹¬›)
| *And 'a formula 'a formula* (**infix** ‹∧› *68*)
| *Or 'a formula 'a formula* (**infix** ‹∨› *68*)
| *Imp 'a formula 'a formula* (**infixr** ‹⟶› *68*)

**lemma** *atoms-finite*[*simp*,*intro*!]: *finite* (*atoms F*) ⟨*proof*⟩

**primrec** *subformulae* **where**
*subformulae* ⊥ = [⊥] |
*subformulae* (*Atom k*) = [*Atom k*] |
*subformulae* (*Not F*) = *Not F* # *subformulae F* |
*subformulae* (*And F G*) = *And F G* # *subformulae F* @ *subformulae G* |
*subformulae* (*Imp F G*) = *Imp F G* # *subformulae F* @ *subformulae G* |
*subformulae* (*Or F G*) = *Or F G* # *subformulae F* @ *subformulae G*

**lemma** *atoms-are-subformulae*: *Atom* ‘ *atoms F* ⊆ *set* (*subformulae F*)
⟨*proof*⟩

**lemma** *subsubformulae*: *G* ∈ *set* (*subformulae F*) ⟹ *H* ∈ *set* (*subformulae G*)
⟹ *H* ∈ *set* (*subformulae F*)
⟨*proof*⟩

**lemma** *subformula-atoms*: *G* ∈ *set* (*subformulae F*) ⟹ *atoms G* ⊆ *atoms F*
⟨*proof*⟩

**lemma** *subformulae-self*[*simp*,*intro*]: *F* ∈ *set* (*subformulae F*)
⟨*proof*⟩

**lemma** *subformulas-in-subformulas*:
*G* ∧ *H* ∈ *set* (*subformulae F*) ⟹ *G* ∈ *set* (*subformulae F*) ∧ *H* ∈ *set* (*subformulae F*)
*G* ∨ *H* ∈ *set* (*subformulae F*) ⟹ *G* ∈ *set* (*subformulae F*) ∧ *H* ∈ *set* (*subformulae F*)
*G* ⟶ *H* ∈ *set* (*subformulae F*) ⟹ *G* ∈ *set* (*subformulae F*) ∧ *H* ∈ *set* (*subformulae F*)
¬ *G* ∈ *set* (*subformulae F*) ⟹ *G* ∈ *set* (*subformulae F*)
⟨*proof*⟩

**lemma** *length-subformulae*: *length* (*subformulae F*) = *size F* ⟨*proof*⟩

## 1.1 Derived Connectives

**definition** *Top* (‹⊤›) **where**
⊤ ≡ ⊥ ⟶ ⊥
**lemma** *top-atoms-simp*[*simp*]: *atoms* ⊤ = {} ⟨*proof*⟩

**primrec** *BigAnd* :: *'a formula list* ⇒ *'a formula* (‹⋀-›) **where**
⋀*Nil* = (¬⊥) — essentially, it doesn't matter what I use here. But since I want to use this in CNFs, implication is not a nice thing to have. |
⋀(*F*#*Fs*) = *F* ∧ ⋀*Fs*

**lemma** *atoms-BigAnd*[*simp*]: *atoms* (⋀*Fs*) = ⋃(*atoms* ' *set Fs*)
⟨*proof*⟩

**primrec** *BigOr* :: *'a formula list* ⇒ *'a formula* (‹⋁-›) **where**
⋁*Nil* = ⊥ |
⋁(*F*#*Fs*) = *F* ∨ ⋁*Fs*

Formulas are countable if their atoms are, and *countable-datatype* is really helpful with that.

**instance** *formula* :: (*countable*) *countable* ⟨*proof*⟩

**definition** *prod-unions A B* ≡ *case A of* (*a*,*b*) ⇒ *case B of* (*c*,*d*) ⇒ (*a* ∪ *c*, *b* ∪ *d*)
**primrec** *pn-atoms* **where**
*pn-atoms* (*Atom A*) = ({*A*},{}) |
*pn-atoms Bot* = ({},{}) |
*pn-atoms* (*Not F*) = *prod.swap* (*pn-atoms F*) |
*pn-atoms* (*And F G*) = *prod-unions* (*pn-atoms F*) (*pn-atoms G*) |
*pn-atoms* (*Or F G*) = *prod-unions* (*pn-atoms F*) (*pn-atoms G*) |
*pn-atoms* (*Imp F G*) = *prod-unions* (*prod.swap* (*pn-atoms F*)) (*pn-atoms G*)
**lemma** *pn-atoms-atoms*: *atoms F* = *fst* (*pn-atoms F*) ∪ *snd* (*pn-atoms F*)
⟨*proof*⟩

A very trivial simplifier. Does wonders as a postprocessor for the Harrison-style Craig interpolations.

**context begin**
**definition** *isstop F* ≡ *F* = ¬⊥ ∨ *F* = ⊤
**fun** *simplify-consts* **where**
*simplify-consts* (*Atom k*) = *Atom k* |
*simplify-consts* ⊥ = ⊥ |
*simplify-consts* (*Not F*) = (*let S* = *simplify-consts F in case S of* (*Not G*) ⇒ *G* |
- ⇒
*if isstop S then* ⊥ *else* ¬ *S*) |
*simplify-consts* (*And F G*) = (*let S* = *simplify-consts F*; *T* = *simplify-consts G in*
(
*if S* = ⊥ *then* ⊥ *else*
*if isstop S then T* — not ⊤, *T else*
*if T* = ⊥ *then* ⊥ *else*
*if isstop T then S else*
*if S* = *T then S else*
*S* ∧ *T*)) |
*simplify-consts* (*Or F G*) = (*let S* = *simplify-consts F*; *T* = *simplify-consts G in* (
*if S* = ⊥ *then T else*
*if isstop S then* ⊤ *else*

```
  if T = ⊥ then S else
  if isstop T then ⊤ else
  if S = T then S else
  S ∨ T)) |
simplify-consts (Imp F G) = (let S = simplify-consts F; T = simplify-consts G in
(
  if S = ⊥ then ⊤ else
  if isstop S then T else
  if isstop T then ⊤ else
  if T = ⊥ then ¬ S else
  if S = T then ⊤ else
  case S of Not H ⇒ (case T of Not I ⇒
    I → H | - ⇒
    H ∨ T) | - ⇒
    S → T))

lemma simplify-consts-size-le: size (simplify-consts F) ≤ size F
⟨proof⟩

lemma simplify-const: atoms F = {} ⟹ isstop (simplify-consts F) ∨ (simplify-consts
F) = ⊥
  ⟨proof⟩
value (size ⊤, size (¬⊥))

end


fun all-formulas-of-size where
all-formulas-of-size K 0 = {⊥} ∪ Atom ‘ K |
all-formulas-of-size K (Suc n) = (
  let af = ⋃(set [all-formulas-of-size K m. m ← [0..<Suc n]]) in
  (⋃F∈af.
    (⋃ G∈af. if size F + size G ≤ Suc n then {And F G, Or F G, Imp F G} else
{})
   ∪ (if size F ≤ Suc n then {Not F} else {}))
  ∪ af)


lemma all-formulas-of-size: F ∈ all-formulas-of-size K n ⟷ (size F ≤ Suc n ∧
atoms F ⊆ K) (is ?l ⟷ ?r)
⟨proof⟩


end
```

## 1.2 Semantics

```
theory Sema
imports Formulas
```

**begin**

**type-synonym** *'a valuation* = *'a* ⇒ *bool*

The implicit statement here is that an assignment or valuation is always defined on all atoms (because HOL is a total logic). Thus, there are no unsuitable assignments.

**primrec** *formula-semantics* :: *'a valuation* ⇒ *'a formula* ⇒ *bool* (**infix** ‹⊨› *51*) **where**
$\mathcal{A} \models \mathit{Atom}\ k = \mathcal{A}\ k$ |
- ⊨ ⊥ = *False* |
$\mathcal{A} \models \mathit{Not}\ F = (\neg\ \mathcal{A} \models F)$ |
$\mathcal{A} \models \mathit{And}\ F\ G = (\mathcal{A} \models F \wedge \mathcal{A} \models G)$ |
$\mathcal{A} \models \mathit{Or}\ F\ G = (\mathcal{A} \models F \vee \mathcal{A} \models G)$ |
$\mathcal{A} \models \mathit{Imp}\ F\ G = (\mathcal{A} \models F \longrightarrow \mathcal{A} \models G)$

**abbreviation** *valid* (‹⊨ -› *51*) **where**
$\models F \equiv \forall A.\ A \models F$

**lemma** *irrelevant-atom*[*simp*]: $A \notin \mathit{atoms}\ F \Longrightarrow (\mathcal{A}(A := V)) \models F \longleftrightarrow \mathcal{A} \models F$
⟨*proof*⟩
**lemma** *relevant-atoms-same-semantics*: $\forall k \in \mathit{atoms}\ F.\ \mathcal{A}_1\ k = \mathcal{A}_2\ k \Longrightarrow \mathcal{A}_1 \models F \longleftrightarrow \mathcal{A}_2 \models F$
⟨*proof*⟩

**context begin**

Just a definition more similar to [9, p. 5]. Unfortunately, using this as the main definition would get in the way of automated reasoning all the time.

**private primrec** *formula-semantics-alt* **where**
*formula-semantics-alt* $\mathcal{A}$ (*Atom k*) = $\mathcal{A}\ k$ |
*formula-semantics-alt* $\mathcal{A}$ (*Bot*) = *False* |
*formula-semantics-alt* $\mathcal{A}$ (*Not a*) = (*if formula-semantics-alt* $\mathcal{A}$ *a then False else True*) |
*formula-semantics-alt* $\mathcal{A}$ (*And a b*) = (*if formula-semantics-alt* $\mathcal{A}$ *a then formula-semantics-alt* $\mathcal{A}$ *b else False*) |
*formula-semantics-alt* $\mathcal{A}$ (*Or a b*) = (*if formula-semantics-alt* $\mathcal{A}$ *a then True else formula-semantics-alt* $\mathcal{A}$ *b*) |
*formula-semantics-alt* $\mathcal{A}$ (*Imp a b*) = (*if formula-semantics-alt* $\mathcal{A}$ *a then formula-semantics-alt* $\mathcal{A}$ *b else True*)
**private lemma** *formula-semantics-alt* $\mathcal{A}\ F \longleftrightarrow \mathcal{A} \models F$
⟨*proof*⟩

If you fancy a definition more similar to [3, p. 39], this is probably the closest you can go without going incredibly ugly.

**private primrec** *formula-semantics-tt* **where**
*formula-semantics-tt* $\mathcal{A}$ (*Atom k*) = $\mathcal{A}\ k$ |
*formula-semantics-tt* $\mathcal{A}$ (*Bot*) = *False* |

*formula-semantics-tt* $\mathcal{A}$ (*Not a*) = (*case formula-semantics-tt* $\mathcal{A}$ *a of True* ⇒ *False*
| *False* ⇒ *True*) |
*formula-semantics-tt* $\mathcal{A}$ (*And a b*) = (*case* (*formula-semantics-tt* $\mathcal{A}$ *a*, *formula-semantics-tt* $\mathcal{A}$ *b*) *of*
(*False*, *False*) ⇒ *False*
| (*False*, *True*) ⇒ *False*
| (*True*, *False*) ⇒ *False*
| (*True*, *True*) ⇒ *True*) |
*formula-semantics-tt* $\mathcal{A}$ (*Or a b*) = (*case* (*formula-semantics-tt* $\mathcal{A}$ *a*, *formula-semantics-tt* $\mathcal{A}$ *b*) *of*
(*False*, *False*) ⇒ *False*
| (*False*, *True*) ⇒ *True*
| (*True*, *False*) ⇒ *True*
| (*True*, *True*) ⇒ *True*) |
*formula-semantics-tt* $\mathcal{A}$ (*Imp a b*) = (*case* (*formula-semantics-tt* $\mathcal{A}$ *a*, *formula-semantics-tt* $\mathcal{A}$ *b*) *of*
(*False*, *False*) ⇒ *True*
| (*False*, *True*) ⇒ *True*
| (*True*, *False*) ⇒ *False*
| (*True*, *True*) ⇒ *True*)
**private lemma** $A \models F \longleftrightarrow$ *formula-semantics-tt A F*
⟨*proof*⟩
**end**

**definition** *entailment* :: *'a formula set* ⇒ *'a formula* ⇒ *bool* (‹(- ⊨/ -)› [*53*,*53*] *53*) **where**
$\Gamma \Vdash F \equiv (\forall \mathcal{A}.\ ((\forall G \in \Gamma.\ \mathcal{A} \models G) \longrightarrow (\mathcal{A} \models F)))$

We write entailment differently than semantics ($\models$ vs. $\Vdash$). For humans, it is usually pretty clear what is meant in a specific situation, but it often needs to be decided from context that Isabelle/HOL does not have.

Some helpers for the derived connectives

**lemma** *top-semantics*[*simp*,*intro*!]: $A \models \top$ ⟨*proof*⟩
**lemma** *BigAnd-semantics*[*simp*]: $A \models \bigwedge F \longleftrightarrow (\forall f \in$ *set F*. $A \models f)$ ⟨*proof*⟩
**lemma** *BigOr-semantics*[*simp*]: $A \models \bigvee F \longleftrightarrow (\exists f \in$ *set F*. $A \models f)$ ⟨*proof*⟩

Definitions for sets of formulae, used for compactness and model existence.

**definition** *sat S* ≡ $\exists \mathcal{A}.\ \forall F \in S.\ \mathcal{A} \models F$
**definition** *fin-sat S* ≡ $(\forall s \subseteq S.$ *finite s* ⟶ *sat s*)

**lemma** *entail-sat*: $\Gamma \Vdash \bot \longleftrightarrow \neg$ *sat* $\Gamma$
⟨*proof*⟩

**lemma** *pn-atoms-updates*: *p* ∉ *snd* (*pn-atoms F*) ⟹ *n* ∉ *fst* (*pn-atoms F*) ⟹
((*M* ⊨ *F* ⟶ (*M*(*p* := *True*) ⊨ *F* ∧ *M*(*n* := *False*) ⊨ *F*))) ∧ ((¬(*M* ⊨ *F*))
⟶ ¬(*M*(*n* := *True*) ⊨ *F*) ∧ ¬(*M*(*p* := *False*) ⊨ *F*))
⟨*proof*⟩

**lemma** *const-simplifier-correct*: $\mathcal{A} \models$ *simplify-consts* $F \longleftrightarrow \mathcal{A} \models F$
⟨*proof*⟩

**end**

## 1.3 Substitutions

**theory** *Substitution*
**imports** *Formulas*
**begin**

**primrec** *subst* **where**
*subst* $A$ $F$ (*Atom* $B$) = (*if* $A = B$ *then* $F$ *else Atom* $B$) |
*subst* - - $\bot = \bot$ |
*subst* $A$ $F$ $(G \vee H)$ = (*subst* $A$ $F$ $G \vee$ *subst* $A$ $F$ $H$) |
*subst* $A$ $F$ $(G \wedge H)$ = (*subst* $A$ $F$ $G \wedge$ *subst* $A$ $F$ $H$) |
*subst* $A$ $F$ $(G \rightarrow H)$ = (*subst* $A$ $F$ $G \rightarrow$ *subst* $A$ $F$ $H$) |
*subst* $A$ $F$ $(\neg H)$ = $(\neg$ (*subst* $A$ $F$ $H$))
**term** *subst*
**abbreviation** *subst-syntax* (‹(-[/(-/'//-)])› [70,70] 69) **where**
$A[B\ /\ C] \equiv$ *subst* $C$ $B$ $A$

**lemma** *no-subst*[*simp*]: $k \notin$ *atoms* $F \Longrightarrow F[G\ /\ k] = F$ ⟨*proof*⟩
**lemma** *subst-atoms*: $k \in$ *atoms* $F \Longrightarrow$ *atoms* $(F[G\ /\ k])$ = *atoms* $F - \{k\} \cup$ *atoms* $G$
⟨*proof*⟩

**lemma** *subst-atoms-simp*: *atoms* $(F[G\ /\ k])$ = *atoms* $F - \{k\} \cup$ (*if* $k \in$ *atoms* $F$ *then atoms* $G$ *else* $\{\}$)
⟨*proof*⟩

**end**
**theory** *Substitution-Sema*
**imports** *Substitution Sema*
**begin**

**lemma** *substitution-lemma*: $\mathcal{A} \models F[G\ /\ n] \longleftrightarrow \mathcal{A}(n := \mathcal{A} \models G) \models F$ ⟨*proof*⟩

**end**

## 1.4 Conjunctive Normal Forms

**theory** *CNF*
**imports** *Main HOL−Library.Simps-Case-Conv*
**begin**

**datatype** *'a literal* = *Pos 'a* (‹(-$^{+}$)› [1000] 999) | *Neg 'a* (‹(-$^{-1}$)› [1000] 999)

**type-synonym** *'a clause* = *'a literal set*
**abbreviation** *empty-clause* (‹□› ) **where** $\square \equiv \{\}$ :: *'a clause*

**primrec** *atoms-of-lit* **where**
*atoms-of-lit* (*Pos k*) = *k* |
*atoms-of-lit* (*Neg k*) = *k*
**case-of-simps** *lit-atoms-cases*: *atoms-of-lit.simps*

**definition** *atoms-of-cnf c* = *atoms-of-lit* ‘ $\bigcup$ *c*
**lemma** *atoms-of-cnf-alt*: *atoms-of-cnf c* = $\bigcup$ (((‘) *atoms-of-lit*) ‘ *c*)
⟨*proof*⟩

**lemma** *atoms-of-cnf-Un*: *atoms-of-cnf* (*S* ∪ *T*) = *atoms-of-cnf S* ∪ *atoms-of-cnf T*
⟨*proof*⟩

**term** {*0*$^+$}::*nat clause*
**translations**
{*x*} <= *CONST insert x* □
**term** {*0*$^+$}::*nat clause*

**end**

CNFs alone are nice, but now we want to relate between CNFs and formulas.

