

Propositional Proof Systems

Julius Michaelis and Tobias Nipkow

October 11, 2017

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

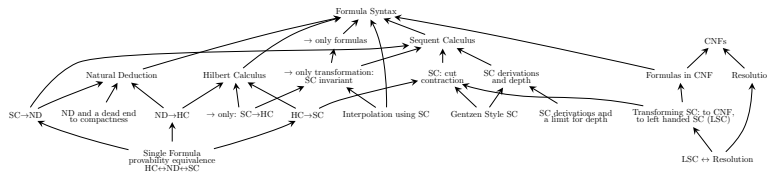


Figure 1: Overview of results considering Proof Transformation

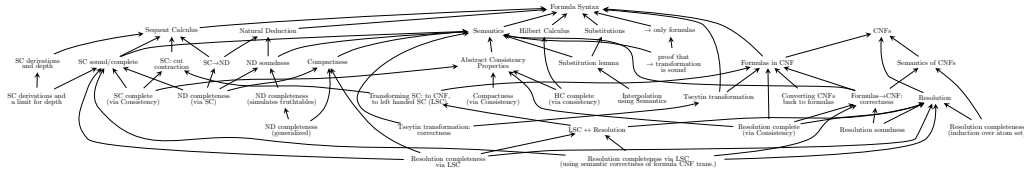


Figure 2: Overview of results considering Semantics

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 semantics 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
```

	<i>Bot</i>	(\perp)
	<i>Not 'a formula</i>	(\neg)
	<i>And 'a formula 'a formula</i>	(infix \wedge 68)
	<i>Or 'a formula 'a formula</i>	(infix \vee 68)
	<i>Imp 'a formula 'a formula</i>	(infixr \rightarrow 68)

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

primrec *subformulae* **where**

subformulae $\perp = [\perp]$ |
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 \subseteq set (subformulae F)*
<proof>

lemma *subsubformulae*: *G \in set (subformulae F) \implies H \in set (subformulae G)*
 \implies *H \in set (subformulae F)*
<proof>

lemma *subformula-atoms*: *G \in set (subformulae F) \implies atoms G \subseteq atoms F*
<proof>

lemma *subformulae-self*[*simp,intro*]: *F \in set (subformulae F)*
<proof>

lemma *subformulas-in-subformulas*:

G \wedge H \in set (subformulae F) \implies G \in set (subformulae F) \wedge H \in set (subformulae F)

G \vee H \in set (subformulae F) \implies G \in set (subformulae F) \wedge H \in set (subformulae F)

G \rightarrow H \in set (subformulae F) \implies G \in set (subformulae F) \wedge H \in set (subformulae F)

\neg G \in set (subformulae F) \implies G \in set (subformulae F)
<proof>

lemma *length-subformulae*: *length (subformulae F) = size F <proof>*

1.1 Derived Connectives

definition *Top* (\top) **where**

$\top \equiv \perp \rightarrow \perp$

lemma *top-atoms-simp*[*simp*]: *atoms $\top = \{\}$ <proof>*

primrec *BigAnd* :: 'a formula list \Rightarrow 'a formula (\bigwedge -) **where**
 $\bigwedge Nil = (\neg \perp)$ — 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. |
 $\bigwedge (F \# Fs) = F \wedge \bigwedge Fs$

lemma *atoms-BigAnd[simp]*: $atoms (\bigwedge Fs) = \bigcup (atoms \text{ ` set } Fs)$
 $\langle proof \rangle$

primrec *BigOr* :: 'a formula list \Rightarrow 'a formula (\bigvee -) **where**
 $\bigvee Nil = \perp$ |
 $\bigvee (F \# Fs) = F \vee \bigvee Fs$

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

instance *formula* :: (countable) countable $\langle proof \rangle$

definition *prod-unions* $A B \equiv \text{case } A \text{ of } (a,b) \Rightarrow \text{case } B \text{ of } (c,d) \Rightarrow (a \cup c, b \cup d)$

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*: $atoms F = fst (pn-atoms F) \cup snd (pn-atoms F)$
 $\langle proof \rangle$

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

context *begin*

definition *isstop* $F \equiv F = \neg \perp \vee F = \top$

fun *simplify-consts* **where**

$simplify-consts (Atom k) = Atom k$ |
 $simplify-consts \perp = \perp$ |
 $simplify-consts (Not F) = (let S = simplify-consts F in \text{case } S \text{ of } (Not G) \Rightarrow G |$
 $- \Rightarrow$
 $\text{if } isstop S \text{ then } \perp \text{ else } \neg S)$ |
 $simplify-consts (And F G) = (let S = simplify-consts F; T = simplify-consts G in$
 $($
 $\text{if } S = \perp \text{ then } \perp \text{ else}$
 $\text{if } isstop S \text{ then } T (* \text{ not } \top, T *) \text{ else}$
 $\text{if } T = \perp \text{ then } \perp \text{ else}$
 $\text{if } isstop T \text{ then } S \text{ else}$
 $\text{if } S = T \text{ then } S \text{ else}$
 $S \wedge T))$ |
 $simplify-consts (Or F G) = (let S = simplify-consts F; T = simplify-consts G in$
 $($
 $\text{if } S = \perp \text{ then } T \text{ else}$

```

    if isstop S then  $\top$  else
    if  $T = \perp$  then S else
    if isstop T then  $\top$  else
    if  $S = T$  then S else
     $S \vee T$ ) |
simplify-consts (Imp F G) = (let S = simplify-consts F; T = simplify-consts G in
(
  if  $S = \perp$  then  $\top$  else
  if isstop S then T else
  if isstop T then  $\top$  else
  if  $T = \perp$  then  $\neg S$  else
  if  $S = T$  then  $\top$  else
  case S of Not H  $\Rightarrow$  (case T of Not I  $\Rightarrow$ 
    I  $\rightarrow$  H | -  $\Rightarrow$ 
    H  $\vee$  T) | -  $\Rightarrow$ 
    S  $\rightarrow$  T))

```

lemma *simplify-consts-size-le*: $\text{size} (\text{simplify-consts } F) \leq \text{size } F$
 <proof>

lemma *simplify-const*: $\text{atoms } F = \{\} \implies \text{isstop} (\text{simplify-consts } F) \vee (\text{simplify-consts } F) = \perp$
 <proof>
value ($\text{size } \top$, $\text{size} (\neg \perp)$)

end

fun *all-formulas-of-size* **where**
all-formulas-of-size K 0 = $\{\perp\} \cup \text{Atom } 'K$ |
all-formulas-of-size K (Suc n) = (
 let af = $\bigcup \text{set} [\text{all-formulas-of-size } K \ m. \ m \leftarrow [0..<\text{Suc } n]]$ in
 ($\bigcup F \in \text{af}.$
 ($\bigcup G \in \text{af}.$ if $\text{size } F + \text{size } G \leq \text{Suc } n$ then $\{\text{And } F \ G, \text{Or } F \ G, \text{Imp } F \ G\}$ else
 $\{\}$)
 \cup (if $\text{size } F \leq \text{Suc } n$ then $\{\text{Not } F\}$ else $\{\}$))
 \cup af)

lemma *all-formulas-of-size*: $F \in \text{all-formulas-of-size } K \ n \longleftrightarrow (\text{size } F \leq \text{Suc } n \wedge \text{atoms } F \subseteq K)$ (**is** ?l \longleftrightarrow ?r)
 <proof>

end

1.2 Semantics

theory *Sema*

```

imports Formulas
begin

```

```

type-synonym 'a valuation = 'a  $\Rightarrow$  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-antics :: 'a valuation  $\Rightarrow$  'a formula  $\Rightarrow$  bool (infix  $\models$  51)
where

```

```

A  $\models$  Atom k = A k |
-  $\models \perp$  = False |
A  $\models$  Not F = ( $\neg$  A  $\models$  F) |
A  $\models$  And F G = (A  $\models$  F  $\wedge$  A  $\models$  G) |
A  $\models$  Or F G = (A  $\models$  F  $\vee$  A  $\models$  G) |
A  $\models$  Imp F G = (A  $\models$  F  $\longrightarrow$  A  $\models$  G)

```

```

abbreviation valid ( $\models$  - 51) where
 $\models$  F  $\equiv$   $\forall$  A. A  $\models$  F

```

```

lemma irrelevant-atom[simp]: A  $\notin$  atoms F  $\Longrightarrow$  (A(A := V))  $\models$  F  $\longleftrightarrow$  A  $\models$  F
  <proof>

```

```

lemma relevant-atoms-same-semantics:  $\forall$  k  $\in$  atoms F. A1 k = A2 k  $\Longrightarrow$  A1  $\models$ 
F  $\longleftrightarrow$  A2  $\models$  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 A (Atom k) = A k |
formula-semantics-alt A (Bot) = False |
formula-semantics-alt A (Not a) = (if formula-semantics-alt A a then False else
  True) |
formula-semantics-alt A (And a b) = (if formula-semantics-alt A a then formula-semantics-alt
  A b else False) |
formula-semantics-alt A (Or a b) = (if formula-semantics-alt A a then True else
  formula-semantics-alt A b) |
formula-semantics-alt A (Imp a b) = (if formula-semantics-alt A a then formula-semantics-alt
  A b else True)

```

```

private lemma formula-semantics-alt A F  $\longleftrightarrow$  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 A (Atom k) = A k |

```

```

formula-semantic-tt  $\mathcal{A}$  (Bot) = False |
formula-semantic-tt  $\mathcal{A}$  (Not a) = (case formula-semantic-tt  $\mathcal{A}$  a of True  $\Rightarrow$  False
| False  $\Rightarrow$  True) |
formula-semantic-tt  $\mathcal{A}$  (And a b) = (case (formula-semantic-tt  $\mathcal{A}$  a, formula-semantic-tt
 $\mathcal{A}$  b) of
  (False, False)  $\Rightarrow$  False
| (False, True)  $\Rightarrow$  False
| (True, False)  $\Rightarrow$  False
| (True, True)  $\Rightarrow$  True) |
formula-semantic-tt  $\mathcal{A}$  (Or a b) = (case (formula-semantic-tt  $\mathcal{A}$  a, formula-semantic-tt
 $\mathcal{A}$  b) of
  (False, False)  $\Rightarrow$  False
| (False, True)  $\Rightarrow$  True
| (True, False)  $\Rightarrow$  True
| (True, True)  $\Rightarrow$  True) |
formula-semantic-tt  $\mathcal{A}$  (Imp a b) = (case (formula-semantic-tt  $\mathcal{A}$  a, formula-semantic-tt
 $\mathcal{A}$  b) of
  (False, False)  $\Rightarrow$  True
| (False, True)  $\Rightarrow$  True
| (True, False)  $\Rightarrow$  False
| (True, True)  $\Rightarrow$  True)
private lemma  $A \models F \longleftrightarrow$  formula-semantic-tt  $A$  F
  <proof>
end

```

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

We write entailment differently than semantics (\models vs. \models). 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-semantic[simp,intro!]:  $A \models \top$  <proof>
lemma BigAnd-semantic[simp]:  $A \models \bigwedge F \longleftrightarrow (\forall f \in \text{set } F. A \models f)$  <proof>
lemma BigOr-semantic[simp]:  $A \models \bigvee F \longleftrightarrow (\exists f \in \text{set } F. A \models f)$  <proof>

```

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

```

definition sat  $S \equiv \exists \mathcal{A}. \forall F \in S. \mathcal{A} \models F$ 
definition fin-sat  $S \equiv (\forall s \subseteq S. \text{finite } s \longrightarrow \text{sat } s)$ 

```

```

lemma entail-sat:  $\Gamma \models \perp \longleftrightarrow \neg \text{sat } \Gamma$ 
  <proof>

```

```

lemma pn-atoms-updates:  $p \notin \text{snd } (\text{pn-atoms } F) \implies n \notin \text{fst } (\text{pn-atoms } F) \implies$ 
  (( $M \models F \longrightarrow (M(p := \text{True}) \models F \wedge M(n := \text{False}) \models F)$ ))  $\wedge$  (( $\neg(M \models F)$ )
 $\longrightarrow \neg(M(n := \text{True}) \models F) \wedge \neg(M(p := \text{False}) \models F)$ ))
  <proof>