**theory** *CNF-Formulas*
**imports** *Formulas CNF*
**begin**

**context begin**

**function** (*sequential*) *nnf* **where**
*nnf* (*Atom k*) = (*Atom k*) |
*nnf* ⊥ = ⊥ |
*nnf* (*Not* (*And F G*)) = *Or* (*nnf* (*Not F*)) (*nnf* (*Not G*)) |
*nnf* (*Not* (*Or F G*)) = *And* (*nnf* (*Not F*)) (*nnf* (*Not G*)) |
*nnf* (*Not* (*Not F*)) = *nnf F* |
*nnf* (*Not* (*Imp F G*)) = *And* (*nnf F*) (*nnf* (*Not G*)) |
*nnf* (*Not F*) = (*Not F*) |
*nnf* (*And F G*) = *And* (*nnf F*) (*nnf G*) |
*nnf* (*Or F G*) = *Or* (*nnf F*) (*nnf G*) |
*nnf* (*Imp F G*) = *Or* (*nnf* (*Not F*)) (*nnf G*)
⟨*proof*⟩ **fun** *nnf-cost* **where**
*nnf-cost* (*Atom* -) = *42* |
*nnf-cost* ⊥ = *42* |
*nnf-cost* (*Not F*) = *Suc* (*nnf-cost F*) |
*nnf-cost* (*And F G*) = *Suc* (*nnf-cost F* + *nnf-cost G*) |
*nnf-cost* (*Or F G*) = *Suc* (*nnf-cost F* + *nnf-cost G*) |

*nnf-cost* (*Imp F G*) = *Suc* (*Suc* (*nnf-cost F* + *nnf-cost G*))

**termination** *nnf* ⟨*proof*⟩

**lemma** *nnf* ((*Atom* (*k*::*nat*)) ⟶ (*Not* ((*Atom l*) ∨ (*Not* (*Atom m*))))) = ¬ (*Atom k*) ∨ (¬ (*Atom l*) ∧ *Atom m*)
⟨*proof*⟩

**fun** *is-lit-plus* **where**
*is-lit-plus* ⊥ = *True* |
*is-lit-plus* (*Not* ⊥) = *True* |
*is-lit-plus* (*Atom* -) = *True* |
*is-lit-plus* (*Not* (*Atom* -)) = *True* |
*is-lit-plus* - = *False*
**case-of-simps** *is-lit-plus-cases*: *is-lit-plus.simps*
**fun** *is-disj* **where**

*is-disj* (*Or F G*) = (*is-lit-plus F* ∧ *is-disj G*) |
*is-disj F* = *is-lit-plus F*
**fun** *is-cnf* **where**

*is-cnf* (*And F G*) = (*is-cnf F* ∧ *is-cnf G*) |
*is-cnf H* = *is-disj H*
**fun** *is-nnf* **where**
*is-nnf* (*Imp F G*) = *False* |
*is-nnf* (*And F G*) = (*is-nnf F* ∧ *is-nnf G*) |
*is-nnf* (*Or F G*) = (*is-nnf F* ∧ *is-nnf G*) |
*is-nnf F* = *is-lit-plus F*

**lemma** *is-nnf-nnf*: *is-nnf* (*nnf F*)
⟨*proof*⟩
**lemma** *nnf-no-imp*: *A* ⟶ *B* ∉ *set* (*subformulae* (*nnf F*))
⟨*proof*⟩
**lemma** *subformulae-nnf*: *is-nnf F* ⟹ *G* ∈ *set* (*subformulae F*) ⟹ *is-nnf G*
⟨*proof*⟩
**lemma** *is-nnf-NotD*: *is-nnf* (¬ *F*) ⟹ (∃ *k*. *F* = *Atom k*) ∨ *F* = ⊥
⟨*proof*⟩

**fun** *cnf* :: *'a formula* ⇒ *'a clause set* **where**
*cnf* (*Atom k*) = {{ $k^+$ }} |
*cnf* (*Not* (*Atom k*)) = {{ $k^{-1}$ }} |
*cnf* ⊥ = {□} |
*cnf* (*Not* ⊥) = {} |
*cnf* (*And F G*) = *cnf F* ∪ *cnf G* |
*cnf* (*Or F G*) = {*C* ∪ *D*| *C D*. *C* ∈ (*cnf F*) ∧ *D* ∈ (*cnf G*)}

**lemma** *cnf-fin*:
**assumes** *is-nnf F*
**shows** *finite* (*cnf F*) *C* ∈ *cnf F* ⟹ *finite C*

⟨*proof*⟩

**fun** *cnf-lists* :: *'a formula* ⇒ *'a literal list list* **where**
*cnf-lists* (*Atom k*) = [[ $k^+$ ]] |
*cnf-lists* (*Not* (*Atom k*)) = [[ $k^{-1}$ ]] |
*cnf-lists* ⊥ = [[]] |
*cnf-lists* (*Not* ⊥) = [] |
*cnf-lists* (*And F G*) = *cnf-lists F* @ *cnf-lists G* |
*cnf-lists* (*Or F G*) = [*f* @ *g*. *f* ← (*cnf-lists F*), *g* ← (*cnf-lists G*)]

**primrec** *form-of-lit* **where**
*form-of-lit* (*Pos k*) = *Atom k* |
*form-of-lit* (*Neg k*) = ¬(*Atom k*)
**case-of-simps** *form-of-lit-cases*: *form-of-lit.simps*

**definition** *disj-of-clause c* ≡ ⋁[*form-of-lit l*. *l* ← *c*]
**definition** *form-of-cnf F* ≡ ⋀[*disj-of-clause c*. *c* ← *F*]
**definition** *cnf-form-of* ≡ *form-of-cnf* ∘ *cnf-lists*
**lemmas** *cnf-form-of-defs* = *cnf-form-of-def form-of-cnf-def disj-of-clause-def*

**lemma** *disj-of-clause-simps*[*simp*]:
*disj-of-clause* [] = ⊥
*disj-of-clause* (*F*#*FF*) = *form-of-lit F* ∨ *disj-of-clause FF*
⟨*proof*⟩

**lemma** *is-cnf-BigAnd*: *is-cnf* (⋀*ls*) ⟷ (∀ *l* ∈ *set ls*. *is-cnf l*)
⟨*proof*⟩ **lemma** *BigOr-is-not-cnf''*: *is-cnf* (⋁*ls*) ⟹ (∀ *l* ∈ *set ls*. *is-lit-plus l*)
⟨*proof*⟩ **lemma** *BigOr-is-not-cnf'*: (∀ *l* ∈ *set ls*. *is-lit-plus l*) ⟹ *is-cnf* (⋁*ls*)
⟨*proof*⟩

**lemma** *BigOr-is-not-cnf*: *is-cnf* (⋁*ls*) ⟷ (∀ *l* ∈ *set ls*. *is-lit-plus l*)
⟨*proof*⟩

**lemma** *is-nnf-BigAnd*[*simp*]: *is-nnf* (⋀*ls*) ⟷ (∀ *l* ∈ *set ls*. *is-nnf l*)
⟨*proof*⟩
**lemma** *is-nnf-BigOr*[*simp*]: *is-nnf* (⋁*ls*) ⟷ (∀ *l* ∈ *set ls*. *is-nnf l*)
⟨*proof*⟩
**lemma** *form-of-lit-is-nnf*[*simp*,*intro*!]: *is-nnf* (*form-of-lit x*)
⟨*proof*⟩
**lemma** *form-of-lit-is-lit*[*simp*,*intro*!]: *is-lit-plus* (*form-of-lit x*)
⟨*proof*⟩
**lemma** *disj-of-clause-is-nnf*[*simp*,*intro*!]: *is-nnf* (*disj-of-clause F*)
⟨*proof*⟩

**lemma** *cnf-form-of-is*: *is-nnf F* ⟹ *is-cnf* (*cnf-form-of F*)
⟨*proof*⟩

**lemma** *nnf-cnf-form*: *is-nnf F* ⟹ *is-nnf* (*cnf-form-of F*)

⟨*proof*⟩

**lemma** *cnf-BigAnd*: *cnf* $(\bigwedge ls) = (\bigcup x \in set\ ls.\ cnf\ x)$
⟨*proof*⟩

**lemma** *cnf-BigOr*: *cnf* $(\bigvee (x\ @\ y)) = \{f \cup g\ |f\ g.\ f \in cnf\ (\bigvee x) \wedge g \in cnf\ (\bigvee y)\}$
⟨*proof*⟩

**lemma** *cnf-cnf*: *is-nnf F* $\Longrightarrow$ *cnf* (*cnf-form-of F*) = *cnf F*
⟨*proof*⟩

**lemma** *is-nnf-nnf-id*: *is-nnf F* $\Longrightarrow$ *nnf F* = *F*
⟨*proof*⟩

**lemma** *disj-of-clause-is*: *is-disj* (*disj-of-clause R*)
⟨*proof*⟩

**lemma** *form-of-cnf-is-nnf*: *is-nnf* (*form-of-cnf R*)
⟨*proof*⟩

**lemma** *cnf-disj*: *cnf* (*disj-of-clause R*) = {*set R*}
⟨*proof*⟩
**lemma** *cnf-disj-ex*: *is-disj F* $\Longrightarrow \exists R.$ *cnf F* = {*R*} $\vee$ *cnf F* = {}
⟨*proof*⟩

**lemma** *cnf-form-of-cnf*: *cnf* (*form-of-cnf S*) = *set* (*map set S*)
⟨*proof*⟩

**lemma** *disj-is-nnf*: *is-disj F* $\Longrightarrow$ *is-nnf F*
⟨*proof*⟩

**lemma** *nnf-BigAnd*: *nnf* $(\bigwedge F) = \bigwedge(map\ nnf\ F)$
⟨*proof*⟩

**end**

**end**
**theory** *CNF-Sema*
**imports** *CNF*
**begin**

**primrec** *lit-semantics* :: $('a \Rightarrow bool) \Rightarrow 'a\ literal \Rightarrow bool$ **where**
*lit-semantics* $\mathcal{A}$ $(k^+) = \mathcal{A}\ k$ |
*lit-semantics* $\mathcal{A}$ $(k^{-1}) = (\neg\mathcal{A}\ k)$
**case-of-simps** *lit-semantics-cases*: *lit-semantics.simps*
**definition** *clause-semantics* **where**

*clause-semantics* $\mathcal{A}$ *C* $\equiv \exists\, L \in C.$ *lit-semantics* $\mathcal{A}$ *L*
**definition** *cnf-semantics* **where**
*cnf-semantics* $\mathcal{A}$ *S* $\equiv \forall\, C \in S.$ *clause-semantics* $\mathcal{A}$ *C*

**end**
**theory** *CNF-Formulas-Sema*
**imports** *CNF-Sema CNF-Formulas Sema*
**begin**

**lemma** *nnf-semantics*: $\mathcal{A} \models$ *nnf F* $\longleftrightarrow$ $\mathcal{A} \models$ *F*
⟨*proof*⟩

**lemma** *cnf-semantics*: *is-nnf F* $\Longrightarrow$ *cnf-semantics* $\mathcal{A}$ (*cnf F*) $\longleftrightarrow$ $\mathcal{A} \models$ *F*
⟨*proof*⟩

**lemma** *cnf-form-semantics*:
**fixes** *F* :: *'a formula*
**assumes** *nnf*: *is-nnf F*
**shows** $\mathcal{A} \models$ *cnf-form-of F* $\longleftrightarrow$ $\mathcal{A} \models$ *F*
⟨*proof*⟩

**corollary** $\exists\, G.\ \mathcal{A} \models F \longleftrightarrow \mathcal{A} \models G \wedge$ *is-cnf G*
⟨*proof*⟩

**end**

### 1.4.1 Going back: CNFs to formulas

**theory** *CNF-To-Formula*
**imports** *CNF-Formulas HOL−Library.List-Lexorder*
**begin**

One downside of CNFs is that they cannot be converted back to formulas as-is in full generality. If we assume an order on the atoms, we can convert finite CNFs.

**instantiation** *literal* :: (*ord*) *ord*
**begin**

**definition**
*literal-less-def*: *xs* < *ys* $\longleftrightarrow$ (
*if atoms-of-lit xs* = *atoms-of-lit ys*
*then* (*case xs of Neg* - ⇒ (*case ys of Pos* - ⇒ *True* | - ⇒ *False*) | - ⇒ *False*)
*else atoms-of-lit xs* < *atoms-of-lit ys*)

**definition**
*literal-le-def*: (*xs* :: - *literal*) ≤ *ys* $\longleftrightarrow$ *xs* < *ys* ∨ *xs* = *ys*

**instance** ⟨*proof*⟩

**end**

**instance** *literal* :: (*linorder*) *linorder*
⟨*proof*⟩

**definition** *formula-of-cnf* **where**
*formula-of-cnf S* ≡ *form-of-cnf* (*sorted-list-of-set* (*sorted-list-of-set* ‘ *S*))

To use the lexicographic order on lists, we first have to convert the clauses to lists, then the set of lists of literals to a list.

**lemma** *simplify-consts* (*formula-of-cnf* ({{*Pos 0*}} :: *nat clause set*)) = *Atom 0*
⟨*proof*⟩

**lemma** *cnf-formula-of-cnf*:
**assumes** *finite S* ∀ *C* ∈ *S*. *finite C*
**shows** *cnf* (*formula-of-cnf S*) = *S*
⟨*proof*⟩

**end**

### 1.4.2 Tseytin transformation

**theory** *Tseytin*
**imports** *Formulas CNF-Formulas*
**begin**

The *cnf* transformation clearly has exponential complexity. If the intention is to use Resolution to decide validity of a formula, that is clearly a deal-breaker for any practical implementation, since validity can be decided by brute force in exponential time. This theory pair shows the Tseytin transformation, a way to transform a formula while preserving validity. The *cnf* of the transformed formula will have clauses with maximally 3 atoms, and an amount of clauses linear in the size of the formula, at the cost of introducing one new atom for each subformula of *F* (i.e. *size F* many).

**definition** *pair-fun-upd f p* ≡ (*case p of* (*k*,*v*) ⇒ *fun-upd f k v*)

**lemma** *fold-pair-upd-triv*: *A* ∉ *fst* ‘ *set U* ⟹ *foldl pair-fun-upd F U A* = *F A*
⟨*proof*⟩

**lemma** *distinct-pair-update-one*: (*k*,*v*) ∈ *set U* ⟹ *distinct* (*map fst U*) ⟹ *foldl pair-fun-upd F U k* = *v*
⟨*proof*⟩

**lemma** *distinct-zipunzip*: *distinct xs* ⟹ *distinct* (*map fst* (*zip xs ys*)) ⟨*proof*⟩

**lemma** *foldl-pair-fun-upd-map-of*: *distinct* (*map fst U*) $\Longrightarrow$ *foldl pair-fun-upd F U* = ($\lambda$*k. case map-of U k of Some v* $\Rightarrow$ *v* | *None* $\Rightarrow$ *F k*)
⟨*proof*⟩

**lemma** *map-of-map-apsnd*: *map-of* (*map* (*apsnd t*) *M*) = *map-option t* $\circ$ (*map-of M*)
⟨*proof*⟩

**definition** *biimp* (**infix** ‹$\leftrightarrow$› *67*) **where** *F* $\leftrightarrow$ *G* $\equiv$ (*F* $\rightarrow$ *G*) $\wedge$ (*G* $\rightarrow$ *F*)
**lemma** *atoms-biimp*[*simp*]: *atoms* (*F* $\leftrightarrow$ *G*) = *atoms F* $\cup$ *atoms G*
⟨*proof*⟩
**lemma** *biimp-size*[*simp*]: *size* (*F* $\leftrightarrow$ *G*) = (*2* $*$ (*size F* + *size G*)) + *3*
⟨*proof*⟩

**locale** *freshstuff* =
**fixes** *fresh* :: *'a set* $\Rightarrow$ *'a*
**assumes** *isfresh*: *finite S* $\Longrightarrow$ *fresh S* $\notin$ *S*
**begin**

**primrec** *nfresh* **where**
*nfresh S 0* = [] |
*nfresh S* (*Suc n*) = (*let f* = *fresh S in f* # *nfresh* (*f* $\triangleright$ *S*) *n*)

**lemma** *length-nfresh*: *length* (*nfresh S n*) = *n*
⟨*proof*⟩

**lemma** *nfresh-isfresh*: *finite S* $\Longrightarrow$ *set* (*nfresh S n*) $\cap$ *S* = {}
⟨*proof*⟩

**lemma** *nfresh-distinct*: *finite S* $\Longrightarrow$ *distinct* (*nfresh S n*)
⟨*proof*⟩

**definition** *tseytin-assmt F* $\equiv$ *let SF* = *remdups* (*subformulae F*) *in zip* (*nfresh* (*atoms F*) (*length SF*)) *SF*

**lemma** *tseytin-assmt-distinct*: *distinct* (*map fst* (*tseytin-assmt F*))
⟨*proof*⟩

**lemma** *tseytin-assmt-has*: *G* $\in$ *set* (*subformulae F*) $\Longrightarrow$ $\exists$ *n*. (*n*,*G*) $\in$ *set* (*tseytin-assmt F*)
⟨*proof*⟩

**lemma** *tseytin-assmt-new-atoms*: (*k*,*l*) $\in$ *set* (*tseytin-assmt F*) $\Longrightarrow$ *k* $\notin$ *atoms F*
⟨*proof*⟩

**primrec** *tseytin-tran1* **where**

*tseytin-tran1 S (Atom k) = [Atom k ↔ S (Atom k)] |*
*tseytin-tran1 S ⊥ = [⊥ ↔ S ⊥] |*
*tseytin-tran1 S (Not F) = [S (Not F) ↔ Not (S F)] @ tseytin-tran1 S F |*
*tseytin-tran1 S (And F G) = [S (And F G) ↔ And (S F) (S G)] @ tseytin-tran1 S F @ tseytin-tran1 S G |*
*tseytin-tran1 S (Or F G) = [S (Or F G) ↔ Or (S F) (S G)] @ tseytin-tran1 S F @ tseytin-tran1 S G |*
*tseytin-tran1 S (Imp F G) = [S (Imp F G) ↔ Imp (S F) (S G)] @ tseytin-tran1 S F @ tseytin-tran1 S G*
**definition** *tseytin-toatom F ≡ Atom ∘ the ∘ map-of (map (λ(a,b). (b,a)) (tseytin-assmt F))*
**definition** *tseytin-tran F ≡ ⋀(let S = tseytin-toatom F in S F # tseytin-tran1 S F)*

**lemma** *distinct-snd-tseytin-assmt*: *distinct (map snd (tseytin-assmt F))*
⟨*proof*⟩

**lemma** *tseytin-assmt-backlookup*: **assumes** *J ∈ set (subformulae F)*
**shows** *(the (map-of (map (λ(a, b). (b, a)) (tseytin-assmt F)) J), J) ∈ set (tseytin-assmt F)*
⟨*proof*⟩

**lemma** *tseytin-tran-small-clauses*: *∀ C ∈ cnf (nnf (tseytin-tran F)). card C ≤ 3*
⟨*proof*⟩

**lemma** *tseytin-tran-few-clauses*: *card (cnf (nnf (tseytin-tran F))) ≤ 3 * size F + 1*
⟨*proof*⟩

**lemma** *tseytin-tran-new-atom-count*: *card (atoms (tseytin-tran F)) ≤ size F + card (atoms F)*
⟨*proof*⟩

**end**

**definition** *freshnat S ≡ Suc (Max (0 ▷ S))*
**primrec** *nfresh-natcode* **where**
*nfresh-natcode S 0 = [] |*
*nfresh-natcode S (Suc n) = (let f = freshnat S in f # nfresh-natcode (f ▷ S) n)*
**interpretation** *freshnats*: *freshstuff freshnat* ⟨*proof*⟩

**lemma** [*code-unfold*]: *freshnats.nfresh = nfresh-natcode*
⟨*proof*⟩
**lemmas** *freshnats-code*[*code-unfold*] = *freshnats.tseytin-tran-def freshnats.tseytin-toatom-def freshnats.tseytin-assmt-def freshnats.nfresh.simps*

**lemma** *freshnats.tseytin-tran (Atom 0 → (¬ (Atom 1))) = ⋀[*
*Atom 2,*
*Atom 2 ↔ Atom 3 → Atom 4,*

*Atom 0* $\leftrightarrow$ *Atom 3*,
*Atom 4* $\leftrightarrow$ $\neg$ (*Atom 5*),
*Atom 1* $\leftrightarrow$ *Atom 5*
] (**is** *?l* = *?r*)
⟨*proof*⟩

**end**
**theory** *Tseytin-Sema*
**imports** *Sema Tseytin*
**begin**

**lemma** *biimp-simp*[*simp*]: $\mathcal{A} \models F \leftrightarrow G \longleftrightarrow (\mathcal{A} \models F \longleftrightarrow \mathcal{A} \models G)$
⟨*proof*⟩

**locale** *freshstuff-sema* = *freshstuff*
**begin**

**definition** *tseytin-update* $\mathcal{A}$ *F* ≡ (*let U* = *map* (*apsnd* (*formula-semantics* $\mathcal{A}$))
(*tseytin-assmt F*) *in foldl pair-fun-upd* $\mathcal{A}$ *U*)

**lemma** *tseyting-update-keep-subformula-sema*: *G* ∈ *set* (*subformulae F*) ⟹ *tseytin-update*
$\mathcal{A}$ *F* $\models$ *G* $\longleftrightarrow$ $\mathcal{A}$ $\models$ *G*
⟨*proof*⟩

**lemma** (*k*,*G*) ∈ *set* (*tseytin-assmt F*) ⟹ *tseytin-update* $\mathcal{A}$ *F k* $\longleftrightarrow$ *tseytin-update*
$\mathcal{A}$ *F* $\models$ *G*
⟨*proof*⟩

**lemma** *tseytin-updates*: (*k*,*G*) ∈ *set* (*tseytin-assmt F*) ⟹ *tseytin-update* $\mathcal{A}$ *F k*
$\longleftrightarrow$ *tseytin-update* $\mathcal{A}$ *F* $\models$ *G*
⟨*proof*⟩

**lemma** *tseytin-tran1*: *G* ∈ *set* (*subformulae F*) ⟹ *H* ∈ *set* (*tseytin-tran1 S G*)
⟹ ∀ *J* ∈ *set* (*subformulae F*). *tseytin-update* $\mathcal{A}$ *F* $\models$ *J* $\longleftrightarrow$ *tseytin-update* $\mathcal{A}$ *F*
$\models$ (*S J*) ⟹ *tseytin-update* $\mathcal{A}$ *F* $\models$ *H*
⟨*proof*⟩

**lemma** *all-tran-formulas-validated*: ∀ *J* ∈ *set* (*subformulae F*). *tseytin-update* $\mathcal{A}$ *F*
$\models$ *J* $\longleftrightarrow$ *tseytin-update* $\mathcal{A}$ *F* $\models$ (*tseytin-toatom F J*)
⟨*proof*⟩

**lemma** *tseytin-tran-equisat*: $\mathcal{A} \models F \longleftrightarrow$ *tseytin-update* $\mathcal{A}$ *F* $\models$ (*tseytin-tran F*)
⟨*proof*⟩

**lemma** *tseytin-tran1-orig-connection*: *G* ∈ *set* (*subformulae F*) ⟹ (∀ *H*∈*set* (*tseytin-tran1*
(*tseytin-toatom F*) *G*). $\mathcal{A}$ $\models$ *H*) ⟹
$\mathcal{A}$ $\models$ *G* $\longleftrightarrow$ $\mathcal{A}$ $\models$ *tseytin-toatom F G*
⟨*proof*⟩

**lemma** *tseytin-untran*: $\mathcal{A} \models (\textit{tseytin-tran}\ F) \Longrightarrow \mathcal{A} \models F$
⟨*proof*⟩
**lemma** *tseytin-tran-equiunsatisfiable*: $\models \neg F \longleftrightarrow \models \neg\ (\textit{tseytin-tran}\ F)$ (**is** *?l* $\longleftrightarrow$ *?r*)
⟨*proof*⟩

**end**

**interpretation** *freshsemanats*: *freshstuff-sema freshnat*
⟨*proof*⟩
**print-theorems**

**end**

## 1.5 Implication-only formulas

**theory** *MiniFormulas*
**imports** *Formulas*
**begin**

**fun** *is-mini-formula* **where**
*is-mini-formula* (*Atom* -) = *True* |
*is-mini-formula* $\bot$ = *True* |
*is-mini-formula* (*Imp F G*) = (*is-mini-formula F* $\wedge$ *is-mini-formula G*) |
*is-mini-formula* - = *False*

The similarity between these "mini" formulas and Johansson's minimal calculus of implications [8] is mostly in name. Johansson does replace $\neg\ F$ by $F \longrightarrow \bot$ in one place, but generally keeps it. The main focus of [8] is on removing rules from Calculi anyway, not on removing connectives. We are only borrowing the name.

**primrec** *to-mini-formula* **where**
*to-mini-formula* (*Atom k*) = *Atom k* |
*to-mini-formula* $\bot$ = $\bot$ |
*to-mini-formula* (*Imp F G*) = *to-mini-formula F* $\longrightarrow$ *to-mini-formula G* |
*to-mini-formula* (*Not F*) = *to-mini-formula F* $\longrightarrow \bot$ |
*to-mini-formula* (*And F G*) = (*to-mini-formula F* $\longrightarrow$ (*to-mini-formula G* $\longrightarrow \bot$)) $\longrightarrow \bot$ |
*to-mini-formula* (*Or F G*) = (*to-mini-formula F* $\longrightarrow \bot$) $\longrightarrow$ *to-mini-formula G*

**lemma** *to-mini-is-mini*[*simp*,*intro*!]: *is-mini-formula* (*to-mini-formula F*)
⟨*proof*⟩
**lemma** *mini-to-mini*: *is-mini-formula F* $\Longrightarrow$ *to-mini-formula F* = *F*
⟨*proof*⟩
**corollary** *mini-mini*[*simp*]: *to-mini-formula* (*to-mini-formula F*) = *to-mini-formula F*
⟨*proof*⟩

We could have used an arbitrary other combination, e.g. *Atom*, $\neg$, and ($\wedge$). The choice for *Atom*, $\bot$, and ($\rightarrow$) was made because it is (to the best of my knowledge) the only combination that only requires three elements and verifies:

**lemma** *mini-formula-atoms*: *atoms* (*to-mini-formula F*) = *atoms F*
⟨*proof*⟩

(The story would be different if we had different junctors, e.g. if we allowed a NAND.)

**end**
**theory** *MiniFormulas-Sema*
**imports** *MiniFormulas Sema*
**begin**

**lemma** $A \models F \longleftrightarrow A \models$ *to-mini-formula F*
⟨*proof*⟩

**end**

## 1.6 Consistency

We follow the proofs by Melvin Fitting [2].

**theory** *Consistency*
**imports** *Sema*
**begin**

**definition** *Hintikka S* ≡ (
$\bot \notin S$
$\wedge$ ($\forall$ *k*. *Atom k* $\in S \longrightarrow \neg$ (*Atom k*) $\in S \longrightarrow$ *False*)
$\wedge$ ($\forall F\ G.\ F \wedge G \in S \longrightarrow F \in S \wedge G \in S$)
$\wedge$ ($\forall F\ G.\ F \vee G \in S \longrightarrow F \in S \vee G \in S$)
$\wedge$ ($\forall F\ G.\ F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S$)
$\wedge$ ($\forall F.\ \neg\ (\neg F) \in S \longrightarrow F \in S$)
$\wedge$ ($\forall F\ G.\ \neg(F \wedge G) \in S \longrightarrow \neg\ F \in S \vee \neg\ G \in S$)
$\wedge$ ($\forall F\ G.\ \neg(F \vee G) \in S \longrightarrow \neg\ F \in S \wedge \neg\ G \in S$)
$\wedge$ ($\forall F\ G.\ \neg(F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg\ G \in S$)
)

**lemma** *Hintikka* {*Atom 0* $\wedge$ ((¬ (*Atom 1*)) $\rightarrow$ *Atom 2*), ((¬ (*Atom 1*)) $\rightarrow$ *Atom 2*), *Atom 0*, ¬(¬ (*Atom 1*)), *Atom 1*}
⟨*proof*⟩

**theorem** *Hintikkas-lemma*:
**assumes** *H*: *Hintikka S*
**shows** *sat S*
⟨*proof*⟩

**definition** *pcp C* ≡ ($\forall S \in C$.

$\bot \notin S$
$\wedge$ $(\forall k.\ Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \in S \longrightarrow False)$
$\wedge$ $(\forall F\ G.\ F \wedge G \in S \longrightarrow F \triangleright G \triangleright S \in C)$
$\wedge$ $(\forall F\ G.\ F \vee G \in S \longrightarrow F \triangleright S \in C \vee G \triangleright S \in C)$
$\wedge$ $(\forall F\ G.\ F \rightarrow G \in S \longrightarrow \neg F \triangleright S \in C \vee G \triangleright S \in C)$
$\wedge$ $(\forall F.\ \neg\ (\neg F) \in S \longrightarrow F \triangleright S \in C)$
$\wedge$ $(\forall F\ G.\ \neg(F \wedge G) \in S \longrightarrow \neg\ F \triangleright S \in C \vee \neg\ G \triangleright S \in C)$
$\wedge$ $(\forall F\ G.\ \neg(F \vee G) \in S \longrightarrow \neg\ F \triangleright \neg\ G \triangleright S \in C)$
$\wedge$ $(\forall F\ G.\ \neg(F \rightarrow G) \in S \longrightarrow F \triangleright \neg\ G \triangleright S \in C)$
)

**lemma** *pcp {} pcp {{}} pcp {{Atom 0}}* ⟨*proof*⟩
**lemma** *pcp* {{(¬ (*Atom 1*)) → *Atom 2*},
{((¬ (*Atom 1*)) → *Atom 2*), ¬(¬ (*Atom 1*))},
{((¬ (*Atom 1*)) → *Atom 2*), ¬(¬ (*Atom 1*)), *Atom 1*}} ⟨*proof*⟩

Fitting uses uniform notation [10] for the definition of *pcp*. We try to mimic this, more to see whether it works than because it is ultimately necessary.

**inductive** *Con* :: *'a formula => 'a formula => 'a formula => bool* **where**
*Con (And F G) F G |*
*Con (Not (Or F G)) (Not F) (Not G) |*
*Con (Not (Imp F G)) F (Not G) |*
*Con (Not (Not F)) F F*

**inductive** *Dis* :: *'a formula => 'a formula => 'a formula => bool* **where**
*Dis (Or F G) F G |*
*Dis (Imp F G) (Not F) G |*
*Dis (Not (And F G)) (Not F) (Not G) |*
*Dis (Not (Not F)) F F*

**lemma** *Con (Not (Not F)) F F Dis (Not (Not F)) F F* ⟨*proof*⟩

**lemma** *con-dis-simps*:
*Con a1 a2 a3* = (*a1 = a2* ∧ *a3* ∨ (∃ *F G. a1* = ¬ (*F* ∨ *G*) ∧ *a2* = ¬ *F* ∧ *a3* = ¬ *G*) ∨ (∃ *G. a1* = ¬ (*a2* → *G*) ∧ *a3* = ¬ *G*) ∨ *a1* = ¬ (¬ *a2*) ∧ *a3* = *a2*)
*Dis a1 a2 a3* = (*a1 = a2* ∨ *a3* ∨ (∃ *F G. a1* = *F* → *G* ∧ *a2* = ¬ *F* ∧ *a3* = *G*) ∨ (∃ *F G. a1* = ¬ (*F* ∧ *G*) ∧ *a2* = ¬ *F* ∧ *a3* = ¬ *G*) ∨ *a1* = ¬ (¬ *a2*) ∧ *a3* = *a2*)
⟨*proof*⟩

**lemma** *Hintikka-alt*: *Hintikka S* = (
$\bot \notin S$
$\wedge$ $(\forall k.\ Atom\ k \in S \longrightarrow \neg\ (Atom\ k) \in S \longrightarrow False)$
$\wedge$ $(\forall F\ G\ H.\ Con\ F\ G\ H \longrightarrow F \in S \longrightarrow G \in S \wedge H \in S)$
$\wedge$ $(\forall F\ G\ H.\ Dis\ F\ G\ H \longrightarrow F \in S \longrightarrow G \in S \vee H \in S)$
)

⟨*proof*⟩

**lemma** *pcp-alt*: *pcp C* = (∀ *S* ∈ *C*.
⊥ ∉ *S*
∧ (∀ *k*. *Atom k* ∈ *S* ⟶ ¬ (*Atom k*) ∈ *S* ⟶ *False*)
∧ (∀ *F G H*. *Con F G H* ⟶ *F* ∈ *S* ⟶ *G* ▷ *H* ▷ *S* ∈ *C*)
∧ (∀ *F G H*. *Dis F G H* ⟶ *F* ∈ *S* ⟶ *G* ▷ *S* ∈ *C* ∨ *H* ▷ *S* ∈ *C*)
)
⟨*proof*⟩

**definition** *subset-closed C* ≡ (∀ *S* ∈ *C*. ∀ *s*⊆*S*. *s* ∈ *C*)
**definition** *finite-character C* ≡ (∀ *S*. *S* ∈ *C* ⟷ (∀ *s* ⊆ *S*. *finite s* ⟶ *s* ∈ *C*))

**lemma** *ex1*: *pcp C* ⟹ ∃ *C'*. *C* ⊆ *C'* ∧ *pcp C'* ∧ *subset-closed C'*
⟨*proof*⟩

**lemma** *sallI*: (⋀*s*. *s* ⊆ *S* ⟹ *P s*) ⟹ ∀ *s* ⊆ *S*. *P s* ⟨*proof*⟩

**lemma** *ex2*:
**assumes** *fc*: *finite-character C*
**shows** *subset-closed C*
⟨*proof*⟩

**lemma**
**assumes** *C*: *pcp C*
**assumes** *S*: *subset-closed C*
**shows** *ex3*: ∃ *C'*. *C* ⊆ *C'* ∧ *pcp C'* ∧ *finite-character C'*
⟨*proof*⟩

**primrec** *pcp-seq* **where**
*pcp-seq C S 0* = *S* |
*pcp-seq C S* (*Suc n*) = (
*let Sn* = *pcp-seq C S n*; *Sn1* = *from-nat n* ▷ *Sn in*
*if Sn1* ∈ *C then Sn1 else Sn*
)

**lemma** *pcp-seq-in*: *pcp C* ⟹ *S* ∈ *C* ⟹ *pcp-seq C S n* ∈ *C*
⟨*proof*⟩

**lemma** *pcp-seq-mono*: *n* ≤ *m* ⟹ *pcp-seq C S n* ⊆ *pcp-seq C S m*
⟨*proof*⟩

**lemma** *pcp-seq-UN*: ⋃{*pcp-seq C S n*|*n*. *n* ≤ *m*} = *pcp-seq C S m*
⟨*proof*⟩

**lemma** *wont-get-added*: (*F* :: ('*a* :: *countable*) *formula*) ∉ *pcp-seq C S* (*Suc* (*to-nat* *F*)) ⟹ *F* ∉ *pcp-seq C S* (*Suc* (*to-nat F*) + *n*)

We don't necessarily have *n* = *to-nat* (*from-nat n*), so this doesn't hold.

⟨*proof*⟩

**definition** *pcp-lim C S* ≡ ⋃{*pcp-seq C S n*|*n*. *True*}

**lemma** *pcp-seq-sub*: *pcp-seq C S n* ⊆ *pcp-lim C S*
⟨*proof*⟩

**lemma** *pcp-lim-inserted-at-ex*: *x* ∈ *pcp-lim C S* ⟹ ∃ *k*. *x* ∈ *pcp-seq C S k*
⟨*proof*⟩

**lemma** *pcp-lim-in*:
**assumes** *c*: *pcp C*
**and** *el*: *S* ∈ *C*
**and** *sc*: *subset-closed C*
**and** *fc*: *finite-character C*
**shows** *pcp-lim C S* ∈ *C* (**is** *?cl* ∈ *C*)
⟨*proof*⟩

**lemma** *cl-max*:
**assumes** *c*: *pcp C*
**assumes** *sc*: *subset-closed C*
**assumes** *el*: *K* ∈ *C*
**assumes** *su*: *pcp-lim C S* ⊆ *K*
**shows** *pcp-lim C S* = *K* (**is** *?e*)
⟨*proof*⟩

**lemma** *cl-max′*:
**assumes** *c*: *pcp C*
**assumes** *sc*: *subset-closed C*
**shows** *F* ▷ *pcp-lim C S* ∈ *C* ⟹ *F* ∈ *pcp-lim C S*
*F* ▷ *G* ▷ *pcp-lim C S* ∈ *C* ⟹ *F* ∈ *pcp-lim C S* ∧ *G* ∈ *pcp-lim C S*
⟨*proof*⟩

**lemma** *pcp-lim-Hintikka*:
**assumes** *c*: *pcp C*
**assumes** *sc*: *subset-closed C*
**assumes** *fc*: *finite-character C*
**assumes** *el*: *S* ∈ *C*
**shows** *Hintikka* (*pcp-lim C S*)
⟨*proof*⟩

**theorem** *pcp-sat*: — model existence theorem
**fixes** *S* :: *′a* :: *countable formula set*
**assumes** *c*: *pcp C*
**assumes** *el*: *S* ∈ *C*
**shows** *sat S*
⟨*proof*⟩

**end**

## 1.7 Compactness

**theory** *Compactness*
**imports** *Sema*
**begin**

**lemma** *fin-sat-extend*: *fin-sat S* $\Longrightarrow$ *fin-sat* (*insert F S*) $\vee$ *fin-sat* (*insert* ($\neg F$) *S*)
⟨*proof*⟩

**context**
**begin**

**lemma** *fin-sat-antimono*: *fin-sat F* $\Longrightarrow$ *G* $\subseteq$ *F* $\Longrightarrow$ *fin-sat G* ⟨*proof*⟩
**lemmas** *fin-sat-insert* = *fin-sat-antimono*[*OF - subset-insertI*]

**primrec** *extender* :: *nat* $\Rightarrow$ ('*a* :: *countable*) *formula set* $\Rightarrow$ '*a formula set* **where**
*extender 0 S* = *S* |
*extender* (*Suc n*) *S* = (
  *let r* = *extender n S*;
  *rt* = *insert* (*from-nat n*) *r*;
  *rf* = *insert* ($\neg$(*from-nat n*)) *r*
  *in if fin-sat rf then rf else rt*
)

**private lemma** *extender-fin-sat*: *fin-sat S* $\Longrightarrow$ *fin-sat* (*extender n S*)
⟨*proof*⟩

**definition** *extended S* = $\bigcup$ {*extender n S*|*n*. *True*}

**lemma** *extended-max*: *F* $\in$ *extended S* $\vee$ *Not F* $\in$ *extended S*
⟨*proof*⟩ **lemma** *extender-Sucset*: *extender k S* $\subseteq$ *extender* (*Suc k*) *S* ⟨*proof*⟩ **lemma**
*extender-deeper*: *F* $\in$ *extender k S* $\Longrightarrow$ *k* $\leq$ *l* $\Longrightarrow$ *F* $\in$ *extender l S* ⟨*proof*⟩ **lemma**
*extender-subset*: *S* $\subseteq$ *extender k S*
⟨*proof*⟩

**lemma** *extended-fin-sat*:
  **assumes** *fin-sat S*
  **shows** *fin-sat* (*extended S*)
⟨*proof*⟩

**lemma** *extended-superset*: *S* $\subseteq$ *extended S* ⟨*proof*⟩

**lemma** *extended-complem*:
  **assumes** *fs*: *fin-sat S*
  **shows** (*F* $\in$ *extended S*) $\neq$ (*Not F* $\in$ *extended S*)
⟨*proof*⟩

**lemma** *not-fin-sat-extended-UNIV*: **fixes** $S :: 'a ::$ *countable formula set* **assumes** $\neg$*fin-sat* $S$ **shows** *extended* $S = UNIV$

Note that this crucially depends on the fact that we check *first* whether adding $\neg\ F$ makes the set not satisfiable, and add $F$ otherwise *without any further checks.* The proof of compactness does (to the best of my knowledge) depend on neither of these two facts.

⟨*proof*⟩

**lemma** *extended-tran*: $S \subseteq T \Longrightarrow$ *extended* $S \subseteq$ *extended* $T$

This lemma doesn't hold: think of making S empty and inserting a formula into T s.t. it can never be satisfied simultaneously with the first non-tautological formula in the extension S. Showing that this is possible is not worth the effort, since we can't influence the ordering of formulae. But we showed it anyway.

⟨*proof*⟩
**lemma** *extended-not-increasing*: $\exists S\ T.$ *fin-sat* $S \wedge$ *fin-sat* $T \wedge \neg\ (S \subseteq T \longrightarrow$ *extended* $S \subseteq$ *extended* $(T :: 'a ::$ *countable formula set*$))$
⟨*proof*⟩ **lemma** *not-in-extended-FE*: *fin-sat* $S \Longrightarrow (\neg$*sat* (*insert* (*Not* $F$) $G)) \Longrightarrow$ $F \notin$ *extended* $S \Longrightarrow G \subseteq$ *extended* $S \Longrightarrow$ *finite* $G \Longrightarrow$ *False*
⟨*proof*⟩

**lemma** *extended-id*: *extended* (*extended* $S$) = *extended* $S$
⟨*proof*⟩

**lemma** *ext-model*:
**assumes** $r$: *fin-sat* $S$
**shows** $(\lambda k.$ *Atom* $k \in$ *extended* $S) \models F \longleftrightarrow F \in$ *extended* $S$
⟨*proof*⟩

**theorem** *compactness*:
**fixes** $S :: 'a ::$ *countable formula set*
**shows** *sat* $S \longleftrightarrow$ *fin-sat* $S$ (**is** *?l* = *?r*)
⟨*proof*⟩

**corollary** *compact-entailment*:
**fixes** $F :: 'a ::$ *countable formula*
**assumes** *fent*: $\Gamma \models F$
**shows** $\exists \Gamma'.$ *finite* $\Gamma' \wedge \Gamma' \subseteq \Gamma \wedge \Gamma' \models F$
⟨*proof*⟩

**corollary** *compact-to-formula*:
**fixes** $F :: 'a ::$ *countable formula*
**assumes** *fent*: $\Gamma \models F$
**obtains** $\Gamma'$ **where** *set* $\Gamma' \subseteq \Gamma \models (\bigwedge \Gamma') \rightarrow F$

⟨*proof*⟩

**end**

**end**
**theory** *Compactness-Consistency*
**imports** *Consistency*
**begin**

**theorem** *sat S* ⟷ *fin-sat* (*S* :: *'a* :: *countable formula set*) (**is** *?l* = *?r*)
⟨*proof*⟩

**end**

## 1.8 Craig Interpolation using Semantics

**theory** *Sema-Craig*
**imports** *Substitution-Sema*
**begin**

Semantic proof of Craig interpolation following Harrison [5].