```

lemma *const-simplifier-correct*: $\mathcal{A} \models \text{simplify-consts } F \longleftrightarrow \mathcal{A} \models F$
 ⟨*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 - - $\perp = \perp$ |

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**

$A[B / C] \equiv \text{subst } C B A$

lemma *no-subst[simp]*: $k \notin \text{atoms } F \implies F[G / k] = F$ ⟨*proof*⟩

lemma *subst-atoms*: $k \in \text{atoms } F \implies \text{atoms } (F[G / k]) = \text{atoms } F - \{k\} \cup$

atoms G

⟨*proof*⟩

lemma *subst-atoms-simp*: *atoms* ($F[G / k]$) = *atoms* $F - \{k\} \cup$ (*if* $k \in \text{atoms } F$

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* (($-$) [1000] 999)

type-synonym *'a clause* = *'a literal set*

abbreviation *empty-clause* (\square) **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* $c = \text{atoms-of-lit } \cdot \bigcup c$

lemma *atoms-of-cnf-alt*: *atoms-of-cnf* $c = \bigcup ((\text{op } \cdot \text{ atoms-of-lit}) \cdot c)$
{proof}

lemma *atoms-of-cnf-Un*: *atoms-of-cnf* ($S \cup T$) = *atoms-of-cnf* $S \cup \text{atoms-of-cnf}$
 T

{proof}

term $\{0^+\}$::nat clause

translations

$\{x\} \leq \text{CONST insert } x \square$

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 $\perp = \perp$ |

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* $-$) = 4^2 |

nnf-cost $\perp = 4^2$ |

nnf-cost (*Not* F) = *Suc* (*nnf-cost* F) |

nnf-cost (*And* F G) = *Suc* (*nnf-cost* F + *nnf-cost* G) |

$nnf\text{-cost } (Or\ F\ G) = Suc\ (nnf\text{-cost } F + nnf\text{-cost } G) \mid$
 $nnf\text{-cost } (Imp\ F\ G) = Suc\ (Suc\ (nnf\text{-cost } F + nnf\text{-cost } G))$

termination nnf $\langle proof \rangle$

lemma $nnf\ ((Atom\ (k::nat)) \rightarrow (Not\ ((Atom\ l) \vee (Not\ (Atom\ m)))) = \neg\ (Atom\ k) \vee (\neg\ (Atom\ l) \wedge Atom\ m)$
 $\langle proof \rangle$

fun $is\text{-lit-plus}$ **where**

$is\text{-lit-plus } \perp = True \mid$
 $is\text{-lit-plus } (Not\ \perp) = True \mid$
 $is\text{-lit-plus } (Atom\ -) = True \mid$
 $is\text{-lit-plus } (Not\ (Atom\ -)) = True \mid$
 $is\text{-lit-plus } - = False$

case-of-simps $is\text{-lit-plus-cases}$: $is\text{-lit-plus.simps}$

fun $is\text{-disj}$ **where**

$is\text{-disj } (Or\ F\ G) = (is\text{-lit-plus } F \wedge is\text{-disj } G) \mid$
 $is\text{-disj } F = is\text{-lit-plus } F$

fun $is\text{-cnf}$ **where**

$is\text{-cnf } (And\ F\ G) = (is\text{-cnf } F \wedge is\text{-cnf } G) \mid$
 $is\text{-cnf } H = is\text{-disj } H$

fun $is\text{-nnf}$ **where**

$is\text{-nnf } (Imp\ F\ G) = False \mid$
 $is\text{-nnf } (And\ F\ G) = (is\text{-nnf } F \wedge is\text{-nnf } G) \mid$
 $is\text{-nnf } (Or\ F\ G) = (is\text{-nnf } F \wedge is\text{-nnf } G) \mid$
 $is\text{-nnf } F = is\text{-lit-plus } F$

lemma $is\text{-nnf-nnf}$: $is\text{-nnf } (nnf\ F)$

$\langle proof \rangle$

lemma $nnf\text{-no-imp}$: $A \rightarrow B \notin set\ (subformulae\ (nnf\ F))$

$\langle proof \rangle$

lemma $subformulae\text{-nnf}$: $is\text{-nnf } F \Longrightarrow G \in set\ (subformulae\ F) \Longrightarrow is\text{-nnf } G$

$\langle proof \rangle$

lemma $is\text{-nnf-NotD}$: $is\text{-nnf } (\neg\ F) \Longrightarrow (\exists k. F = Atom\ k) \vee F = \perp$

$\langle proof \rangle$

fun cnf $:: 'a\ formula \Rightarrow 'a\ clause\ set$ **where**

$cnf\ (Atom\ k) = \{\{ k^+ \}\} \mid$
 $cnf\ (Not\ (Atom\ k)) = \{\{ k^{-1} \}\} \mid$
 $cnf\ \perp = \{\square\} \mid$
 $cnf\ (Not\ \perp) = \{\} \mid$
 $cnf\ (And\ F\ G) = cnf\ F \cup cnf\ G \mid$
 $cnf\ (Or\ F\ G) = \{C \cup D \mid C\ D. C \in (cnf\ F) \wedge D \in (cnf\ G)\}$

lemma $cnf\text{-fin}$:

assumes $is\text{-nnf } F$

shows $finite (cnf F) C \in cnf F \implies finite C$
 ⟨proof⟩

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

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

definition *disj-of-clause* $c \equiv \bigvee [form-of-lit l. l \leftarrow c]$
definition *form-of-cnf* $F \equiv \bigwedge [disj-of-clause c. c \leftarrow F]$
definition *cnf-form-of* $\equiv form-of-cnf \circ cnf-lists$
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 [] = \perp
disj-of-clause ($F \# FF$) = *form-of-lit* $F \vee disj-of-clause FF$
 ⟨proof⟩

lemma *is-cnf-BigAnd*: *is-cnf* ($\bigwedge ls$) $\longleftrightarrow (\forall l \in set\ ls. is-cnf\ l)$
 ⟨proof⟩ **lemma** *BigOr-is-not-cnf''*: *is-cnf* ($\bigvee ls$) $\implies (\forall l \in set\ ls. is-lit-plus\ l)$
 ⟨proof⟩ **lemma** *BigOr-is-not-cnf'*: $(\forall l \in set\ ls. is-lit-plus\ l) \implies is-cnf\ (\bigvee ls)$
 ⟨proof⟩