**lemma** *subst-true-false*:
  **assumes** $\mathcal{A} \models F$
  **shows** $\mathcal{A} \models ((F[\top \,/\, n]) \vee (F[\bot \,/\, n]))$
⟨*proof*⟩

**theorem** *interpolation*:
  **assumes** $\models \Gamma \rightarrow \Delta$
  **obtains** $\varrho$ **where**
    $\models \Gamma \rightarrow \varrho \models \varrho \rightarrow \Delta$
    *atoms* $\varrho \subseteq$ *atoms* $\Gamma$
    *atoms* $\varrho \subseteq$ *atoms* $\Delta$
⟨*proof*⟩

The above proof is constructive, and it is actually very easy to write a procedure down.

**function** *interpolate* **where**
*interpolate F H* = (
*let K* = *atoms F* − *atoms H in*
  *if K* = {}
  *then F*
  *else* (
    *let k* = *Min K*
    *in interpolate* ((*F*[⊤ / *k*]) ∨ (*F*[⊥ / *k*])) *H*
  )
) ⟨*proof*⟩

Showing termination is slightly technical...

**termination** *interpolate*

⟨*proof*⟩

Surprisingly, *interpolate* is even executable, despite all the set operations involving *atoms*

**lemma** *interpolate* (*And* (*Atom* (*0*::*nat*)) (*Atom 1*)) (*Or* (*Atom 1*) (*Atom 2*)) =
(⊤ ∧ *Atom 1*) ∨ (⊥ ∧ *Atom 1*) ⟨*proof*⟩
**value**[*code*] *simplify-consts* (*interpolate* (*And* (*Atom* (*0*::*nat*)) (*Atom 1*)) (*Or* (*Atom 1*) (*Atom 2*)))

**lemma** *let P* = *Atom* (*0* :: *nat*); *Q* = *Atom 1*; *R* = *Atom 2*; *T* = *Atom 3*;
$\varphi$ = (¬(*P* ∧ *Q*)) → (¬*R* ∧ *Q*);
$\psi$ = (*T* → *P*) ∨ (*T* → (¬*R*));
*I* = *interpolate* $\varphi$ $\psi$ *in*
(*size I*) = *23* ∧ *simplify-consts I* = *Atom 2* → *Atom 0*
⟨*proof*⟩

**theorem** *nonexistential-interpolation*:
**assumes** ⊨ *F* → *H*
**shows**
⊨ *F* → *interpolate F H* (**is** *?t1*) ⊨ *interpolate F H* → *H* (**is** *?t2*)
*atoms* (*interpolate F H*) ⊆ *atoms F* ∩ *atoms H* (**is** *?s*)
⟨*proof*⟩

So no, the proof is by no means easier this way. Admittedly, part of the fuzz is due to *Min*, but replacing atoms with something that returns lists doesn't make it better.

**end**

# 2 Proof Systems

## 2.1 Sequent Calculus

**theory** *SC*
**imports** *Formulas HOL−Library.Multiset*
**begin**

**abbreviation** *msins* (‹-, -› [*56*,*56*] *56*) **where** *x*,*M* == *add-mset x M*

We do not formalize the concept of sequents, only that of sequent calculus derivations.

**inductive** *SCp* :: *'a formula multiset* ⇒ *'a formula multiset* ⇒ *bool* (‹(- ⇒/ -)› [*53*] *53*) **where**
*BotL*: ⊥ ∈# Γ ⟹ Γ⇒Δ |
*Ax*: *Atom k* ∈# Γ ⟹ *Atom k* ∈# Δ ⟹ Γ⇒Δ |
*NotL*: Γ ⇒ *F*,Δ ⟹ *Not F*, Γ ⇒ Δ |
*NotR*: *F*,Γ ⇒ Δ ⟹ Γ ⇒ *Not F*, Δ |
*AndL*: *F*,*G*,Γ ⇒ Δ ⟹ *And F G*, Γ ⇒ Δ |
*AndR*: ⟦ Γ ⇒ *F*,Δ; Γ ⇒ *G*,Δ ⟧ ⟹ Γ ⇒ *And F G*, Δ |

*OrL*: $[\![ F,\Gamma \Rightarrow \Delta;\ G,\Gamma \Rightarrow \Delta ]\!] \Longrightarrow Or\ F\ G,\ \Gamma \Rightarrow \Delta$ |
*OrR*: $\Gamma \Rightarrow F,G,\Delta \Longrightarrow \Gamma \Rightarrow Or\ F\ G,\ \Delta$ |
*ImpL*: $[\![ \Gamma \Rightarrow F,\Delta;\ G,\Gamma \Rightarrow \Delta ]\!] \Longrightarrow Imp\ F\ G,\ \Gamma \Rightarrow \Delta$ |
*ImpR*: $F,\Gamma \Rightarrow G,\Delta \Longrightarrow \Gamma \Rightarrow Imp\ F\ G,\ \Delta$

Many of the proofs here are inspired Troelstra and Schwichtenberg [11].

**lemma**
**shows** *BotL-canonical*[*intro*!]: $\bot,\Gamma \Rightarrow \Delta$
**and** *Ax-canonical*[*intro*!]: $Atom\ k,\Gamma \Rightarrow Atom\ k,\Delta$
⟨*proof*⟩

**lemma** *lem1*: $x \neq y \Longrightarrow x,\ M = y,N \longleftrightarrow x \in\# N \wedge M = y,(N - \{\#x\#\})$
⟨*proof*⟩

**lemma** *lem2*: $x \neq y \Longrightarrow x,\ M = y,\ N \longleftrightarrow y \in\# M \wedge N = x,\ (M - \{\#y\#\})$
⟨*proof*⟩

This is here to deal with a technical problem: the way the simplifier uses $?x$, $?y$, $?M = ?y$, $?x$, $?M$ is really suboptimal for the unification of SC rules.

**lemma** *sc-insertion-ordering*[*simp*]:
$NO\text{-}MATCH\ (I \rightarrow J)\ H \Longrightarrow H,F \rightarrow G,S = F \rightarrow G,H,S$
$NO\text{-}MATCH\ (I \rightarrow J)\ H \Longrightarrow NO\text{-}MATCH\ (I \vee J)\ H \Longrightarrow H,F \vee G,S = F \vee G,H,S$
$NO\text{-}MATCH\ (I \rightarrow J)\ H \Longrightarrow NO\text{-}MATCH\ (I \vee J)\ H \Longrightarrow NO\text{-}MATCH\ (I \wedge J)\ H \Longrightarrow H,F \wedge G,S = F \wedge G,H,S$
$NO\text{-}MATCH\ (I \rightarrow J)\ H \Longrightarrow NO\text{-}MATCH\ (I \vee J)\ H \Longrightarrow NO\text{-}MATCH\ (I \wedge J)\ H \Longrightarrow NO\text{-}MATCH\ (\neg J)\ H \Longrightarrow H,\neg G,S = \neg G,H,S$
$NO\text{-}MATCH\ (I \rightarrow J)\ H \Longrightarrow NO\text{-}MATCH\ (I \vee J)\ H \Longrightarrow NO\text{-}MATCH\ (I \wedge J)\ H \Longrightarrow NO\text{-}MATCH\ (\neg J)\ H \Longrightarrow NO\text{-}MATCH\ (\bot)\ H \Longrightarrow H,\bot,S = \bot,H,S$
$NO\text{-}MATCH\ (I \rightarrow J)\ H \Longrightarrow NO\text{-}MATCH\ (I \vee J)\ H \Longrightarrow NO\text{-}MATCH\ (I \wedge J)\ H \Longrightarrow NO\text{-}MATCH\ (\neg J)\ H \Longrightarrow NO\text{-}MATCH\ (\bot)\ H \Longrightarrow NO\text{-}MATCH\ (Atom\ k)\ H \Longrightarrow H,Atom\ l,S = Atom\ l,H,S$

⟨*proof*⟩

**lemma shows**
$inR1$: $\Gamma \Rightarrow G,\ H,\ \Delta \Longrightarrow \Gamma \Rightarrow H,\ G,\ \Delta$
**and** $inL1$: $G,\ H,\ \Gamma \Rightarrow \Delta \Longrightarrow H,\ G,\ \Gamma \Rightarrow \Delta$
**and** $inR2$: $\Gamma \Rightarrow F,\ G,\ H,\ \Delta \Longrightarrow \Gamma \Rightarrow G,\ H,\ F,\ \Delta$
**and** $inL2$: $F,\ G,\ H,\ \Gamma \Rightarrow \Delta \Longrightarrow G,\ H,\ F,\ \Gamma \Rightarrow \Delta$ ⟨*proof*⟩
**lemmas** *SCp-swap-applies*[*elim*!,*intro*] = *inL1 inL2 inR1 inR2*

**lemma** *NotL-inv*: $Not\ F,\ \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow F,\Delta$
⟨*proof*⟩

**lemma** *AndL-inv*: $And\ F\ G,\ \Gamma \Rightarrow \Delta \Longrightarrow F,G,\Gamma \Rightarrow \Delta$
⟨*proof*⟩

**lemma** *OrL-inv*:

**assumes** *Or F G*, $\Gamma \Rightarrow \Delta$
**shows** *F*,$\Gamma \Rightarrow \Delta \wedge$ *G*,$\Gamma \Rightarrow \Delta$
⟨*proof*⟩

**lemma** *ImpL-inv*:
**assumes** *Imp F G*, $\Gamma \Rightarrow \Delta$
**shows** $\Gamma \Rightarrow$ *F*,$\Delta \wedge$ *G*,$\Gamma \Rightarrow \Delta$
⟨*proof*⟩

**lemma** *ImpR-inv*:
**assumes** $\Gamma \Rightarrow$ *Imp F G*,$\Delta$
**shows** *F*,$\Gamma \Rightarrow$ *G*,$\Delta$
⟨*proof*⟩

**lemma** *AndR-inv*:
**shows** $\Gamma \Rightarrow$ *And F G*, $\Delta \Longrightarrow \Gamma \Rightarrow F$, $\Delta \wedge \Gamma \Rightarrow G$, $\Delta$
⟨*proof*⟩

**lemma** *OrR-inv*: $\Gamma \Rightarrow$ *Or F G*, $\Delta \Longrightarrow \Gamma \Rightarrow F$,*G*,$\Delta$
⟨*proof*⟩

**lemma** *NotR-inv*: $\Gamma \Rightarrow$ *Not F*, $\Delta \Longrightarrow F$,$\Gamma \Rightarrow \Delta$
⟨*proof*⟩

**lemma** *weakenL*: $\Gamma \Rightarrow \Delta \Longrightarrow F$,$\Gamma \Rightarrow \Delta$
⟨*proof*⟩

**lemma** *weakenR*: $\Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow F$,$\Delta$
⟨*proof*⟩

**lemma** *weakenL-set*: $\Gamma \Rightarrow \Delta \Longrightarrow F + \Gamma \Rightarrow \Delta$
⟨*proof*⟩
**lemma** *weakenR-set*: $\Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow F + \Delta$
⟨*proof*⟩

**lemma** *extended-Ax*: $\Gamma \cap\# \Delta \neq \{\#\} \Longrightarrow \Gamma \Rightarrow \Delta$
⟨*proof*⟩

**lemma** *Bot-delR*: $\Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta - \{\#\bot\#\}$
⟨*proof*⟩
**corollary** *Bot-delR-simp*: $\Gamma \Rightarrow \bot$,$\Delta = \Gamma \Rightarrow \Delta$
⟨*proof*⟩

**end**
**theory** *SC-Cut*
**imports** *SC*
**begin**

### 2.1.1 Contraction

First, we need contraction:

**lemma** *contract*:
(*F*,*F*,Γ ⇒ Δ ⟶ *F*,Γ ⇒ Δ) ∧ (Γ ⇒ *F*,*F*,Δ ⟶ Γ ⇒ *F*,Δ)
⟨*proof*⟩
**corollary**
**shows** *contractL*: *F*, *F*, Γ ⇒ Δ ⟹ *F*, Γ ⇒ Δ
**and** *contractR*: Γ ⇒ *F*, *F*, Δ ⟹ Γ ⇒ *F*, Δ
⟨*proof*⟩

### 2.1.2 Cut

Which cut rule we show is up to us:

**lemma** *cut-cs-cf*:
**assumes** *context-sharing-Cut*: ⋀(*A*::'*a formula*) Γ Δ. Γ ⇒ *A*,Δ ⟹ *A*,Γ ⇒ Δ ⟹ Γ ⇒ Δ
**shows** *context-free-Cut*: Γ ⇒ (*A*::'*a formula*),Δ ⟹ *A*,Γ′ ⇒ Δ′ ⟹ Γ + Γ′ ⇒ Δ + Δ′
⟨*proof*⟩
**lemma** *cut-cf-cs*:
**assumes** *context-free-Cut*: ⋀(*A*::'*a formula*) Γ Γ′ Δ Δ′. Γ ⇒ *A*,Δ ⟹ *A*,Γ′ ⇒ Δ′ ⟹ Γ + Γ′ ⇒ Δ + Δ′
**shows** *context-sharing-Cut*: Γ ⇒ (*A*::'*a formula*),Δ ⟹ *A*,Γ ⇒ Δ ⟹ Γ ⇒ Δ
⟨*proof*⟩

According to Troelstra and Schwichtenberg [11], the sharing variant is the one we want to prove.

**lemma** *Cut-Atom-pre*: *Atom k*,Γ ⇒ Δ ⟹ Γ ⇒ *Atom k*,Δ ⟹ Γ ⇒ Δ
⟨*proof*⟩

We can show the admissibility of the cut rule by induction on the cut formula (or, if you will, as a procedure that splits the cut into smaller formulas that get cut). The only mildly complicated case is that of cutting in an *Atom k*. It is, contrary to the general case, only mildly complicated, since the cut formula can only appear principal in the axiom rules.

**theorem** *cut*: Γ ⇒ *F*,Δ ⟹ *F*,Γ ⇒ Δ ⟹ Γ ⇒ Δ
⟨*proof*⟩

**corollary** *cut-cf*: ⟦ Γ ⇒ *A*, Δ; *A*, Γ′ ⇒ Δ′⟧ ⟹ Γ + Γ′ ⇒ Δ + Δ′
⟨*proof*⟩

**lemma assumes** *cut*: ⋀ Γ′ Δ′ (*A*::'*a formula*). ⟦Γ′ ⇒ *A*, Δ′; *A*, Γ′ ⇒ Δ′⟧ ⟹ Γ′ ⇒ Δ′
**shows** *contraction-admissibility*: *F*,*F*,Γ ⇒ Δ ⟹ (*F*::'*a formula*),Γ ⇒ Δ
⟨*proof*⟩

**end**
**theory** *SC-Depth*
**imports** *SC*
**begin**

Many textbook arguments about SC use the depth of the derivation tree as basis for inductions. We had originally thought that this is mandatory for the proof of contraction, but found out it is not. It remains unclear to us whether there is any proof on SC that requires an argument using depth.

We keep our formalization of SC with depth for didactic reasons: we think that arguments about depth do not need much meta-explanation, but structural induction and rule induction usually need extra explanation for students unfamiliar with Isabelle/HOL. They are also a lot harder to execute. We dare the reader to write down (a few of) the cases for, e.g. *AndL-inv'*, by hand.

**inductive** *SCc* :: *'a formula multiset* ⇒ *'a formula multiset* ⇒ *nat* ⇒ *bool* (‹((- ⇒/ -) ↓ -)› [*53*,*53*] *53*) **where**
*BotL*: ⊥ ∈# Γ ⟹ Γ⇒Δ ↓ *Suc n* |
*Ax*: *Atom k* ∈# Γ ⟹ *Atom k* ∈# Δ ⟹ Γ⇒Δ ↓ *Suc n* |
*NotL*: Γ ⇒ *F*,Δ ↓ *n* ⟹ *Not F*, Γ ⇒ Δ ↓ *Suc n* |
*NotR*: *F*,Γ ⇒ Δ ↓ *n* ⟹ Γ ⇒ *Not F*, Δ ↓ *Suc n* |
*AndL*: *F*,*G*,Γ ⇒ Δ ↓ *n* ⟹ *And F G*, Γ ⇒ Δ ↓ *Suc n* |
*AndR*: ⟦ Γ ⇒ *F*,Δ ↓ *n*; Γ ⇒ *G*,Δ ↓ *n* ⟧ ⟹ Γ ⇒ *And F G*, Δ ↓ *Suc n* |
*OrL*: ⟦ *F*,Γ ⇒ Δ ↓ *n*; *G*,Γ ⇒ Δ ↓ *n* ⟧ ⟹ *Or F G*, Γ ⇒ Δ ↓ *Suc n* |
*OrR*: Γ ⇒ *F*,*G*,Δ ↓ *n* ⟹ Γ ⇒ *Or F G*, Δ ↓ *Suc n* |
*ImpL*: ⟦ Γ ⇒ *F*,Δ ↓ *n*; *G*,Γ ⇒ Δ ↓ *n* ⟧ ⟹ *Imp F G*, Γ ⇒ Δ ↓ *Suc n* |
*ImpR*: *F*,Γ ⇒ *G*,Δ ↓ *n* ⟹ Γ ⇒ *Imp F G*, Δ ↓ *Suc n*

**lemma**
**shows** *BotL-canonical'*[*intro*!]: ⊥,Γ⇒Δ ↓ *Suc n*
**and** *Ax-canonical'*[*intro*!]: *Atom k*,Γ ⇒ *Atom k*,Δ ↓ *Suc n*
⟨*proof*⟩

**lemma** *deeper*: Γ ⇒ Δ ↓ *n* ⟹ Γ ⇒ Δ ↓ *n* + *m*
⟨*proof*⟩
**lemma** *deeper-suc*: Γ ⇒ Δ ↓ *n* ⟹ Γ ⇒ Δ ↓ *Suc n*

**thm** *deeper*[*unfolded Suc-eq-plus1*[*symmetric*]]
⟨*proof*⟩

The equivalence is obvious.

**theorem** *SC-SCp-eq*:
**fixes** Γ Δ :: *'a formula multiset*
**shows** (∃ *n*. Γ ⇒ Δ ↓ *n*) ⟷ Γ ⇒ Δ (**is** *?c* ⟷ *?p*)
⟨*proof*⟩

**lemma** *no-0-SC*[*dest*!]: $\Gamma \Rightarrow \Delta \downarrow 0 \Longrightarrow \mathit{False}$
⟨*proof*⟩

**lemma** *inR1'*: $\Gamma \Rightarrow G, H, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow H, G, \Delta \downarrow n$ ⟨*proof*⟩
**lemma** *inL1'*: $G, H, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow H, G, \Gamma \Rightarrow \Delta \downarrow n$ ⟨*proof*⟩
**lemma** *inR2'*: $\Gamma \Rightarrow F, G, H, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow G, H, F, \Delta \downarrow n$ ⟨*proof*⟩
**lemma** *inL2'*: $F, G, H, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow G, H, F, \Gamma \Rightarrow \Delta \downarrow n$ ⟨*proof*⟩
**lemma** *inR3'*: $\Gamma \Rightarrow F, G, H, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow H, F, G, \Delta \downarrow n$ ⟨*proof*⟩
**lemma** *inR4'*: $\Gamma \Rightarrow F, G, H, I, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow H, I, F, G, \Delta \downarrow n$ ⟨*proof*⟩
**lemma** *inL3'*: $F, G, H, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow H, F, G, \Gamma \Rightarrow \Delta \downarrow n$ ⟨*proof*⟩
**lemma** *inL4'*: $F, G, H, I, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow H, I, F, G, \Gamma \Rightarrow \Delta \downarrow n$ ⟨*proof*⟩
**lemmas** *SC-swap-applies*[*intro*,*elim*!] = *inL1' inL2' inL3' inL4' inR1' inR2' inR3' inR4'*

**lemma** $\mathit{Atom}\ C \rightarrow \mathit{Atom}\ D \rightarrow \mathit{Atom}\ E,$
$\quad \mathit{Atom}\ k \rightarrow \mathit{Atom}\ C \wedge \mathit{Atom}\ D,$
$\quad \mathit{Atom}\ k, \{\#\}$
$\Rightarrow \{\#\ \mathit{Atom}\ E\ \#\} \downarrow \mathit{Suc}\ (\mathit{Suc}\ (\mathit{Suc}\ (\mathit{Suc}\ 0))))$
⟨*proof*⟩

**lemma** *Bot-delR'*: $\Gamma \Rightarrow \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow \Delta - \{\#\bot\#\} \downarrow n$
⟨*proof*⟩

**lemma** *NotL-inv'*: $\mathit{Not}\ F, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow F, \Delta \downarrow n$
⟨*proof*⟩

**lemma** *AndL-inv'*: $\mathit{And}\ F\ G, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow F, G, \Gamma \Rightarrow \Delta \downarrow n$
⟨*proof*⟩

**lemma** *OrL-inv'*:
**assumes** $\mathit{Or}\ F\ G, \Gamma \Rightarrow \Delta \downarrow n$
**shows** $F, \Gamma \Rightarrow \Delta \downarrow n \wedge G, \Gamma \Rightarrow \Delta \downarrow n$
⟨*proof*⟩

**lemma** *ImpL-inv'*:
**assumes** $\mathit{Imp}\ F\ G, \Gamma \Rightarrow \Delta \downarrow n$
**shows** $\Gamma \Rightarrow F, \Delta \downarrow n \wedge G, \Gamma \Rightarrow \Delta \downarrow n$
⟨*proof*⟩

**lemma** *ImpR-inv'*:
**assumes** $\Gamma \Rightarrow \mathit{Imp}\ F\ G, \Delta \downarrow n$
**shows** $F, \Gamma \Rightarrow G, \Delta \downarrow n$
⟨*proof*⟩

**lemma** *AndR-inv'*:

**shows** $\Gamma \Rightarrow And\ F\ G, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow F, \Delta \downarrow n \wedge \Gamma \Rightarrow G, \Delta \downarrow n$
⟨*proof*⟩

**lemma** *OrR-inv'*: $\Gamma \Rightarrow Or\ F\ G, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow F,G,\Delta \downarrow n$
⟨*proof*⟩

**lemma** *NotR-inv'*: $\Gamma \Rightarrow Not\ F, \Delta \downarrow n \Longrightarrow F,\Gamma \Rightarrow \Delta \downarrow n$
⟨*proof*⟩

**lemma** *weakenL'*: $\Gamma \Rightarrow \Delta \downarrow n \Longrightarrow F,\Gamma \Rightarrow \Delta \ \downarrow n$
⟨*proof*⟩

**lemma** *weakenR'*: $\Gamma \Rightarrow \Delta \ \downarrow n \Longrightarrow \Gamma \Rightarrow F,\Delta \ \downarrow n$
⟨*proof*⟩

**lemma** *contract'*:
$(F,F,\Gamma \Rightarrow \Delta \downarrow n \longrightarrow F,\Gamma \Rightarrow \Delta \downarrow n) \wedge (\Gamma \Rightarrow F,F,\Delta \downarrow n \longrightarrow \Gamma \Rightarrow F,\Delta \downarrow n)$
⟨*proof*⟩

**lemma** *Cut-Atom-depth*: $Atom\ k,\Gamma \Rightarrow \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow Atom\ k,\Delta \downarrow m \Longrightarrow \Gamma \Rightarrow \Delta \downarrow n + m$
⟨*proof*⟩
**primrec** *cut-bound* :: $nat \Rightarrow nat \Rightarrow 'a\ formula \Rightarrow nat$ **where**
*cut-bound n m* (*Atom* -) = *m* + *n* |
*cut-bound n m Bot* = *n* |
*cut-bound n m* (*Not F*) = *cut-bound m n F* |
*cut-bound n m* (*And F G*) = *cut-bound n* (*cut-bound n m F*) *G* |
*cut-bound n m* (*Or F G*) = *cut-bound* (*cut-bound n m F*) *m G* |
*cut-bound n m* (*Imp F G*) = *cut-bound* (*cut-bound m n F*) *m G*
**theorem** *cut-bound*: $\Gamma \Rightarrow F,\Delta \downarrow n \Longrightarrow F,\Gamma \Rightarrow \Delta \ \downarrow m \Longrightarrow \Gamma \Rightarrow \Delta \downarrow$ *cut-bound n m F*
⟨*proof*⟩

**context begin**
**private primrec** *cut-bound'* :: $nat \Rightarrow 'a\ formula \Rightarrow nat$ **where**
*cut-bound' n* (*Atom* -) = *2*n* |
*cut-bound' n Bot* = *n* |
*cut-bound' n* (*Not F*) = *cut-bound' n F* |
*cut-bound' n* (*And F G*) = *cut-bound'* (*cut-bound' n F*) *G* |
*cut-bound' n* (*Or F G*) = *cut-bound'* (*cut-bound' n F*) *G* |
*cut-bound' n* (*Imp F G*) = *cut-bound'* (*cut-bound' n F*) *G*

**private lemma** *cut-bound'-mono*: $a \leq b \Longrightarrow$ *cut-bound' a F* $\leq$ *cut-bound' b F*
⟨*proof*⟩ **lemma** *cut-bound-mono*: $a \leq c \Longrightarrow b \leq d \Longrightarrow$ *cut-bound a b F* $\leq$ *cut-bound c d F*
⟨*proof*⟩ **lemma** *cut-bound-max*: *max n* (*cut-bound'* (*max n m*) *F*) = *cut-bound'* (*max n m*) *F*
⟨*proof*⟩ **lemma** *cut-bound-max'*: *max n* (*cut-bound' n F*) = *cut-bound' n F*

⟨*proof*⟩ **lemma** *cut-bound-'*: *cut-bound n m F* ≤ *cut-bound'* (*max n m*) *F*
⟨*proof*⟩

**primrec** *depth* :: *'a formula* ⇒ *nat* **where**
*depth* (*Atom* -) = *0* |
*depth Bot* = *0* |
*depth* (*Not F*) = *Suc* (*depth F*) |
*depth* (*And F G*) = *Suc* (*max* (*depth F*) (*depth G*)) |
*depth* (*Or F G*) = *Suc* (*max* (*depth F*) (*depth G*)) |
*depth* (*Imp F G*) = *Suc* (*max* (*depth F*) (*depth G*))

**private primrec** *cbnd* **where**
*cbnd k 0* = *2*∗*k* |
*cbnd k* (*Suc n*) = *cbnd* (*cbnd k n*) *n*

**private lemma** *cbnd-grow*: (*k* :: *nat*) ≤ *cbnd k d*
⟨*proof*⟩ **lemma** *cbnd-mono*: **assumes** *b* ≤ *d* **shows** *cbnd* (*a*::*nat*) *b* ≤ *cbnd a d*
⟨*proof*⟩ **lemma** *cut-bound'-cbnd*: *cut-bound' n F* ≤ *cbnd n* (*depth F*)
⟨*proof*⟩

**value** *map* (*cbnd* (*0*::*int*)) [*0*,*1*,*2*,*3*,*4*]
**value** *map* (*cbnd* (*1*::*int*)) [*0*,*1*,*2*,*3*,*4*]
**value** *map* (*cbnd* (*2*::*int*)) [*0*,*1*,*2*,*3*,*4*]
**value** *map* (*cbnd* (*3*::*int*)) [*0*,*1*,*2*,*3*,*4*]
**value** [*nbe*] *map* (*int* ∘ (λ*n*. *n div 3*) ∘ *cut-bound 3 3* ∘ (λ*n*. ((λ*F*. *And F F*) ^^ *n*) (*Atom 0*))) [*0*,*1*,*2*,*3*,*4*,*5*,*6*,*7*]
**value** [*nbe*] *map* (*int* ∘ (λ*n*. *n div 3*) ∘ *cut-bound' 3* ∘ (λ*n*. ((λ*F*. *And F F*) ^^ *n*) (*Atom 0*))) [*0*,*1*,*2*,*3*,*4*]

**value** [*nbe*] *map* (*int* ∘ (λ*n*. *n div 3*) ∘ *cut-bound 3 3* ∘ (λ*n*. ((λ*F*. *Imp* (*Or F F*) (*And F F*)) ^^ *n*) (*Atom 0*))) [*0*,*1*,*2*]
**value** [*nbe*] *map* (*int* ∘ (λ*n*. *n div 3*) ∘ *cut-bound' 3* ∘ (λ*n*. ((λ*F*. *Imp* (*Or F F*) (*And F F*)) ^^ *n*) (*Atom 0*))) [*0*,*1*,*2*]

**value** [*nbe*] (λ*F*. *And* (*Or F F*) (*Or F F*)) ^^ *2*

**lemma** *n* + ((*n* + *m*) ∗ *2* ^ (*size F* − *Suc 0*) +
(*n* + (*n* + *m* + (*n* + *m*) ∗ *2* ^ (*size F* − *Suc 0*))) ∗ *2* ^ (*size G* − *Suc 0*))
≤ (*n* + (*m* :: *nat*)) ∗ *2* ^ (*size F* + *size G*)
⟨*proof*⟩

**lemma** *cut-bound* (*n* :: *nat*) *m F* ≤ (*n* + *m*) ∗ (*2* ^ (*size F* − *1*) + *1*)
⟨*proof*⟩ **lemma** *cbnd-comm*: *cbnd* (*l* ∗ *k*::*nat*) *n* = *l* ∗ *cbnd* (*k*::*nat*) *n*
⟨*proof*⟩ **lemma** *cbnd-closed*: *cbnd* (*k*::*nat*) *n* = *k* ∗ *2* ^ (*2* ^ *n*)
⟨*proof*⟩

**theorem** *cut'*: **assumes** Γ ⇒ *F*,Δ ↓ *n F*,Γ ⇒ Δ ↓ *n* **shows** Γ ⇒ Δ ↓ *n* ∗ *2* ^ (*2* ^ *depth F*)
⟨*proof*⟩

**end**

**end**

### 2.1.3 Mimicking the original

**theory** *SC-Gentzen*
**imports** *SC-Depth SC-Cut*
**begin**

This system attempts to mimic the original sequent calculus ("Reihen von Formeln, durch Kommata getrennt", translates roughly to sequences of formulas, separated by a comma) [4].

**inductive** *SCg* :: *'a formula list* ⇒ *'a formula list* ⇒ *bool* (**infix** ‹⇒› *30*) **where**
*Anfang*: $[\mathfrak{D}] \Rightarrow [\mathfrak{D}]$ |
*FalschA*: $[\bot] \Rightarrow []$ |
*VerduennungA*: $\Gamma \Rightarrow \Theta \Longrightarrow \mathfrak{D}\#\Gamma \Rightarrow \Theta$ |
*VerduennungS*: $\Gamma \Rightarrow \Theta \Longrightarrow \Gamma \Rightarrow \mathfrak{D}\#\Theta$ |
*ZusammenziehungA*: $\mathfrak{D}\#\mathfrak{D}\#\Gamma \Rightarrow \Theta \Longrightarrow \mathfrak{D}\#\Gamma \Rightarrow \Theta$ |
*ZusammenziehungS*: $\Gamma \Rightarrow \mathfrak{D}\#\mathfrak{D}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathfrak{D}\#\Theta$ |
*VertauschungA*: $\Delta@\mathfrak{D}\#\mathfrak{E}\#\Gamma \Rightarrow \Theta \Longrightarrow \Delta@\mathfrak{E}\#\mathfrak{D}\#\Gamma \Rightarrow \Theta$ |
*VertauschungS*: $\Gamma \Rightarrow \Theta@\mathfrak{E}\#\mathfrak{D}\#\Lambda \Longrightarrow \Gamma \Rightarrow \Theta@\mathfrak{D}\#\mathfrak{E}\#\Lambda$ |
*Schnitt*: $[\![\Gamma \Rightarrow \mathfrak{D}\#\Theta;\ \mathfrak{D}\#\Delta \Rightarrow \Lambda]\!] \Longrightarrow \Gamma@\Delta \Rightarrow \Theta@\Lambda$ |
*UES*: $[\![\Gamma \Rightarrow \mathfrak{A}\#\Theta;\ \Gamma \Rightarrow \mathfrak{B}\#\Theta]\!] \Longrightarrow \Gamma \Rightarrow \mathfrak{A}\wedge\mathfrak{B}\#\Theta$ |
*UEA1*: $\mathfrak{A}\#\Gamma \Rightarrow \Theta \Longrightarrow \mathfrak{A}\wedge\mathfrak{B}\#\Gamma \Rightarrow \Theta$ | *UEA2*: $\mathfrak{B}\#\Gamma \Rightarrow \Theta \Longrightarrow \mathfrak{A}\wedge\mathfrak{B}\#\Gamma \Rightarrow \Theta$ |
*OEA*: $[\![\mathfrak{A}\#\Gamma \Rightarrow \Theta;\ \mathfrak{B}\#\Gamma \Rightarrow \Theta]\!] \Longrightarrow \mathfrak{A}\vee\mathfrak{B}\#\Gamma \Rightarrow \Theta$ |
*OES1*: $\Gamma \Rightarrow \mathfrak{A}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathfrak{A}\vee\mathfrak{B}\#\Theta$ | *OES2*: $\Gamma \Rightarrow \mathfrak{B}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathfrak{A}\vee\mathfrak{B}\#\Theta$ |
*FES*: $\mathfrak{A}\#\Gamma \Rightarrow \mathfrak{B}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathfrak{A}\rightarrow\mathfrak{B}\#\Theta$ |
*FEA*: $[\![\Gamma \Rightarrow \mathfrak{A}\#\Theta;\ \mathfrak{B}\#\Delta \Rightarrow \Lambda]\!] \Longrightarrow \mathfrak{A}\rightarrow\mathfrak{B}\#\Gamma@\Delta \Rightarrow \Theta@\Lambda$ |
*NES*: $\mathfrak{A}\#\Gamma \Rightarrow \Theta \Longrightarrow \Gamma \Rightarrow \neg\mathfrak{A}\#\Theta$ |
*NEA*: $\Gamma \Rightarrow \mathfrak{A}\#\Theta \Longrightarrow \neg\mathfrak{A}\#\Gamma \Rightarrow \Theta$

Nota bene: E here stands for "Einführung", which is introduction and not elemination.

The rule for $\bot$ is not part of the original calculus. Its addition is necessary to show equivalence to our *SCp*.

Note that we purposefully did not recreate the fact that Gentzen sometimes puts his principal formulas on end and sometimes on the beginning of the list.

**lemma** *AnfangTauschA*: $\mathfrak{D}\#\Delta@\Gamma \Rightarrow \Theta \Longrightarrow \Delta@\mathfrak{D}\#\Gamma \Rightarrow \Theta$
⟨*proof*⟩
**lemma** *AnfangTauschS*: $\Gamma \Rightarrow \mathfrak{D}\#\Delta@\Theta \Longrightarrow \Gamma \Rightarrow \Delta@\mathfrak{D}\#\Theta$
⟨*proof*⟩
**lemma** *MittenTauschA*: $\Delta@\mathfrak{D}\#\Gamma \Rightarrow \Theta \Longrightarrow \mathfrak{D}\#\Delta@\Gamma \Rightarrow \Theta$
⟨*proof*⟩
**lemma** *MittenTauschS*: $\Gamma \Rightarrow \Delta@\mathfrak{D}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathfrak{D}\#\Delta@\Theta$

⟨*proof*⟩

**lemma** *BotLe*: $\bot \in set\ \Gamma \Longrightarrow \Gamma \Rightarrow \Delta$
⟨*proof*⟩

**lemma** *Axe*: $A \in set\ \Gamma \Longrightarrow A \in set\ \Delta \Longrightarrow \Gamma \Rightarrow \Delta$
⟨*proof*⟩

**lemma** *VerduennungListeA*: $\Gamma \Rightarrow \Theta \Longrightarrow \Gamma @ \Gamma \Rightarrow \Theta$
⟨*proof*⟩
**lemma** *VerduennungListeS*: $\Gamma \Rightarrow \Theta \Longrightarrow \Gamma \Rightarrow \Theta @ \Theta$
⟨*proof*⟩

**lemma** *ZusammenziehungListeA*: $\Gamma @ \Gamma \Rightarrow \Theta \Longrightarrow \Gamma \Rightarrow \Theta$
⟨*proof*⟩
**lemma** *ZusammenziehungListeS*: $\Gamma \Rightarrow \Theta @ \Theta \Longrightarrow \Gamma \Rightarrow \Theta$
⟨*proof*⟩

**theorem** *gentzen-sc-eq*: *mset* $\Gamma \Rightarrow$ *mset* $\Delta \longleftrightarrow \Gamma \Rightarrow \Delta$ ⟨*proof*⟩

**end**

### 2.1.4 Soundness, Completeness

**theory** *SC-Sema*
**imports** *SC Sema*
**begin**

**definition** *sequent-semantics* :: *'a valuation* ⇒ *'a formula multiset* ⇒ *'a formula multiset* ⇒ *bool* (‹(- ⊨ (- ⇒/ -))› [*53*, *53*,*53*] *53*) **where**
$\mathcal{A} \models \Gamma \Rightarrow \Delta \equiv (\forall \gamma \in\# \Gamma.\ \mathcal{A} \models \gamma) \longrightarrow (\exists \delta \in\# \Delta.\ \mathcal{A} \models \delta)$
**abbreviation** *sequent-valid* :: *'a formula multiset* ⇒ *'a formula multiset* ⇒ *bool* (‹(⊨ (- ⇒/ -))› [*53*,*53*] *53*) **where**
$\models \Gamma \Rightarrow \Delta \equiv \forall A.\ A \models \Gamma \Rightarrow \Delta$
**abbreviation** *sequent-nonvalid* :: *'a valuation* ⇒ *'a formula multiset* ⇒ *'a formula multiset* ⇒ *bool* (‹(- ¬⊨ (- ⇒/ -))› [*53*, *53*,*53*] *53*) **where**
$\mathcal{A}\ \neg\models \Gamma \Rightarrow \Delta \equiv \neg \mathcal{A} \models \Gamma \Rightarrow \Delta$

**lemma** *sequent-intuitonistic-semantics*: $\models \Gamma \Rightarrow \{\#\delta\#\} \longleftrightarrow$ *set-mset* $\Gamma \Vdash \delta$
⟨*proof*⟩

**lemma** *SC-soundness*: $\Gamma \Rightarrow \Delta \Longrightarrow \models \Gamma \Rightarrow \Delta$
⟨*proof*⟩

**definition** *sequent-cost* $\Gamma\ \Delta$ = *Suc* (*sum-list* (*sorted-list-of-multiset* (*image-mset size* ($\Gamma$ + $\Delta$))))

**function**(*sequential*)

*sc* :: *'a formula list* ⇒ *'a list* ⇒ *'a formula list* ⇒ *'a list* ⇒ (*'a list* × *'a list*) *set*
**where**
*sc* (⊥ # Γ) *A* Δ *B* = {} |
*sc* [] *A* [] *B* = (*if set A* ∩ *set B* = {} *then* {(*remdups A*,*remdups B*)} *else* {}) |
*sc* (*Atom k* # Γ) *A* Δ *B* = *sc* Γ (*k*#*A*) Δ *B* |
*sc* (*Not F* # Γ) *A* Δ *B* = *sc* Γ *A* (*F*#Δ) *B* |
*sc* (*And F G* # Γ) *A* Δ *B* = *sc* (*F*#*G*#Γ) *A* Δ *B* |
*sc* (*Or F G* # Γ) *A* Δ *B* = *sc* (*F*#Γ) *A* Δ *B* ∪ *sc* (*G*#Γ) *A* Δ *B* |
*sc* (*Imp F G* # Γ) *A* Δ *B* = *sc* Γ *A* (*F*#Δ) *B* ∪ *sc* (*G*#Γ) *A* Δ *B* |
*sc* Γ *A* (⊥#Δ) *B* = *sc* Γ *A* Δ *B* |
*sc* Γ *A* (*Atom k* # Δ) *B* = *sc* Γ *A* Δ (*k*#*B*) |
*sc* Γ *A* (*Not F* # Δ) *B* = *sc* (*F*#Γ) *A* Δ *B* |
*sc* Γ *A* (*And F G* # Δ) *B* = *sc* Γ *A* (*F*#Δ) *B* ∪ *sc* Γ *A* (*G*#Δ) *B* |
*sc* Γ *A* (*Or F G* # Δ) *B* = *sc* Γ *A* (*F*#*G*#Δ) *B* |
*sc* Γ *A* (*Imp F G* # Δ) *B* = *sc* (*F*#Γ) *A* (*G*#Δ) *B*
⟨*proof*⟩

**definition** *list-sequent-cost* Γ Δ = *2*∗*sum-list* (*map size* (Γ@Δ)) + *length* (Γ@Δ)
**termination** *sc* ⟨*proof*⟩

**lemma** *sc* [] [] [((*Atom 0* → *Atom 1*) → *Atom 0*) → *Atom 1*] [] = {([*0*], [*1* :: *nat*])}

⟨*proof*⟩

**lemma** *sc-sim*:
**fixes** Γ Δ :: *'a formula list* **and** *G D* :: *'a list*
**assumes** *sc* Γ *A* Δ *B* = {}
**shows** *image-mset Atom* (*mset A*) + *mset* Γ ⇒ *image-mset Atom* (*mset B*) + *mset* Δ
⟨*proof*⟩

**lemma** *scc-ce-distinct*:
(*C*,*E*) ∈ *sc* Γ *G* Δ *D* ⟹ *set C* ∩ *set E* = {}
⟨*proof*⟩

Completeness set aside, this is an interesting fact on the side: Sequent Calculus can provide counterexamples.