lemma *BigOr-is-not-cnf*: *is-cnf* ($\bigvee ls$) $\longleftrightarrow (\forall l \in set\ ls. is-lit-plus\ l)$
 ⟨proof⟩

lemma *is-nnf-BigAnd*[*simp*]: *is-nnf* ($\bigwedge ls$) $\longleftrightarrow (\forall l \in set\ ls. is-nnf\ l)$
 ⟨proof⟩

lemma *is-nnf-BigOr*[*simp*]: *is-nnf* ($\bigvee ls$) $\longleftrightarrow (\forall l \in set\ ls. is-nnf\ l)$
 ⟨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* $F \implies is-cnf\ (cnf-form-of\ F)$
 ⟨proof⟩

lemma *nnf-cnf-form*: $is\text{-}nnf\ F \implies is\text{-}nnf\ (cnf\text{-}form\text{-}of\ F)$
<proof>

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

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

lemma *cnf-cnf*: $is\text{-}nnf\ F \implies cnf\ (cnf\text{-}form\text{-}of\ F) = cnf\ F$
<proof>

lemma *is-nnf-nnf-id*: $is\text{-}nnf\ F \implies nnf\ F = F$
<proof>

lemma *disj-of-clause-is*: $is\text{-}disj\ (disj\text{-}of\text{-}clause\ R)$
<proof>

lemma *form-of-cnf-is-nnf*: $is\text{-}nnf\ (form\text{-}of\text{-}cnf\ R)$
<proof>

lemma *cnf-disj*: $cnf\ (disj\text{-}of\text{-}clause\ R) = \{set\ R\}$
<proof>

lemma *cnf-disj-ex*: $is\text{-}disj\ F \implies \exists R.\ cnf\ F = \{R\} \vee cnf\ F = \{\}$
<proof>

lemma *cnf-form-of-cnf*: $cnf\ (form\text{-}of\text{-}cnf\ S) = set\ (map\ set\ S)$
<proof>

lemma *disj-is-nnf*: $is\text{-}disj\ F \implies is\text{-}nnf\ F$
<proof>

lemma *nnf-BigAnd*: $nnf\ (\bigwedge F) = \bigwedge (map\ nnf\ F)$
<proof>

end

end

theory *CNF-Sema*

imports *CNF*

begin

primrec *lit-semantic* :: $('a \Rightarrow bool) \Rightarrow 'a\ literal \Rightarrow bool$ **where**

lit-semantic $\mathcal{A}\ (k^+) = \mathcal{A}\ k \mid$

lit-semantic $\mathcal{A}\ (k^{-1}) = (\neg \mathcal{A}\ k)$

case-of-simps *lit-semantic-cases*: *lit-semantic.simps*

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

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

lemma *nnf-semantic*: $\mathcal{A} \models \textit{nnf} F \longleftrightarrow \mathcal{A} \models F$
<proof>

lemma *cnf-semantic*: $\textit{is-nnf} F \implies \textit{cnf-semantic} \mathcal{A} (\textit{cnf} F) \longleftrightarrow \mathcal{A} \models F$
<proof>

lemma *cnf-form-semantic*:
fixes $F :: \textit{'a formula}$
assumes $\textit{nnf}: \textit{is-nnf} F$
shows $\mathcal{A} \models \textit{cnf-form-of} F \longleftrightarrow \mathcal{A} \models F$
<proof>

corollary $\exists G. \mathcal{A} \models F \longleftrightarrow \mathcal{A} \models G \wedge \textit{is-cnf} G$
<proof>

end

1.4.1 Going back: CNFs to formulas

theory *CNF-To-Formula*
imports *CNF-Formulas HOL-Library.List-lexord*
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* :: $(\textit{ord}) \textit{ord}$
begin

definition
literal-less-def: $xs < ys \longleftrightarrow ($
if $\textit{atoms-of-lit} xs = \textit{atoms-of-lit} ys$
then $(\textit{case} xs \textit{ of Neg } - \Rightarrow (\textit{case} ys \textit{ of Pos } - \Rightarrow \textit{True} \mid - \Rightarrow \textit{False}) \mid - \Rightarrow \textit{False})$
else $\textit{atoms-of-lit} xs < \textit{atoms-of-lit} ys)$

definition

literal-le-def: $(xs :: \text{- literal}) \leq ys \iff xs < ys \vee xs = ys$

instance $\langle \text{proof} \rangle$

end

instance *literal* :: (*linorder*) *linorder*
 $\langle \text{proof} \rangle$

definition *formula-of-cnf* **where**

formula-of-cnf $S \equiv \text{form-of-cnf} (\text{sorted-list-of-set} (\text{sorted-list-of-set} \text{' } S))$

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* ($\{\{Pos\ 0\}\} :: \text{nat clause set}$)) = *Atom 0*
 $\langle \text{proof} \rangle$

lemma *cnf-formula-of-cnf*:

assumes *finite* $S \forall C \in S. \text{finite } C$

shows *cnf* (*formula-of-cnf* S) = S

$\langle \text{proof} \rangle$

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* $f\ p \equiv (\text{case } p \text{ of } (k,v) \Rightarrow \text{fun-upd } f\ k\ v)$

lemma *fold-pair-upd-triv*: $A \notin \text{fst ' set } U \implies \text{foldl } \text{pair-fun-upd } F\ U\ A = F\ A$
 $\langle \text{proof} \rangle$

lemma *distinct-pair-update-one*: $(k,v) \in \text{set } U \implies \text{distinct } (\text{map } \text{fst } U) \implies \text{foldl } \text{pair-fun-upd } F\ U\ k = v$
 $\langle \text{proof} \rangle$

lemma *distinct-zipunzip*: $\text{distinct } xs \implies \text{distinct } (\text{map fst } (\text{zip } xs \ ys))$ $\langle \text{proof} \rangle$

lemma *foldl-pair-fun-upd-map-of*: $\text{distinct } (\text{map fst } U) \implies \text{foldl pair-fun-upd } F \ U$
 $= (\lambda k. \text{case map-of } U \ k \ \text{of } \text{Some } v \Rightarrow v \mid \text{None} \Rightarrow F \ k)$
 $\langle \text{proof} \rangle$