**theorem** *SC-counterexample*:
(*C*,*D*) ∈ *sc* Γ *A* Δ *B* ⟹
(λ*a*. *a* ∈ *set C*) ¬⊨ *image-mset Atom* (*mset A*) + *mset* Γ ⇒ *image-mset Atom* (*mset B*) + *mset* Δ
⟨*proof*⟩

**corollary** *SC-counterexample'*:
**assumes** (*C*,*D*) ∈ *sc* Γ [] Δ []
**shows** (λ*k*. *k* ∈ *set C*) ¬⊨ *mset* Γ ⇒ *mset* Δ
⟨*proof*⟩

**theorem** *SC-sound-complete*: $\Gamma \Rightarrow \Delta \longleftrightarrow \models \Gamma \Rightarrow \Delta$
⟨*proof*⟩

**theorem** $\models \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta$
⟨*proof*⟩

**end**
**theory** *SC-Depth-Limit*
**imports** *SC-Sema SC-Depth*
**begin**

**lemma** *SC-completeness*: $\models \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta \downarrow$ *sequent-cost* $\Gamma\ \Delta$
⟨*proof*⟩

Making this proof of completeness go through should be possible, but finding the right way to split the cases could get verbose. The variant with the search procedure is a lot more elegant.

**lemma** *sc-sim-depth*:
**assumes** *sc* $\Gamma\ A\ \Delta\ B = \{\}$
**shows** *image-mset Atom* (*mset A*) + *mset* $\Gamma \Rightarrow$ *image-mset Atom* (*mset B*) + *mset* $\Delta \downarrow$ *sum-list* (*map size* ($\Gamma$@$\Delta$)) + (*if set A* $\cap$ *set B* = $\{\}$ *then 0 else 1*)
⟨*proof*⟩

**corollary** *sc-depth-complete*:
**assumes** *s*: $\models \Gamma \Rightarrow \Delta$
**shows** $\Gamma \Rightarrow \Delta \downarrow$ *sum-mset* (*image-mset size* ($\Gamma$+$\Delta$))
⟨*proof*⟩

**end**
**theory** *SC-Compl-Consistency*
**imports** *Consistency SC-Cut SC-Sema*
**begin**

**context begin**
**private lemma** *reasonable*:
$\forall \Gamma'.\ F \quad \triangleright$ *set-mset* $\Gamma$ = *set-mset* $\Gamma' \longrightarrow P\ \Gamma' \Longrightarrow P\ (F, \Gamma)$
$\forall \Gamma'.\ F \triangleright G \triangleright$ *set-mset* $\Gamma$ = *set-mset* $\Gamma' \longrightarrow P\ \Gamma' \Longrightarrow P\ (F, G, \Gamma)$ ⟨*proof*⟩

**lemma** *SC-consistent*: *pcp* $\{$*set-mset* $\Gamma|\ \Gamma.\ \neg(\Gamma \Rightarrow \{\#\})\}$
⟨*proof*⟩

**end**

**lemma**

**fixes** $\Gamma\ \Delta$ :: *'a* :: *countable formula multiset*
**shows** $\models \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta$
⟨*proof*⟩

**end**

## 2.2 Natural Deduction

**theory** *ND*
**imports** *Formulas*
**begin**

**inductive** *ND* :: *'a formula set* ⇒ *'a formula* ⇒ *bool* (**infix** ‹⊢› *30*) **where**
*Ax*: $F \in \Gamma \Longrightarrow \Gamma \vdash F$ |
*NotE*: $[\![\Gamma \vdash Not\ F; \Gamma \vdash F\ ]\!] \Longrightarrow \Gamma \vdash \bot$ |
*NotI*: $F \triangleright \Gamma \vdash \bot \Longrightarrow \Gamma \vdash Not\ F$ |
*CC*: $Not\ F \triangleright \Gamma \vdash \bot \Longrightarrow \Gamma \vdash F$ |
*AndE1*: $\Gamma \vdash And\ F\ G \Longrightarrow \Gamma \vdash F$ |
*AndE2*: $\Gamma \vdash And\ F\ G \Longrightarrow \Gamma \vdash G$ |
*AndI*: $[\![\ \Gamma \vdash F; \Gamma \vdash G\ ]\!] \Longrightarrow \Gamma \vdash And\ F\ G$ |
*OrI1*: $\Gamma \vdash F \Longrightarrow \Gamma \vdash Or\ F\ G$ |
*OrI2*: $\Gamma \vdash G \Longrightarrow \Gamma \vdash Or\ F\ G$ |
*OrE*: $[\![\ \Gamma \vdash Or\ F\ G; F \triangleright \Gamma \vdash H; G \triangleright \Gamma \vdash H\ ]\!] \Longrightarrow \Gamma \vdash H$ |
*ImpI*: $F \triangleright \Gamma \vdash G \Longrightarrow \Gamma \vdash Imp\ F\ G$ |
*ImpE*: $[\![\ \Gamma \vdash Imp\ F\ G; \Gamma \vdash F\ ]\!] \Longrightarrow \Gamma \vdash G$

**lemma** *Weaken*: $[\![\ \Gamma \vdash F; \Gamma \subseteq \Gamma'\ ]\!] \Longrightarrow \Gamma' \vdash F$
⟨*proof*⟩

**lemma** *BotE* : $\Gamma \vdash \bot \Longrightarrow \Gamma \vdash F$
⟨*proof*⟩

**lemma** *Not2E*: $Not(Not\ F) \triangleright \Gamma \vdash F$
⟨*proof*⟩

**lemma** *Not2I*: $F \triangleright \Gamma \vdash Not(Not\ F)$
⟨*proof*⟩

**lemma** *Not2IE*: $F \triangleright \Gamma \vdash G \Longrightarrow Not\ (Not\ F) \triangleright \Gamma \vdash G$
⟨*proof*⟩

**lemma** *NDtrans*: $\Gamma \vdash F \Longrightarrow F \triangleright \Gamma \vdash G \Longrightarrow \Gamma \vdash G$
⟨*proof*⟩

**lemma** *AndL-sim*: $F \triangleright G \triangleright \Gamma \vdash H \Longrightarrow And\ F\ G \triangleright \Gamma \vdash H$
⟨*proof*⟩

**lemma** *NotSwap*: *Not F*▷ Γ ⊢ *G* ⟹ *Not G*▷ Γ ⊢ *F*
⟨*proof*⟩
**lemma** *AndR-sim*: ⟦ *Not F*▷ Γ ⊢ *H*; *Not G*▷ Γ ⊢ *H* ⟧ ⟹ *Not*(*And F G*)▷ Γ ⊢ *H*
⟨*proof*⟩

**lemma** *OrL-sim*: ⟦ *F*▷ Γ ⊢ *H*; *G*▷Γ ⊢ *H* ⟧ ⟹ *F* ∨ *G*▷ Γ ⊢ *H*
⟨*proof*⟩

**lemma** *OrR-sim*: ⟦ ¬ *F*▷ ¬ *G*▷ Γ ⊢ ⊥⟧ ⟹ ¬ (*G* ∨ *F*)▷ Γ ⊢ ⊥
⟨*proof*⟩

**lemma** *ImpL-sim*: ⟦ ¬ *F*▷ Γ ⊢ ⊥; *G*▷ Γ ⊢ ⊥⟧ ⟹ *F* → *G*▷ Γ ⊢ ⊥
⟨*proof*⟩

**lemma** *ImpR-sim*: ⟦ ¬ *G*▷ *F*▷ Γ ⊢ ⊥⟧ ⟹ ¬ (*F* → *G*)▷ Γ ⊢ ⊥
⟨*proof*⟩

**lemma** *ND-lem*: {} ⊢ *Not F* ∨ *F*
⟨*proof*⟩

**lemma** *ND-caseDistinction*: ⟦ *F*▷Γ ⊢ *H*; *Not F*▷Γ ⊢ *H* ⟧ ⟹ Γ ⊢ *H*
⟨*proof*⟩

**lemma** ⟦¬ *F*▷ Γ ⊢ *H*; *G*▷ Γ ⊢ *H*⟧ ⟹ *F* → *G*▷ Γ ⊢ *H*
⟨*proof*⟩

**lemma** *ND-deMorganAnd*: {¬ (*F* ∧ *G*)} ⊢ ¬ *F* ∨ ¬ *G*
⟨*proof*⟩

**lemma** *ND-deMorganOr*: {¬ (*F* ∨ *G*)} ⊢ ¬ *F* ∧ ¬ *G*
⟨*proof*⟩

**lemma** *sim-sim*: *F*▷ Γ ⊢ *H* ⟹ *G*▷Γ ⊢ *F* ⟹ *G*▷ Γ ⊢ *H*
⟨*proof*⟩
**thm** *sim-sim*[**where** Γ={}, *rotated*, *no-vars*]

**lemma** *Top-provable*[*simp*,*intro*!]: Γ ⊢ ⊤ ⟨*proof*⟩

**lemma** *NotBot-provable*[*simp*,*intro*!]: Γ ⊢ ¬ ⊥ ⟨*proof*⟩

**lemma** *Top-useless*: Γ ⊢ *F* ⟹ Γ − {⊤} ⊢ *F*
⟨*proof*⟩

**lemma** *AssmBigAnd*: *set G* ⊢ *F* ⟷ {} ⊢ (⋀*G* → *F*)
⟨*proof*⟩

**end**
**theory** *ND-Sound*

**imports** *ND Sema*
**begin**

**lemma** *BigAndImp*: $A \models (\bigwedge P \rightarrow G) \longleftrightarrow ((\forall F \in set\ P.\ A \models F) \longrightarrow A \models G)$
⟨*proof*⟩

**lemma** *ND-sound*: $\Gamma \vdash F \Longrightarrow \Gamma \Vdash F$
⟨*proof*⟩

**end**
**theory** *ND-Compl-Truthtable*
**imports** *ND-Sound*
**begin**

This proof is inspired by Huth and Ryan [7].

**definition** *turn-true* $\mathcal{A}\ F \equiv$ *if* $\mathcal{A} \models F$ *then* $F$ *else* $(Not\ F)$
**lemma** *lemma0*[*simp*,*intro*!]: $\mathcal{A} \models$ *turn-true* $\mathcal{A}\ F$ ⟨*proof*⟩

**lemma** *turn-true-simps*[*simp*]:
$\mathcal{A} \models F \Longrightarrow$ *turn-true* $\mathcal{A}\ F = F$
$\neg\ \mathcal{A} \models F \Longrightarrow$ *turn-true* $\mathcal{A}\ F = \neg\ F$
⟨*proof*⟩

**definition** *line-assm* :: *'a valuation* ⇒ *'a set* ⇒ *'a formula set* **where**
*line-assm* $\mathcal{A} \equiv ($‘$)\ (\lambda k.$ *turn-true* $\mathcal{A}\ (Atom\ k))$
**definition** *line-suitable* :: *'a set* ⇒ *'a formula* ⇒ *bool* **where**
*line-suitable* $Z\ F \equiv (atoms\ F \subseteq Z)$
**lemma** *line-suitable-junctors*[*simp*]:
*line-suitable* $\mathcal{A}\ (Not\ F) =$ *line-suitable* $\mathcal{A}\ F$
*line-suitable* $\mathcal{A}\ (And\ F\ G) = ($*line-suitable* $\mathcal{A}\ F \wedge$ *line-suitable* $\mathcal{A}\ G)$
*line-suitable* $\mathcal{A}\ (Or\ F\ G) = ($*line-suitable* $\mathcal{A}\ F \wedge$ *line-suitable* $\mathcal{A}\ G)$
*line-suitable* $\mathcal{A}\ (Imp\ F\ G) = ($*line-suitable* $\mathcal{A}\ F \wedge$ *line-suitable* $\mathcal{A}\ G)$
⟨*proof*⟩

**lemma** *line-assm-Cons*[*simp*]: *line-assm* $\mathcal{A}\ (k \triangleright ks) =$ (*if* $\mathcal{A}\ k$ *then Atom k else Not* (*Atom k*)) ▷ *line-assm* $\mathcal{A}\ ks$
⟨*proof*⟩

**lemma** *NotD*: $\Gamma \vdash \neg\ F \Longrightarrow F \triangleright \Gamma \vdash \bot$ ⟨*proof*⟩

**lemma** *truthline-ND-proof*:
**fixes** $F$ :: *'a formula*
**assumes** *line-suitable* $Z\ F$
**shows** *line-assm* $\mathcal{A}\ Z \vdash$ *turn-true* $\mathcal{A}\ F$
⟨*proof*⟩
**thm** *NotD*[*THEN BotE*]

**lemma** *deconstruct-assm-set*:
  **assumes** *IH*: $\bigwedge \mathcal{A}$. *line-assm* $\mathcal{A}$ $(k \triangleright Z) \vdash F$
  **shows** $\bigwedge \mathcal{A}$. *line-assm* $\mathcal{A}$ $Z \vdash F$
⟨*proof*⟩

**theorem** *ND-complete*:
  **assumes** *taut*: $\models F$
  **shows** $\{\} \vdash F$
⟨*proof*⟩

**corollary** *ND-sound-complete*: $\{\} \vdash F \longleftrightarrow \models F$
  ⟨*proof*⟩

**end**
**theory** *ND-Compl-Truthtable-Compact*
**imports** *ND-Compl-Truthtable Compactness*
**begin**

**theorem**
  **fixes** $\Gamma$ :: *'a* :: *countable formula set*
  **shows** $\Gamma \models F \Longrightarrow \Gamma \vdash F$
⟨*proof*⟩

**end**

## 2.3 Hilbert Calculus

**theory** *HC*
**imports** *Formulas*
**begin**

We can define Hilbert Calculus as the modus ponens closure over a set of axioms:

**inductive** *HC* :: *'a formula set* ⇒ *'a formula* ⇒ *bool* (**infix** ‹$\vdash_H$› *30*) **for** $\Gamma$ :: *'a formula set* **where**
*Ax*: $F \in \Gamma \Longrightarrow \Gamma \vdash_H F$ |
*MP*: $\Gamma \vdash_H F \Longrightarrow \Gamma \vdash_H F \rightarrow G \Longrightarrow \Gamma \vdash_H G$

.

**context begin**

The problem with that is defining the axioms. Normally, we just write that $F \rightarrow G \rightarrow F$ is an axiom, and mean that anything can be substituted for $F$ and $G$. Now, we can't just write down a set $\{F \rightarrow (G \rightarrow F), \backslash dots$. Instead, defining it as an inductive set with no induction is a good idea.

**inductive-set** *AX0* **where**
$F \rightarrow (G \rightarrow F) \in AX0$ |

$(F \to (G \to H)) \to ((F \to G) \to (F \to H)) \in AX0$
**inductive-set** *AX10* **where**
$F \in AX0 \Longrightarrow F \in AX10$ |
$F \to (F \vee G) \in AX10$ |
$G \to (F \vee G) \in AX10$ |
$(F \to H) \to ((G \to H) \to ((F \vee G) \to H)) \in AX10$ |
$(F \wedge G) \to F \in AX10$ |
$(F \wedge G) \to G \in AX10$ |
$F \to (G \to (F \wedge G)) \in AX10$ |
$(F \to \bot) \to \neg F \in AX10$ |
$\neg F \to (F \to \bot) \in AX10$ |
$(\neg F \to \bot) \to F \in AX10$
**lemmas** *HC-intros*[*intro*!] =
*AX0.intros*[*THEN HC.intros*(1)]
*AX0.intros*[*THEN AX10.intros*(1), *THEN HC.intros*(1)]
*AX10.intros*(2−)[*THEN HC.intros*(1)]

The first four axioms, as originally formulated by Hilbert [6].

**inductive-set** *AXH* **where**
$(F \to (G \to F)) \in AXH$ |
$(F \to (F \to G)) \to (F \to G) \in AXH$ |
$(F \to (G \to H)) \to (G \to (F \to H)) \in AXH$ |
$(G \to H) \to ((F \to G) \to (F \to H)) \in AXH$

**lemma** *HC-mono*: $S \vdash_H F \Longrightarrow S \subseteq T \Longrightarrow T \vdash_H F$
⟨*proof*⟩
**lemma** *AX010*: $AX0 \subseteq AX10$
⟨*proof*⟩
**lemma** *AX100*[*simp*]: $AX0 \cup AX10 = AX10$ ⟨*proof*⟩

Hilbert's first four axioms and *AX0* are syntactically quite different. Derivability does not change:

**lemma** *hilbert-folgeaxiome-as-strong-as-AX0*: $AX0 \cup \Gamma \vdash_H F \longleftrightarrow AXH \cup \Gamma \vdash_H F$
⟨*proof*⟩

**lemma** $AX0 \vdash_H F \to F$ ⟨*proof*⟩

**lemma** *imp-self*: $AX0 \vdash_H F \to F$ ⟨*proof*⟩

**theorem** *Deduction-theorem*: $AX0 \cup insert\ F\ \Gamma \vdash_H G \Longrightarrow AX0 \cup \Gamma \vdash_H F \to G$
⟨*proof*⟩

**end**

**end**
**theory** *HC-Compl-Consistency*

**imports** *Consistency HC*
**begin**

**context begin**
**private lemma** *dt*: $F \triangleright \Gamma \cup AX10 \vdash_H G \Longrightarrow \Gamma \cup AX10 \vdash_H F \rightarrow G$
⟨*proof*⟩
**lemma** *sim*: $\Gamma \cup AX10 \vdash_H F \Longrightarrow F \triangleright \Gamma \cup AX10 \vdash_H G \Longrightarrow \Gamma \cup AX10 \vdash_H G$
⟨*proof*⟩
**lemma** *sim-conj*: $F \triangleright G \triangleright \Gamma \cup AX10 \vdash_H H \Longrightarrow \Gamma \cup AX10 \vdash_H F \Longrightarrow \Gamma \cup AX10 \vdash_H G \Longrightarrow \Gamma \cup AX10 \vdash_H H$
⟨*proof*⟩
**lemma** *sim-disj*: $[\![F \triangleright \Gamma \cup AX10 \vdash_H H;\ G \triangleright \Gamma \cup AX10 \vdash_H H;\ \Gamma \cup AX10 \vdash_H F \vee G]\!] \Longrightarrow \Gamma \cup AX10 \vdash_H H$
⟨*proof*⟩ **lemma** *someax*: $\Gamma \cup AX10 \vdash_H F \rightarrow \neg F \rightarrow \bot$
⟨*proof*⟩

**lemma** *lem*: $\Gamma \cup AX10 \vdash_H \neg F \vee F$
⟨*proof*⟩

**lemma** *exchg*: $\Gamma \cup AX10 \vdash_H F \vee G \Longrightarrow \Gamma \cup AX10 \vdash_H G \vee F$
⟨*proof*⟩

**lemma** *lem2*: $\Gamma \cup AX10 \vdash_H F \vee \neg F$ ⟨*proof*⟩

**lemma** *imp-sim*: $\Gamma \cup AX10 \vdash_H F \rightarrow G \Longrightarrow \Gamma \cup AX10 \vdash_H \neg F \vee G$
⟨*proof*⟩

**lemma** *inpcp*: $\Gamma \cup AX10 \vdash_H \bot \Longrightarrow \Gamma \cup AX10 \vdash_H F$
⟨*proof*⟩

**lemma** *HC-case-distinction*: $\Gamma \cup AX10 \vdash_H F \rightarrow G \Longrightarrow \Gamma \cup AX10 \vdash_H \neg F \rightarrow G \Longrightarrow \Gamma \cup AX10 \vdash_H G$
⟨*proof*⟩

**lemma** *nand-sim*: $\Gamma \cup AX10 \vdash_H \neg (F \wedge G) \Longrightarrow \Gamma \cup AX10 \vdash_H \neg F \vee \neg G$
⟨*proof*⟩

**lemma** *HC-contrapos-nn*:
$[\![\Gamma \cup AX10 \vdash_H \neg F; \Gamma \cup AX10 \vdash_H G \rightarrow F]\!] \Longrightarrow \Gamma \cup AX10 \vdash_H \neg G$
⟨*proof*⟩

**lemma** *nor-sim*:
**assumes** $\Gamma \cup AX10 \vdash_H \neg (F \vee G)$
**shows** $\Gamma \cup AX10 \vdash_H \neg F\ \ \Gamma \cup AX10 \vdash_H \neg G$
⟨*proof*⟩

**lemma** *HC-contrapos-np*:

$[\![\Gamma \cup AX10 \vdash_H \neg F; \Gamma \cup AX10 \vdash_H \neg G \to F]\!] \Longrightarrow \Gamma \cup AX10 \vdash_H G$
⟨*proof*⟩

**lemma** *not-imp*: $\Gamma \cup AX10 \vdash_H \neg F \to F \to G$
⟨*proof*⟩

**lemma** *HC-consistent*: *pcp* $\{\Gamma |\ \Gamma.\ \neg(\Gamma \cup AX10 \vdash_H \bot)\}$
⟨*proof*⟩

**end**

**corollary** *HC-complete*:
**fixes** $F$ :: $'a$ :: *countable formula*
**shows** $\models F \Longrightarrow AX10 \vdash_H F$
⟨*proof*⟩

**end**

## 2.4 Resolution

**theory** *Resolution*
**imports** *CNF HOL−Library.While-Combinator*
**begin**

Resolution is different from the other proof systems: its derivations do not represent proofs in the way the other systems do. Rather, they represent invariant additions (under satisfiability) to set of clauses.

**inductive** *Resolution* :: $'a$ *literal set set* ⇒ $'a$ *literal set* ⇒ *bool* (‹- ⊢ -› [*30*] *28*)
**where**
*Ass*: $C \in S \Longrightarrow S \vdash C$ |
*R*: $S \vdash C \Longrightarrow S \vdash D \Longrightarrow k^+ \in C \Longrightarrow k^{-1} \in D \Longrightarrow S \vdash (C - \{k^+\}) \cup (D - \{k^{-1}\})$

The problematic part of this formulation is that we can't talk about a "Resolution Refutation" in an inductive manner. In the places where Gallier's proofs [3] do that, we have to work around that.

**lemma** *Resolution-weaken*: $S \vdash D \Longrightarrow T \cup S \vdash D$
⟨*proof*⟩

**lemma** *Resolution-unnecessary*:
**assumes** *sd*: $\forall C \in T.\ S \vdash C$
**shows** $T \cup S \vdash D \longleftrightarrow S \vdash D$ (**is** *?l* ⟷ *?r*)
⟨*proof*⟩

**lemma** *Resolution-taint-assumptions*: $S \cup T \vdash C \Longrightarrow \exists R \subseteq D.\ ((\cup)\ D\ `\ S) \cup T \vdash R \cup C$

⟨*proof*⟩

Resolution is "strange": Given a set of clauses that is presumed to be satisfiable, it derives new clauses that can be added while preserving the satisfiability of the set of clauses. However, not every clause that could be added while keeping satisfiability can actually be added by resolution. Especially, the above lemma *Resolution-taint-assumptions* gives us the derivability of a clause $R \cup C$, where $R \subseteq D$. What we might actually want is the derivability of $D \cup C$. Any model that satisfies $R \cup C$ obviously satisfies $D \cup C$ (since they are disjunctions), but Resolution only allows to derive the former.

Nevertheless, *Resolution-taint-assumptions*, can still be a quite useful lemma: picking $D$ to be a singleton set only leaves two possibilities for $R$.

**lemma** *Resolution-useless-infinite*:
**assumes** *R*: $S \vdash R$
**assumes** *finite R*
**shows** $\exists S' \subseteq S.$ *Ball* $S'$ *finite* $\wedge$ *finite* $S' \wedge (S' \vdash R)$
⟨*proof*⟩

Now we define and verify a toy resolution prover. Function *res* computes the set of resolvents of a clause set:

**context begin**

**definition** *res* :: *'a clause set* ⇒ *'a clause set* **where**
*res S* =
($\bigcup$ *C1* ∈ *S*. $\bigcup$ *C2* ∈ *S*. $\bigcup$ *L1* ∈ *C1*. $\bigcup$ *L2* ∈ *C2*.
(*case* (*L1*,*L2*) *of* (*Pos i*,*Neg j*) ⇒ *if i*=*j then* {(*C1* − {*Pos i*}) ∪ (*C2* − {*Neg j*})} *else* {}
| - ⇒ {}))

**private definition** *ex1* ≡ {{*Neg* (*0*::*int*)}, {*Pos 0*, *Pos 1*, *Neg 2*}, {*Pos 0*, *Pos 1*, *Pos 2*}, {*Pos 0*, *Neg 1*}}
**value** *res ex1*

**definition** *Rwhile* :: *'a clause set* ⇒ *'a clause set option* **where**
*Rwhile* = *while-option* (λ*S*. □ ∉ *S* ∧ ¬ *res S* ⊆ *S*) (λ*S*. *res S* ∪ *S*)

**value** [*code*] *Rwhile ex1*
**lemma** □ ∈ *the* (*Rwhile ex1*) ⟨*proof*⟩

**lemma** *Rwhile-sound*: **assumes** *Rwhile S* = *Some S'*
**shows** ∀ *C* ∈ *S'*. *Resolution S C*
⟨*proof*⟩

**definition** *all-clauses S* = {*s*. *s* ⊆ {*Pos k*|*k*. *k* ∈ *atoms-of-cnf S*} ∪ {*Neg k*|*k*. *k* ∈ *atoms-of-cnf S*}}

**lemma** *s-sub-all-clauses*: $S \subseteq$ *all-clauses* $S$
⟨*proof*⟩
**lemma** *atoms-res*: *atoms-of-cnf* (*res* $S$) $\subseteq$ *atoms-of-cnf* $S$
⟨*proof*⟩

**lemma** *exlitE*: $(\bigwedge x.\ xe = Pos\ x \Longrightarrow P\ x) \Longrightarrow (\bigwedge x.\ xe = Neg\ x \Longrightarrow False) \Longrightarrow \exists x.\ xe = Pos\ x \wedge P\ x$
⟨*proof*⟩
**lemma** *res-in-all-clauses*: *res* $S \subseteq$ *all-clauses* $S$
⟨*proof*⟩

**lemma** *Res-in-all-clauses*: *res* $S \cup S \subseteq$ *all-clauses* $S$
⟨*proof*⟩
**lemma** *all-clauses-Res-inv*: *all-clauses* (*res* $S \cup S$) = *all-clauses* $S$
⟨*proof*⟩
**lemma** *all-clauses-finite*: *finite* $S \wedge (\forall C \in S.\ finite\ C) \Longrightarrow$ *finite* (*all-clauses* $S$)
⟨*proof*⟩
**lemma** *finite-res*: $\forall C \in S.\ finite\ C \Longrightarrow \forall C \in res\ S.\ finite\ C$
⟨*proof*⟩

**lemma** *finite* $T \Longrightarrow S \subseteq T \Longrightarrow$ *card* $S <$ *Suc* (*card* $T$)
⟨*proof*⟩

**lemma** *finite* $S \wedge (\forall C \in S.\ finite\ C) \Longrightarrow \exists T.\ Rwhile\ S = Some\ T$
⟨*proof*⟩

**partial-function**(*option*) *Res* **where**
*Res* $S$ = (*let* $R$ = *res* $S \cup S$ *in if* $R = S$ *then Some* $S$ *else Res* $R$)
**declare** *Res.simps*[*code*]

**value** [*code*] *Res ex1*
**lemma** $\square \in$ *the* (*Res ex1*) ⟨*proof*⟩

**lemma** *res*: $C \in res\ S \Longrightarrow S \vdash C$
⟨*proof*⟩

**lemma** *Res-sound*: *Res* $S$ = *Some* $S' \Longrightarrow (\forall C \in S'.\ S \vdash C)$
⟨*proof*⟩

**lemma** *Res-terminates*: *finite* $S \Longrightarrow \forall C \in S.\ finite\ C \Longrightarrow \exists T.\ Res\ S = Some\ T$
⟨*proof*⟩

**code-pred** *Resolution* ⟨*proof*⟩
**print-theorems**

**end**
**end**
**theory** *Resolution-Sound*

**imports** *Resolution CNF-Formulas-Sema*
**begin**

**lemma** *Resolution-insert*: $S \vdash R \Longrightarrow$ *cnf-semantics* $\mathcal{A}$ $S \Longrightarrow$ *cnf-semantics* $\mathcal{A}$ $\{R\}$
⟨*proof*⟩

**lemma** $S \vdash R \Longrightarrow$ *cnf-semantics* $\mathcal{A}$ $S \longleftrightarrow$ *cnf-semantics* $\mathcal{A}$ $(R \triangleright S)$
⟨*proof*⟩

**corollary** *Resolution-cnf-sound*: **assumes** $S \vdash \square$ **shows** $\neg$ *cnf-semantics* $\mathcal{A}$ $S$
⟨*proof*⟩

**corollary** *Resolution-sound*:
**assumes** *rp*: *cnf* (*nnf* $F$) $\vdash \square$
**shows** $\neg\ \mathcal{A} \models F$
⟨*proof*⟩

**end**

### 2.4.1 Completeness

**theory** *Resolution-Compl*
**imports** *Resolution CNF-Sema*
**begin**

Completeness proof following Schöning [9].

**definition** *make-lit* $v$ $a \equiv$ *case* $v$ *of True* $\Rightarrow$ *Pos* $a$ | *False* $\Rightarrow$ *Neg* $a$

**definition** *restrict-cnf-atom* $a$ $v$ $C \equiv \{c - \{$*make-lit* $(\neg v)$ $a\} \mid c.\ c \in C \wedge$ *make-lit* $v$ $a \notin c\}$

**lemma** *restrict-cnf-remove*: *atoms-of-cnf* (*restrict-cnf-atom* $a$ $v$ $c$) $\subseteq$
*atoms-of-cnf* $c - \{a\}$
⟨*proof*⟩

**lemma** *cnf-substitution-lemma*:
*cnf-semantics* $A$ (*restrict-cnf-atom* $a$ $v$ $S$) = *cnf-semantics* $(A(a := v))$ $S$
⟨*proof*⟩

**lemma** *finite-restrict*: *finite* $S \Longrightarrow$ *finite* (*restrict-cnf-atom* $a$ $v$ $S$)
⟨*proof*⟩

The next lemma describes what we have to (or can) do to a CNF after it has been mangled by *restrict-cnf-atom* to get back to (a subset of) the original CNF. The idea behind this will be clearer upon usage.

**lemma** *unrestrict-effects*:
(λc. *if* {*make-lit* (¬v) a} ∪ c ∈ S *then* {*make-lit* (¬v) a} ∪ c *else* c) ‘ *restrict-cnf-atom a v S* ⊆ S
⟨*proof*⟩

**lemma** *can-cope-with-unrestrict-effects*:
**assumes** *pr*: S ⊢ □
**assumes** *su*: S ⊆ T
**shows** ∃ R ⊆ {*make-lit v a*}. (λc. *if* c ∈ n *then* {*make-lit v a*} ∪ c *else* c) ‘ T ⊢ R
⟨*proof*⟩

**lemma** *unrestrict′*:
**fixes** R :: ′a *clause*
**assumes** *rp*: *restrict-cnf-atom a v S* ⊢ □
**shows** ∃ R ⊆ {*make-lit* (¬v) a}. S ⊢ R
⟨*proof*⟩

**lemma** *Resolution-complete-standalone-finite*:
**assumes** *ns*: ∀ A. ¬*cnf-semantics* A S
**assumes** *fin*: *finite* (*atoms-of-cnf S*)
**shows** S ⊢ □
⟨*proof*⟩

What you might actually want is ∀ A. ¬ *cnf-semantics* A S ⟹ S ⊢ □. Unfortunately, applying compactness (to get a finite set with a finite number of atoms) here is problematic: You would need to convert all clauses to disjunction-formulas, but there might be clauses with an infinite number of atoms. Removing those has to be done before applying compactness, we would possibly have to remove an infinite number of infinite clauses. Since the notion of a formula with an infinite number of atoms is not exactly standard, it is probably better to just skip this.

**end**
**theory** *Resolution-Compl-Consistency*
**imports** *Resolution Consistency CNF-Formulas CNF-Formulas-Sema*
**begin**

**lemma** *OrI2′*: (¬P ⟹ Q) ⟹ P ∨ Q ⟨*proof*⟩

**lemma** *atomD*: *Atom k* ∈ S ⟹ {*Pos k*} ∈ ⋃(*cnf* ‘ S) *Not* (*Atom k*) ∈ S ⟹ {*Neg k*} ∈ ⋃(*cnf* ‘ S) ⟨*proof*⟩

**lemma** *pcp-disj*:
⟦F ∨ G ∈ Γ; (∀ *xa*. (*xa* = F ∨ *xa* ∈ Γ) ⟶ *is-cnf xa*) ⟶ (*cnf F* ∪ (⋃x∈Γ. *cnf x*) ⊢ □); (∀ *xa*. (*xa* = G ∨ *xa* ∈ Γ) ⟶ *is-cnf xa*) ⟶ (*cnf G* ∪ (⋃x∈Γ. *cnf x*) ⊢ □); ∀ x∈Γ. *is-cnf x*⟧
⟹ (⋃x∈Γ. *cnf x*) ⊢ □
⟨*proof*⟩

**lemma** *R-consistent*: *pcp* $\{\Gamma|\Gamma.\ \neg((\forall\,\gamma \in \Gamma.\ \textit{is-cnf}\ \gamma) \longrightarrow ((\bigcup(\textit{cnf}\ `\ \Gamma)) \vdash \Box))\}$
⟨*proof*⟩

**theorem** *Resolution-complete*:
**fixes** $F :: 'a :: \textit{countable formula}$
**shows** $\models F \Longrightarrow \textit{cnf}\ (\textit{nnf}\ (\neg F)) \vdash \Box$
⟨*proof*⟩

**end**

# 3 Proof Transformation

This is organized as a ring closure

## 3.1 HC to SC

**theory** *HCSC*
**imports** *HC SC-Cut*
**begin**

**lemma** *extended-AxE*[*intro*!]: $F, \Gamma \Rightarrow F, \Delta$ ⟨*proof*⟩

**theorem** *HCSC*: $\textit{AX10} \cup \textit{set-mset}\ \Gamma \vdash_H F \Longrightarrow \Gamma \Rightarrow \{\#F\#\}$
⟨*proof*⟩

**end**

## 3.2 SC to ND

**theory** *SCND*
**imports** *SC ND*
**begin**

**lemma** *SCND*: $\Gamma \Rightarrow \Delta \Longrightarrow (\textit{set-mset}\ \Gamma) \cup \textit{Not}\ `\ (\textit{set-mset}\ \Delta) \vdash \bot$
⟨*proof*⟩

**end**

## 3.3 ND to HC

**theory** *NDHC*
**imports** *ND HC*
**begin**

The fundamental difference between the two is that Natural Deduction updates its set of assumptions while Hilbert Calculus does not. The Deduction

Theorem $AX0 \cup (?F \triangleright ?\Gamma) \vdash_H ?G \Longrightarrow AX0 \cup ?\Gamma \vdash_H ?F \rightarrow ?G$ helps with this.

**theorem** *NDHC*: $\Gamma \vdash F \Longrightarrow AX10 \cup \Gamma \vdash_H F$
⟨*proof*⟩

**end**

## 3.4 HC, SC, and ND

**theory** *HCSCND*
**imports** *HCSC SCND NDHC*
**begin**

**theorem** *HCSCND*:
  **defines** *hc* $F \equiv AX10 \vdash_H F$
  **defines** *nd* $F \equiv \{\} \vdash F$
  **defines** *sc* $F \equiv \{\#\} \Rightarrow \{\#\ F\ \#\}$
  **shows** *hc* $F \longleftrightarrow$ *nd* $F$ **and** *nd* $F \longleftrightarrow$ *sc* $F$ **and** *sc* $F \longleftrightarrow$ *hc* $F$
⟨*proof*⟩

**end**

## 3.5 Transforming SC proofs into proofs of CNFs

**theory** *LSC*
**imports** *CNF-Formulas SC-Cut*
**begin**

Left handed SC with NNF transformation:

**inductive** *LSC* (⟨(- $\Rightarrow_n$)⟩ [*53*]) **where**
— logic:
*Ax*: $\neg(Atom\ k), Atom\ k, \Gamma \Rightarrow_n$ |
*BotL*: $\bot, \Gamma \Rightarrow_n$ |
*AndL*: $F, G, \Gamma \Rightarrow_n \Longrightarrow F \wedge G, \Gamma \Rightarrow_n$ |
*OrL*: $F, \Gamma \Rightarrow_n \Longrightarrow G, \Gamma \Rightarrow_n \Longrightarrow F \vee G, \Gamma \Rightarrow_n$ |
— nnf rules:
*NotOrNNF*: $\neg F, \neg G, \Gamma \Rightarrow_n \Longrightarrow \neg(F \vee G), \Gamma \Rightarrow_n$ |
*NotAndNNF*: $\neg F, \Gamma \Rightarrow_n \Longrightarrow \neg G, \Gamma \Rightarrow_n \Longrightarrow \neg(F \wedge G), \Gamma \Rightarrow_n$ |
*ImpNNF*: $\neg F, \Gamma \Rightarrow_n \Longrightarrow G, \Gamma \Rightarrow_n \Longrightarrow F \rightarrow G, \Gamma \Rightarrow_n$ |
*NotImpNNF*: $F, \neg G, \Gamma \Rightarrow_n \Longrightarrow \neg(F \rightarrow G), \Gamma \Rightarrow_n$ |
*NotNotNNF*: $F, \Gamma \Rightarrow_n \Longrightarrow \neg(\neg F), \Gamma \Rightarrow_n$
**lemmas** *LSC.intros*[*intro*!]

You can prove that derivability in *SCp* is invariant to *nnf*, and then transform *SCp* to *LSC* while assuming NNF. However, the transformation introduces the trouble of handling the right side of *SCp*. The idea behind this is that handling the transformation is easier when not requiring NNF.

One downside of the whole approach is that we often need everything to be in NNF. To shorten:

**abbreviation** *is-nnf-mset* $\Gamma \equiv \forall x \in$ *set-mset* $\Gamma$. *is-nnf* $x$

**lemma** $\Gamma \Rightarrow \{\#\} \Longrightarrow$ *is-nnf-mset* $\Gamma \Longrightarrow \Gamma \Rightarrow_n$
⟨*proof*⟩

**lemma** *LSC-to-SC*:
**shows** $\Gamma \Rightarrow_n \Longrightarrow \Gamma \Rightarrow \{\#\}$
⟨*proof*⟩

**lemma** *SC-to-LSC*:
**assumes** $\Gamma \Rightarrow \Delta$
**shows** $\Gamma +$ (*image-mset Not* $\Delta$) $\Rightarrow_n$
⟨*proof*⟩

**corollary** *SC-LSC*: $\Gamma \Rightarrow \{\#\} \longleftrightarrow \Gamma \Rightarrow_n$ ⟨*proof*⟩

The nice thing: The NNF-Transformation is even easier to show on the one-sided variant.