lemma *map-of-map-apsnd*: $\text{map-of } (\text{map } (\text{apsnd } t) \ M) = \text{map-option } t \circ (\text{map-of } M)$
 $\langle \text{proof} \rangle$

definition *biimp* (**infix** \leftrightarrow 67) **where** $F \leftrightarrow G \equiv (F \rightarrow G) \wedge (G \rightarrow F)$

lemma *atoms-biimp[simp]*: $\text{atoms } (F \leftrightarrow G) = \text{atoms } F \cup \text{atoms } G$
 $\langle \text{proof} \rangle$

lemma *biimp-size[simp]*: $\text{size } (F \leftrightarrow G) = (2 * (\text{size } F + \text{size } G)) + 3$
 $\langle \text{proof} \rangle$

locale *freshstuff* =

fixes *fresh* :: 'a set \Rightarrow 'a

assumes *isfresh*: $\text{finite } S \implies \text{fresh } S \notin S$

begin

primrec *nfresh* **where**

$\text{nfresh } S \ 0 = []$ |

$\text{nfresh } S \ (\text{Suc } n) = (\text{let } f = \text{fresh } S \ \text{in } f \ \# \ \text{nfresh } (f \triangleright S) \ n)$

lemma *length-nfresh*: $\text{length } (\text{nfresh } S \ n) = n$
 $\langle \text{proof} \rangle$

lemma *nfresh-isfresh*: $\text{finite } S \implies \text{set } (\text{nfresh } S \ n) \cap S = \{\}$
 $\langle \text{proof} \rangle$

lemma *nfresh-distinct*: $\text{finite } S \implies \text{distinct } (\text{nfresh } S \ n)$
 $\langle \text{proof} \rangle$

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

lemma *tseytin-assmt-distinct*: $\text{distinct } (\text{map fst } (\text{tseytin-assmt } F))$
 $\langle \text{proof} \rangle$

lemma *tseytin-assmt-has*: $G \in \text{set } (\text{subformulae } F) \implies \exists n. (n, G) \in \text{set } (\text{tseytin-assmt } F)$
 $\langle \text{proof} \rangle$

lemma *tseytin-assmt-new-atoms*: $(k, l) \in \text{set } (\text{tseytin-assmt } F) \implies k \notin \text{atoms } F$
 $\langle \text{proof} \rangle$

primrec *tseytin-tran1* **where**

tseytin-tran1 S (*Atom* k) = [*Atom* k \leftrightarrow S (*Atom* k)] |

tseytin-tran1 S \perp = [\perp \leftrightarrow S \perp] |

tseytin-tran1 S (*Not* F) = [S (*Not* F) \leftrightarrow *Not* (S F)] @ *tseytin-tran1* S F |

tseytin-tran1 S (*And* F G) = [S (*And* F G) \leftrightarrow *And* (S F) (S G)] @ *tseytin-tran1* S F @ *tseytin-tran1* S G |

tseytin-tran1 S (*Or* F G) = [S (*Or* F G) \leftrightarrow *Or* (S F) (S G)] @ *tseytin-tran1* S F @ *tseytin-tran1* S G |

tseytin-tran1 S (*Imp* F G) = [S (*Imp* F G) \leftrightarrow *Imp* (S F) (S G)] @ *tseytin-tran1* S F @ *tseytin-tran1* S G

definition *tseytin-toatom* F \equiv *Atom* \circ *the* \circ *map-of* (*map* ($\lambda(a,b). (b,a)$) (*tseytin-assmt* F))

definition *tseytin-tran* F \equiv \bigwedge (*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** $J \in$ *set* (*subformulae* F)

shows (*the* (*map-of* (*map* ($\lambda(a, b). (b, a)$) (*tseytin-assmt* F)) J), J) \in *set* (*tseytin-assmt* F)

<proof>

lemma *tseytin-tran-small-clauses*: $\forall C \in$ *cnf* (*nnf* (*tseytin-tran* F)). *card* $C \leq 3$

<proof>

lemma *tseytin-tran-few-clauses*: *card* (*cnf* (*nnf* (*tseytin-tran* F))) $\leq 3 * \text{size } F + 1$

<proof>

lemma *tseytin-tran-new-atom-count*: *card* (*atoms* (*tseytin-tran* F)) \leq *size* $F +$ *card* (*atoms* F)

<proof>

end

definition *freshnat* S \equiv *Suc* (*Max* ($0 \triangleright S$))

primrec *nfresh-natcode* **where**

nfresh-natcode S $0 = []$ |

nfresh-natcode S (*Suc* n) = (*let* $f =$ *freshnat* S *in* f # *nfresh-natcode* ($f \triangleright S$) 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* $0 \rightarrow (\neg$ (*Atom* 1))) = \bigwedge [
 Atom 2 ,


```

    Atom 2 ↔ Atom 3 → Atom 4,
    Atom 0 ↔ Atom 3,
    Atom 4 ↔ ¬ (Atom 5),
    Atom 1 ↔ 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  $\mathcal{A} F \equiv (\text{let } U = \text{map } (\text{apsnd } (\text{formula-antics } \mathcal{A}))$ 
  ( $\text{tseytin-assmt } F$ ) in foldl pair-fun-upd  $\mathcal{A} U)$ 

```

```

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

```

```

lemma  $(k, G) \in \text{set } (\text{tseytin-assmt } F) \implies \text{tseytin-update } \mathcal{A} F k \longleftrightarrow \text{tseytin-update}$ 
 $\mathcal{A} F \models G$ 
  ⟨proof⟩

```

```

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

```

```

lemma tseytin-tran1:  $G \in \text{set } (\text{subformulae } F) \implies H \in \text{set } (\text{tseytin-tran1 } S G)$ 
 $\implies \forall J \in \text{set } (\text{subformulae } F). \text{tseytin-update } \mathcal{A} F \models J \longleftrightarrow \text{tseytin-update } \mathcal{A} F$ 
 $\models (S J) \implies \text{tseytin-update } \mathcal{A} F \models H$ 
  ⟨proof⟩