**lemma** *LSC-NNF*: $\Gamma \Rightarrow_n \Longrightarrow$ *image-mset nnf* $\Gamma \Rightarrow_n$
⟨*proof*⟩

**lemma** *LSC-NNF-back*: *image-mset nnf* $\Gamma \Rightarrow_n \Longrightarrow \Gamma \Rightarrow_n$
⟨*proof*⟩

If we got rid of the rules for NNF, we could call it Gentzen-Schütte-calculus. But it turned out that not doing that works quite fine.

If you stare at left-handed Sequent calcului for too long, and they start staring back: Try imagining that there is a $\bot$ on the right hand side. Also, bear in mind that provability of $\Gamma \Rightarrow_n$ and satisfiability of $\Gamma$ are opposites here.

**lemma** *LHCut*:
**assumes** $F$,$\Gamma \Rightarrow_n \neg F$, $\Gamma \Rightarrow_n$
**shows** $\Gamma \Rightarrow_n$
⟨*proof*⟩

**lemma**
**shows** *LSC-AndL-inv*: $F \wedge G$,$\Gamma \Rightarrow_n \Longrightarrow F$,$G$,$\Gamma \Rightarrow_n$
**and** *LSC-OrL-inv*: $F \vee G$,$\Gamma \Rightarrow_n \Longrightarrow F$,$\Gamma \Rightarrow_n \wedge\ G$,$\Gamma \Rightarrow_n$
⟨*proof*⟩
**lemmas** *LSC-invs* = *LSC-AndL-inv LSC-OrL-inv*

**lemma** *LSC-weaken-set*: $\Gamma \Rightarrow_n \Longrightarrow \Gamma + \Theta \Rightarrow_n$
⟨*proof*⟩

**lemma** *LSC-weaken*: $\Gamma \Rightarrow_n \Longrightarrow F,\Gamma \Rightarrow_n$
⟨*proof*⟩

**lemma** *LSC-Contract*:
**assumes** *sfp*: $F$, $F$, $\Gamma \Rightarrow_n$
**shows** $F$, $\Gamma \Rightarrow_n$
⟨*proof*⟩

**lemma** *cnf*:
**shows**
$F \vee (G \wedge H)$, $\Gamma \Rightarrow_n \longleftrightarrow (F \vee G) \wedge (F \vee H)$, $\Gamma \Rightarrow_n$ (**is** *?l1* $\longleftrightarrow$ *?r1*)
$(G \wedge H) \vee F$, $\Gamma \Rightarrow_n \longleftrightarrow (G \vee F) \wedge (H \vee F)$, $\Gamma \Rightarrow_n$ (**is** *?l2* $\longleftrightarrow$ *?r2*)
⟨*proof*⟩

Interestingly, the DNF congruences are a lot easier to show, requiring neither weakening nor contraction. The reasons are to be sought in the asymmetries between the rules for $(\wedge)$ and $(\vee)$.

**lemma** *dnf*:
**shows**
$F \wedge (G \vee H)$, $\Gamma \Rightarrow_n \longleftrightarrow (F \wedge G) \vee (F \wedge H)$, $\Gamma \Rightarrow_n$ (**is** *?t1*)
$(G \vee H) \wedge F$, $\Gamma \Rightarrow_n \longleftrightarrow (G \wedge F) \vee (H \wedge F)$, $\Gamma \Rightarrow_n$ (**is** *?t2*)
⟨*proof*⟩

**lemma** *LSC-sim-Resolution1*:
**assumes** *R*: $S \vee T$, $\Gamma \Rightarrow_n$
**shows** *Atom k* $\vee$ $S$, $(\neg(\mathit{Atom}\ k)) \vee T$, $\Gamma \Rightarrow_n$
⟨*proof*⟩

**lemma**
*LSC-need-it-once-have-many*:
**assumes** *el*: $A \in \mathit{set}\ F$
**assumes** *once*: *form-of-lit A* $\vee$ *disj-of-clause* (*removeAll A F*),$\Gamma \Rightarrow_n$
**shows** *disj-of-clause F*,$\Gamma \Rightarrow_n$
⟨*proof*⟩

**lemma** *LSC-Sim-resolution-la*:
**fixes** $k :: 'a$
**assumes** *R*: *disj-of-clause* (*removeAll* $(k^+)$ *F* @ *removeAll* $(k^{-1})$ *G*), $\Gamma \Rightarrow_n$
**assumes** *el*: $k^+ \in \mathit{set}\ F\ k^{-1} \in \mathit{set}\ G$
**shows** *disj-of-clause F*, *disj-of-clause G*, $\Gamma \Rightarrow_n$
⟨*proof*⟩

**lemma** *two-list-induct*[*case-names Nil Cons*]: $P\ []\ [] \Longrightarrow (\bigwedge a\ S\ T.\ P\ S\ T \Longrightarrow P\ (a \#\ S)\ T\ \&\&\&\ P\ S\ (a \#\ T)) \Longrightarrow P\ S\ T$
⟨*proof*⟩

**lemma** *distrib1*: $[\![F, \Gamma \Rightarrow_n; \mathit{image\text{-}mset\ disj\text{-}of\text{-}clause}\ (\mathit{mset}\ G) + \Gamma \Rightarrow_n]\!]$
$\Longrightarrow \mathit{mset}\ (\mathit{map}\ (\lambda d.\ F \vee \mathit{disj\text{-}of\text{-}clause}\ d)\ G) + \Gamma \Rightarrow_n$

⟨*proof*⟩

**lemma** *mset-concat-map-cons*:
*mset* (*concat* (*map* (λ*c*. *F c* # *G c*) *S*)) = *mset* (*map F S*) + *mset* (*concat* (*map G S*))
⟨*proof*⟩

**lemma** *distrib*:
*image-mset disj-of-clause* (*mset F*) + Γ ⇒$_n$ ⟹
*image-mset disj-of-clause* (*mset G*) + Γ ⇒$_n$ ⟹
*mset* [*disj-of-clause c* ∨ *disj-of-clause d*. *c* ← *F*, *d* ← *G*] + Γ ⇒$_n$
⟨*proof*⟩

**lemma** *LSC-BigAndL*: *mset F* + Γ ⇒$_n$ ⟹ ⋀*F*, Γ ⇒$_n$
⟨*proof*⟩
**lemma** *LSC-Top-unused*: ⟦Γ ⇒$_n$; *is-nnf-mset* Γ⟧ ⟹ Γ − {#¬ ⊥#} ⇒$_n$
⟨*proof*⟩

**lemma** *LSC-BigAndL-inv*: ⋀*F*, Γ ⇒$_n$ ⟹ ∀ *f* ∈ *set F*. *is-nnf f* ⟹ *is-nnf-mset* Γ ⟹ *mset F* + Γ ⇒$_n$
⟨*proof*⟩

**lemma** *LSC-reassociate-Ands*: {#*disj-of-clause c* ∨ *disj-of-clause d*. (*c*, *d*) ∈# *C*#} + Γ ⇒$_n$ ⟹
*is-nnf-mset* Γ ⟹
{#*disj-of-clause* (*c* @ *d*). (*c*, *d*) ∈# *C*#} + Γ ⇒$_n$
⟨*proof*⟩

**lemma** *LSC-cnf*: Γ ⇒$_n$ ⟹ *is-nnf-mset* Γ ⟹ *image-mset cnf-form-of* Γ ⇒$_n$
⟨*proof*⟩

**end**

## 3.6 Converting between Resolution and SC proofs

**theory** *LSC-Resolution*
**imports** *LSC Resolution*
**begin**

**lemma** *literal-subset-sandwich*:
**assumes** *is-lit-plus F cnf F* = {*C*} *R* ⊆ *C*
**shows** *R* = □ ∨ *R* = *C*
⟨*proof*⟩

Proof following Gallier [3].

**theorem** *CSC-Resolution-pre*: Γ ⇒$_n$ ⟹ ∀ γ ∈ *set-mset* Γ. *is-cnf* γ ⟹ (⋃(*cnf* ‘ *set-mset* Γ)) ⊢ □
⟨*proof*⟩

**corollary** *LSC-Resolution*:
**assumes** $\Gamma \Rightarrow_n$
**shows** $(\bigcup (cnf \text{ ` } nnf \text{ ` } set\text{-}mset\ \Gamma)) \vdash \square$
⟨*proof*⟩

**corollary** *SC-Resolution*:
**assumes** $\Gamma \Rightarrow \{\#\}$
**shows** $(\bigcup (cnf \text{ ` } nnf \text{ ` } set\text{-}mset\ \Gamma)) \vdash \square$
⟨*proof*⟩

**lemma** *Resolution-LSC-pre*:
**assumes** $S \vdash R$
**assumes** *finite R*
**assumes** *finite S Ball S finite*
**shows** $\exists S'\ R'.\ \forall \Gamma.\ set\ R' = R \wedge set\ (map\ set\ S') = S \wedge$
$(disj\text{-}of\text{-}clause\ R', \{\#disj\text{-}of\text{-}clause\ c.\ c \in\# mset\ S'\#\} + \Gamma \Rightarrow_n \longrightarrow \{\#disj\text{-}of\text{-}clause\ c.\ c \in\# mset\ S'\#\} + \Gamma \Rightarrow_n)$

⟨*proof*⟩

**lemma** *Resolution-LSC-pre-nodisj*:
**assumes** $S \vdash R$
**assumes** *finite R*
**assumes** *finite S Ball S finite*
**shows** $\exists S'\ R'.\ \forall \Gamma.\ is\text{-}nnf\text{-}mset\ \Gamma \longrightarrow is\text{-}disj\ R' \wedge is\text{-}nnf\ S' \wedge cnf\ R' = \{R\} \wedge cnf\ S' \subseteq S \wedge$
$(R', S', \Gamma \Rightarrow_n \longrightarrow S', \Gamma \Rightarrow_n)$
⟨*proof*⟩

**corollary** *Resolution-LSC1*:
**assumes** $S \vdash \square$
**shows** $\exists F.\ is\text{-}nnf\ F \wedge cnf\ F \subseteq S \wedge \{\#F\#\} \Rightarrow_n$
⟨*proof*⟩

**corollary** *Resolution-SC1*:
**assumes** $S \vdash \square$
**shows** $\exists F.\ cnf\ (nnf\ F) \subseteq S \wedge \{\#F\#\} \Rightarrow \{\#\}$
⟨*proof*⟩

**end**
**theory** *ND-FiniteAssms*
**imports** *ND*
**begin**

**lemma** *ND-finite-assms*: $\Gamma \vdash F \Longrightarrow \exists \Gamma'.\ \Gamma' \subseteq \Gamma \wedge \mathit{finite}\ \Gamma' \wedge (\Gamma' \vdash F)$
⟨*proof*⟩

We thought that a lemma like this would be necessary for the ND completeness by SC completeness proof (this lemma shows that if we made an ND proof, we can always limit ourselves to a finite set of assumptions – and thus put all the assumptions into one formula). That is not the case, since in the completeness proof, we assume a valid entailment and have to show (the existence of) a derivation. The author hopes that his misunderstanding can help the reader's understanding.

**corollary** *ND-no-assms*:
  **assumes** $\Gamma \vdash F$
  **obtains** $\Gamma'$ **where** *set* $\Gamma' \subseteq \Gamma \wedge (\{\} \vdash \bigwedge \Gamma' \rightarrow F)$
⟨*proof*⟩

**end**

## 3.7 An alternate proof of ND completeness

**theory** *ND-Compl-SC*
**imports** *SC-Sema ND-Sound SCND Compactness*
**begin**

**lemma** *ND-sound-complete-countable*:
  **fixes** $\Gamma$ :: *'a* :: *countable formula set*
  **shows** $\Gamma \vdash F \longleftrightarrow \Gamma \models\!\models F$ (**is** *?n* $\longleftrightarrow$ *?s*)
⟨*proof*⟩

If you do not like the requirement that our atoms are countable, you can also restrict yourself to a finite set of assumptions.

**lemma** *ND-sound-complete-finite*:
  **assumes** *finite* $\Gamma$
  **shows** $\Gamma \vdash F \longleftrightarrow \Gamma \models\!\models F$ (**is** *?n* $\longleftrightarrow$ *?s*)
⟨*proof*⟩

**end**
**theory** *Resolution-Compl-SC-Small*
**imports** *LSC-Resolution Resolution SC-Sema CNF-Formulas-Sema*
**begin**

**lemma** *Resolution-complete'*:
  **assumes** *fin*: *finite S*
  **assumes** *val*: $S \models\!\models F$
  **shows** $\bigcup((\mathit{cnf} \circ \mathit{nnf})\ `\ (\{\neg F\} \cup S)) \vdash \square$
⟨*proof*⟩

**corollary** *Resolution-complete-single*:

**assumes** $\models F$
**shows** *cnf* (*nnf* ($\neg F$)) $\vdash \square$
⟨*proof*⟩

**end**
**theory** *Resolution-Compl-SC-Full*
**imports** *LSC-Resolution Resolution SC-Sema Compactness*
**begin**

**theorem** *Resolution-complete*:
**fixes** $S :: 'a ::$ *countable formula set*
**assumes** *val*: $S \models F$
**shows** $\bigcup((cnf \circ nnf)\ `(\{\neg F\} \cup S)) \vdash \square$

⟨*proof*⟩

**end**

## 3.8 SC and Implication-only formulas

**theory** *MiniSC*
**imports** *MiniFormulas SC*
**begin**

**abbreviation** *is-mini-mset* $\Gamma \equiv \forall F \in$ *set-mset* $\Gamma$. *is-mini-formula* $F$
**lemma** *to-mini-mset-is*: *is-mini-mset* (*image-mset to-mini-formula* $\Gamma$) ⟨*proof*⟩

**lemma** *SC-full-to-mini*:
**defines** *tms* $\equiv$ *image-mset to-mini-formula*
**assumes** *asm*: $\Gamma \Rightarrow \Delta$
**shows** *tms* $\Gamma \Rightarrow$ *tms* $\Delta$
⟨*proof*⟩

**lemma** *SC-mini-to-full*:
**defines** *tms* $\equiv$ *image-mset to-mini-formula*
**assumes** *asm*: *tms* $\Gamma \Rightarrow$ *tms* $\Delta$
**shows** $\Gamma \Rightarrow \Delta$
⟨*proof*⟩

**theorem** *MiniSC-eq*: *image-mset to-mini-formula* $\Gamma \Rightarrow$ *image-mset to-mini-formula* $\Delta \longleftrightarrow \Gamma \Rightarrow \Delta$
⟨*proof*⟩

**end**

### 3.8.1 SC to HC

**theory** *MiniSC-HC*
**imports** *MiniSC HC*

**begin**

**inductive-set** *AX1* **where**
$F \in AX0 \Longrightarrow F \in AX1$ |
$((F \rightarrow \bot) \rightarrow \bot) \rightarrow F \in AX1$
**lemma** *AX01*: $AX0 \subseteq AX1$ ⟨*proof*⟩
**lemma** *AX1-away*: $AX1 \cup \Gamma = AX0 \cup (\Gamma \cup AX1)$ ⟨*proof*⟩

**lemma** *Deduction1*: $F \triangleright (AX1 \cup \Gamma) \vdash_H \bot \longleftrightarrow (AX1 \cup \Gamma) \vdash_H (F \rightarrow \bot)$ ⟨*proof*⟩
**lemma** *Deduction2*: $(F \rightarrow \bot) \triangleright (AX1 \cup \Gamma) \vdash_H \bot \longleftrightarrow (AX1 \cup \Gamma) \vdash_H F$ (**is** *?l* $\longleftrightarrow$ *?r*)
⟨*proof*⟩

**lemma**
$\Gamma \Rightarrow \Delta \Longrightarrow$ *is-mini-mset* $\Gamma \Longrightarrow$ *is-mini-mset* $\Delta \Longrightarrow$
(*set-mset* $\Gamma \cup (\lambda F.\ F \rightarrow \bot)$ ‘ *set-mset* $\Delta \cup AX1) \vdash_H \bot$
⟨*proof*⟩

**end**

### 3.8.2 Craig Interpolation

**theory** *MiniSC-Craig*
**imports** *MiniSC Formulas*
**begin**

**abbreviation** *atoms-mset* **where** *atoms-mset* $\Theta \equiv \bigcup F \in$ *set-mset* $\Theta$. *atoms* $F$

**lemma** *interpolation-equal-styles*:
$(\forall \Gamma\ \Delta\ \Gamma'\ \Delta'.\ \Gamma + \Gamma' \Rightarrow \Delta + \Delta' \longrightarrow (\exists F :: 'a\ formula.\ \Gamma \Rightarrow F,\Delta \wedge F,\Gamma' \Rightarrow \Delta' \wedge atoms\ F \subseteq atoms\text{-}mset\ (\Gamma + \Delta) \wedge atoms\ F \subseteq atoms\text{-}mset\ (\Gamma' + \Delta')))$
$\longleftrightarrow$
$(\forall \Gamma\ \Delta.\ \Gamma \Rightarrow \Delta \longrightarrow (\exists F :: 'a\ formula.\ \Gamma \Rightarrow \{\#F\#\} \wedge \{\#F\#\} \Rightarrow \Delta \wedge atoms\ F \subseteq atoms\text{-}mset\ \Gamma \wedge atoms\ F \subseteq atoms\text{-}mset\ \Delta))$
⟨*proof*⟩

The original version of the interpolation theorem is due to Craig [1]. Our proof partly follows the approach of Troelstra and Schwichtenberg [11] but, especially with the mini formulas, adds its own spin.

**theorem** *SC-Craig-interpolation*:
**assumes** $\Gamma + \Gamma' \Rightarrow \Delta + \Delta'$
**obtains** $F$ **where**
$\Gamma \Rightarrow F,\Delta$
$F,\Gamma' \Rightarrow \Delta'$
*atoms* $F \subseteq$ *atoms-mset* $(\Gamma + \Delta)$
*atoms* $F \subseteq$ *atoms-mset* $(\Gamma' + \Delta')$
⟨*proof*⟩

Note that there is an extension to Craig interpolation: One can show that

atoms that only appear positively/negatively in the original formulas will only appear positively/negatively in the interpolant.

**abbreviation** *patoms-mset* $S \equiv \bigcup F{\in}$*set-mset* $S$. *fst* (*pn-atoms* $F$)
**abbreviation** *natoms-mset* $S \equiv \bigcup F{\in}$*set-mset* $S$. *snd* (*pn-atoms* $F$)

**theorem** *SC-Craig-interpolation-pn*:
**assumes** $\Gamma + \Gamma' \Rightarrow \Delta + \Delta'$
**obtains** $F$ **where**
$\Gamma \Rightarrow F, \Delta$
$F, \Gamma' \Rightarrow \Delta'$
*fst* (*pn-atoms* $F$) $\subseteq$ (*patoms-mset* $\Gamma \cup$ *natoms-mset* $\Delta$) $\cap$ (*natoms-mset* $\Gamma' \cup$ *patoms-mset* $\Delta'$)
*snd* (*pn-atoms* $F$) $\subseteq$ (*natoms-mset* $\Gamma \cup$ *patoms-mset* $\Delta$) $\cap$ (*patoms-mset* $\Gamma' \cup$ *natoms-mset* $\Delta'$)
⟨*proof*⟩

**end**

# References


[1] W. Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *The Journal of Symbolic Logic*, 22(03):250–268, 1957.

[2] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Graduate Texts in Computer Science. Springer, 1990.

[3] J. H. Gallier. *Logic for computer science: foundations of automatic theorem proving*. Courier Dover Publications, 2015.

[4] G. Gentzen. Untersuchungen über das logische Schließen. I. *Mathematische zeitschrift*, 39(1):176–210, 1935.

[5] J. Harrison. *Handbook of practical logic and automated reasoning*. Cambridge University Press, 2009.

[6] D. Hilbert. Die Grundlagen der Mathematik. In *Die Grundlagen der Mathematik*, pages 1–21. Springer, 1928.

[7] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

[8] I. Johansson. Der Minimalkalkül, ein reduzierter intuitionistischer Formalismus. *Compositio mathematica*, 4:119–136, 1937.

[9] U. Schöning. *Logik für Informatiker*. BI Wissenschaftsverlag Mannheim, 1987.

[10] R. M. Smullyan. A unifying principal in quantification theory. *Proceedings of the National Academy of Sciences*, 49(6):828–832, 1963.

[11] A. S. Troelstra and H. Schwichtenberg. *Basic proof theory.* Number 43 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2000.