```

```

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

```

```

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

```

```

lemma tseytin-tran1-orig-connection:  $G \in \text{set } (\text{subformulae } F) \implies (\forall H \in \text{set } (\text{tseytin-tran1}$ 
  ( $\text{tseytin-toatom } F$ )  $G$ ).  $\mathcal{A} \models H \implies$ 
   $\mathcal{A} \models G \longleftrightarrow \mathcal{A} \models \text{tseytin-toatom } F G$ 

```

```

    <proof>

lemma tseytin-untran:  $\mathcal{A} \models (tseytin-tran F) \implies \mathcal{A} \models F$ 
    <proof>
lemma tseytin-tran-equiumsatisfiable:  $\models \neg F \longleftrightarrow \models \neg (tseytin-tran F)$  (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  $\perp$  = 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 $F \rightarrow \perp$ 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  $\perp$  =  $\perp$  |
  to-mini-formula (Imp F G) = to-mini-formula F  $\rightarrow$  to-mini-formula G |
  to-mini-formula (Not F) = to-mini-formula F  $\rightarrow \perp$  |
  to-mini-formula (And F G) = (to-mini-formula F  $\rightarrow$  (to-mini-formula G  $\rightarrow \perp$ ))
   $\rightarrow \perp$  |
  to-mini-formula (Or F G) = (to-mini-formula F  $\rightarrow \perp$ )  $\rightarrow$  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  $\implies$  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 *op* \wedge . The choice for *Atom*, \perp , and *op* \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 $A \models F \longleftrightarrow A \models \text{to-mini-formula } F$

<proof>

end

1.6 Consistency

We follow the proofs by Melvin Fitting [2].

theory *Consistency*

imports *Sema*

begin

definition *Hintikka S* \equiv (

$\perp \notin S$

$\wedge (\forall k. \text{Atom } k \in S \longrightarrow \neg (\text{Atom } k) \in S \longrightarrow \text{False})$

$\wedge (\forall F G. F \wedge G \in S \longrightarrow F \in S \wedge G \in S)$

$\wedge (\forall F G. F \vee G \in S \longrightarrow F \in S \vee G \in S)$

$\wedge (\forall F G. F \rightarrow G \in S \longrightarrow \neg F \in S \vee G \in S)$

$\wedge (\forall F. \neg (\neg F) \in S \longrightarrow F \in S)$

$\wedge (\forall F G. \neg (F \wedge G) \in S \longrightarrow \neg F \in S \vee \neg G \in S)$

$\wedge (\forall F G. \neg (F \vee G) \in S \longrightarrow \neg F \in S \wedge \neg G \in S)$

$\wedge (\forall F G. \neg (F \rightarrow G) \in S \longrightarrow F \in S \wedge \neg G \in S)$

)

lemma *Hintikka* $\{ \text{Atom } 0 \wedge ((\neg (\text{Atom } 1)) \rightarrow \text{Atom } 2), ((\neg (\text{Atom } 1)) \rightarrow \text{Atom } 2), \text{Atom } 0, \neg(\neg (\text{Atom } 1)), \text{Atom } 1 \}$

<proof>

theorem *Hintikkas-lemma:*

assumes *H: Hintikka S*

shows *sat S*

<proof>

definition $pcp\ C \equiv (\forall S \in C.$

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

lemma $pcp\ \{\}\ pcp\ \{\{\}\}\ pcp\ \{\{Atom\ 0\}\}\ \langle proof \rangle$

lemma $pcp\ \{\{\neg (Atom\ 1) \rightarrow Atom\ 2\},$
 $\{(\neg (Atom\ 1) \rightarrow Atom\ 2), \neg (\neg (Atom\ 1))\},$
 $\{(\neg (Atom\ 1) \rightarrow Atom\ 2), \neg (\neg (Atom\ 1)), Atom\ 1\}\ \langle proof \rangle$

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 $Con :: 'a\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula \Rightarrow bool$ **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 $Dis :: 'a\ formula \Rightarrow 'a\ formula \Rightarrow 'a\ formula \Rightarrow bool$ **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 $Con\ (Not\ (Not\ F))\ F\ F\ Dis\ (Not\ (Not\ F))\ F\ F\ \langle proof \rangle$

lemma $con-dis-simps:$

$Con\ a1\ a2\ a3 = (a1 = a2 \wedge a3 \vee (\exists F\ G. a1 = \neg (F \vee G) \wedge a2 = \neg F \wedge a3 = \neg G) \vee (\exists G. a1 = \neg (a2 \rightarrow G) \wedge a3 = \neg G) \vee a1 = \neg (\neg a2) \wedge a3 = a2)$
 $Dis\ a1\ a2\ a3 = (a1 = a2 \vee a3 \vee (\exists F\ G. a1 = F \rightarrow G \wedge a2 = \neg F \wedge a3 = G) \vee (\exists F\ G. a1 = \neg (F \wedge G) \wedge a2 = \neg F \wedge a3 = \neg G) \vee a1 = \neg (\neg a2) \wedge a3 = a2)$
 $\langle proof \rangle$

lemma $Hintikka-alt: Hintikka\ S = ($

$\perp \notin S$
 $\wedge (\forall k. Atom\ k \in S \longrightarrow \neg (Atom\ k) \in S \longrightarrow False)$
 $\wedge (\forall F\ G\ H. Con\ F\ G\ H \longrightarrow F \in S \longrightarrow G \in S \wedge H \in S)$

$\wedge (\forall F G H. \text{Dis } F G H \longrightarrow F \in S \longrightarrow G \in S \vee H \in S)$
)
 <proof>

lemma *pcp-alt*: $\text{pcp } C = (\forall S \in C.$

$\perp \notin S$
 $\wedge (\forall k. \text{Atom } k \in S \longrightarrow \neg (\text{Atom } k) \in S \longrightarrow \text{False})$
 $\wedge (\forall F G H. \text{Con } F G H \longrightarrow F \in S \longrightarrow G \triangleright H \triangleright S \in C)$
 $\wedge (\forall F G H. \text{Dis } F G H \longrightarrow F \in S \longrightarrow G \triangleright S \in C \vee H \triangleright S \in C)$
)
 <proof>

definition *subset-closed* $C \equiv (\forall S \in C. \forall s \subseteq S. s \in C)$

definition *finite-character* $C \equiv (\forall S. S \in C \longleftrightarrow (\forall s \subseteq S. \text{finite } s \longrightarrow s \in C))$

lemma *ex1*: $\text{pcp } C \Longrightarrow \exists C'. C \subseteq C' \wedge \text{pcp } C' \wedge \text{subset-closed } C'$

<proof>

lemma *sall1*: $(\bigwedge s. s \subseteq S \Longrightarrow P s) \Longrightarrow \forall s \subseteq S. P s$ <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*: $\exists C'. C \subseteq C' \wedge \text{pcp } C' \wedge \text{finite-character } C'$

<proof>

primrec *pcp-seq* **where**

pcp-seq $C S 0 = S$ |

pcp-seq $C S (\text{Suc } n) = ($

let $S_n = \text{pcp-seq } C S n$; $S_{n1} = \text{from-nat } n \triangleright S_n$ *in*

if $S_{n1} \in C$ *then* S_{n1} *else* S_n

)

lemma *pcp-seq-in*: $\text{pcp } C \Longrightarrow S \in C \Longrightarrow \text{pcp-seq } C S n \in C$

<proof>

lemma *pcp-seq-mono*: $n \leq m \Longrightarrow \text{pcp-seq } C S n \subseteq \text{pcp-seq } C S m$

<proof>

lemma *pcp-seq-UN*: $\bigcup \{\text{pcp-seq } C S n \mid n. n \leq m\} = \text{pcp-seq } C S m$

<proof>

lemma *wont-get-added*: $(F :: ('a :: \text{countable}) \text{formula}) \notin \text{pcp-seq } C S (\text{Suc } (\text{to-nat } F)) \Longrightarrow F \notin \text{pcp-seq } C S (\text{Suc } (\text{to-nat } F) + n)$

We don't necessarily have $n = \text{to-nat } (\text{from-nat } n)$, so this doesn't hold.

<proof>

definition $\text{pcp-lim } C S \equiv \bigcup \{\text{pcp-seq } C S n \mid n. \text{ True}\}$

lemma $\text{pcp-seq-sub: } \text{pcp-seq } C S n \subseteq \text{pcp-lim } C S$

<proof>

lemma $\text{pcp-lim-inserted-at-ex: } x \in \text{pcp-lim } C S \implies \exists k. x \in \text{pcp-seq } C S k$

<proof>

lemma pcp-lim-in:

assumes $c: \text{pcp } C$

and $el: S \in C$

and $sc: \text{subset-closed } C$

and $fc: \text{finite-character } C$

shows $\text{pcp-lim } C S \in C$ (**is** $?cl \in C$)

<proof>

lemma cl-max:

assumes $c: \text{pcp } C$

assumes $sc: \text{subset-closed } C$

assumes $el: K \in C$

assumes $su: \text{pcp-lim } C S \subseteq K$

shows $\text{pcp-lim } C S = K$ (**is** $?e$)

<proof>

lemma cl-max':

assumes $c: \text{pcp } C$

assumes $sc: \text{subset-closed } C$

shows $F \triangleright \text{pcp-lim } C S \in C \implies F \in \text{pcp-lim } C S$

$F \triangleright G \triangleright \text{pcp-lim } C S \in C \implies F \in \text{pcp-lim } C S \wedge G \in \text{pcp-lim } C S$

<proof>

lemma pcp-lim-Hintikka:

assumes $c: \text{pcp } C$

assumes $sc: \text{subset-closed } C$

assumes $fc: \text{finite-character } C$

assumes $el: S \in C$

shows $\text{Hintikka } (\text{pcp-lim } C S)$

<proof>

theorem pcp-sat: — model existence theorem

fixes $S :: 'a :: \text{countable formula set}$

assumes $c: \text{pcp } C$

assumes $el: S \in C$

shows $\text{sat } S$

<proof>

end

1.7 Compactness

theory *Compactness*

imports *Sema*

begin

lemma *fin-sat-extend*: $\text{fin-sat } S \implies \text{fin-sat } (\text{insert } F S) \vee \text{fin-sat } (\text{insert } (\neg F) S)$
<proof>

context

begin

lemma *fin-sat-antimono*: $\text{fin-sat } F \implies G \subseteq F \implies \text{fin-sat } G$ *<proof>*

lemmas *fin-sat-insert* = *fin-sat-antimono*[*OF - subset-insertI*]

primrec *extender* :: $\text{nat} \Rightarrow ('a :: \text{countable}) \text{ formula set} \Rightarrow 'a \text{ formula set}$ **where**

extender 0 $S = S$ |

extender (Suc n) $S =$ (

let $r = \text{extender } n S$;

$rt = \text{insert } (\text{from-nat } n) r$;

$rf = \text{insert } (\neg(\text{from-nat } n)) r$

in if fin-sat rf then rf else rt

)

private lemma *extender-fin-sat*: $\text{fin-sat } S \implies \text{fin-sat } (\text{extender } n S)$

<proof>

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

lemma *extended-max*: $F \in \text{extended } S \vee \text{Not } F \in \text{extended } S$

<proof> **lemma** *extender-Sucset*: $\text{extender } k S \subseteq \text{extender } (\text{Suc } k) S$ *<proof>* **lemma**

extender-deeper: $F \in \text{extender } k S \implies k \leq l \implies F \in \text{extender } l S$ *<proof>* **lemma**

extender-subset: $S \subseteq \text{extender } k S$

<proof>

lemma *extended-fin-sat*:

assumes *fin-sat* S

shows *fin-sat* (*extended* S)

<proof>

lemma *extended-superset*: $S \subseteq \text{extended } S$ *<proof>*

lemma *extended-complem*:

assumes *fs*: *fin-sat* S

shows $F \in \text{extended } S \neq (\text{Not } F \in \text{extended } S)$

<proof>

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

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*: $S \subseteq T \implies \text{extended } S \subseteq \text{extended } T$

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*: $\exists S T. \text{fin-sat } S \wedge \text{fin-sat } T \wedge \neg (S \subseteq T \implies \text{extended } S \subseteq \text{extended } T)$ ($T :: 'a :: \text{countable formula set}$)

<proof> **lemma** *not-in-extended-FE*: $\text{fin-sat } S \implies (\neg \text{sat } (\text{insert } (\text{Not } F) G)) \implies F \notin \text{extended } S \implies G \subseteq \text{extended } S \implies \text{finite } G \implies \text{False}$

<proof>

lemma *extended-id*: $\text{extended } (\text{extended } S) = \text{extended } S$

<proof>

lemma *ext-model*:

assumes $r: \text{fin-sat } S$

shows $(\lambda k. \text{Atom } k \in \text{extended } S) \models F \longleftrightarrow F \in \text{extended } S$

<proof>

theorem *compactness*:

fixes $S :: 'a :: \text{countable formula set}$

shows $\text{sat } S \longleftrightarrow \text{fin-sat } S$ (**is** $?l = ?r$)

<proof>

corollary *compact-entailment*:

fixes $F :: 'a :: \text{countable formula}$

assumes $\text{fent}: \Gamma \models F$

shows $\exists \Gamma'. \text{finite } \Gamma' \wedge \Gamma' \subseteq \Gamma \wedge \Gamma' \models F$

<proof>

corollary *compact-to-formula*:

fixes $F :: 'a :: \text{countable formula}$

assumes $\text{fent}: \Gamma \models F$


```

obtains  $\Gamma'$  where set  $\Gamma' \subseteq \Gamma \models (\bigwedge \Gamma') \rightarrow F$ 
<proof>

end

end
theory Compactness-Consistency
imports Consistency
begin

theorem sat  $S \leftrightarrow \text{fin-sat } (S :: 'a :: \text{countable formula set})$  (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  $\mathcal{A} \models ((F[\top / n]) \vee (F[\perp / n]))$ 
<proof>

```

```

theorem interpolation:
  assumes  $\models \Gamma \rightarrow \Delta$ 
  obtains  $\varrho$  where
     $\models \Gamma \rightarrow \varrho \models \varrho \rightarrow \Delta$ 
    atoms  $\varrho \subseteq \text{atoms } \Gamma$ 
    atoms  $\varrho \subseteq \text{atoms } \Delta$ 
<proof>

```

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

```

function interpolate where
  interpolate  $F H = ($ 
    let  $K = \text{atoms } F - \text{atoms } H$  in
    if  $K = \{\}$ 
    then  $F$ 
    else (
      let  $k = \text{Min } K$ 
      in interpolate  $((F[\top / k]) \vee (F[\perp / k])) H$ 
    )
  ) <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 $P = \text{Atom } (0 :: \text{nat}); Q = \text{Atom } 1; R = \text{Atom } 2; T = \text{Atom } 3;$
 $\varphi = (\neg(P \wedge Q)) \rightarrow (\neg R \wedge Q);$
 $\psi = (T \rightarrow P) \vee (T \rightarrow (\neg R));$
 $I = \text{interpolate } \varphi \ \psi \text{ in}$
 (size I) = 23 ∧ *simplify-consts* $I = \text{Atom } 2 \rightarrow \text{Atom } 0$
 ⟨*proof*⟩

theorem *nonexistential-interpolation*:

assumes $\models F \rightarrow H$
shows
 $\models F \rightarrow \text{interpolate } F \ H \ (\text{is } ?t1) \models \text{interpolate } F \ H \rightarrow H \ (\text{is } ?t2)$
 $\text{atoms } (\text{interpolate } F \ H) \subseteq \text{atoms } F \cap \text{atoms } H \ (\text{is } ?s)$
 ⟨*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** $x, M == \text{add-mset } x \ M$

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

inductive *SCp* :: 'a formula multiset \Rightarrow 'a formula multiset \Rightarrow bool ((- \Rightarrow / -) [53] 53) **where**
 $\text{BotL: } \perp \in\# \Gamma \Longrightarrow \Gamma \Rightarrow \Delta \mid$
 $\text{Ax: } \text{Atom } k \in\# \Gamma \Longrightarrow \text{Atom } k \in\# \Delta \Longrightarrow \Gamma \Rightarrow \Delta \mid$
 $\text{NotL: } \Gamma \Rightarrow F, \Delta \Longrightarrow \text{Not } F, \Gamma \Rightarrow \Delta \mid$
 $\text{NotR: } F, \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \text{Not } F, \Delta \mid$
 $\text{AndL: } F, G, \Gamma \Rightarrow \Delta \Longrightarrow \text{And } F \ G, \Gamma \Rightarrow \Delta \mid$

AndR: $\llbracket \Gamma \Rightarrow F, \Delta; \Gamma \Rightarrow G, \Delta \rrbracket \Longrightarrow \Gamma \Rightarrow \text{And } F \ G, \Delta \mid$
OrL: $\llbracket F, \Gamma \Rightarrow \Delta; G, \Gamma \Rightarrow \Delta \rrbracket \Longrightarrow \text{Or } F \ G, \Gamma \Rightarrow \Delta \mid$
OrR: $\Gamma \Rightarrow F, G, \Delta \Longrightarrow \Gamma \Rightarrow \text{Or } F \ G, \Delta \mid$
ImpL: $\llbracket \Gamma \Rightarrow F, \Delta; G, \Gamma \Rightarrow \Delta \rrbracket \Longrightarrow \text{Imp } F \ G, \Gamma \Rightarrow \Delta \mid$
ImpR: $F, \Gamma \Rightarrow G, \Delta \Longrightarrow \Gamma \Rightarrow \text{Imp } F \ G, \Delta$

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

lemma

shows *BotL-canonical*[*intro!*]: $\perp, \Gamma \Rightarrow \Delta$
and *Ax-canonical*[*intro!*]: *Atom* $k, \Gamma \Rightarrow \text{Atom } k, \Delta$
 ⟨*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-MATCH $(I \rightarrow J) \ H \Longrightarrow H, F \rightarrow G, S = F \rightarrow G, H, S$
NO-MATCH $(I \rightarrow J) \ H \Longrightarrow \text{NO-MATCH } (I \vee J) \ H \Longrightarrow H, F \vee G, S = F \vee G, H, S$
NO-MATCH $(I \rightarrow J) \ H \Longrightarrow \text{NO-MATCH } (I \wedge J) \ H \Longrightarrow \text{NO-MATCH } (I \wedge J) \ H \Longrightarrow H, F \wedge G, S = F \wedge G, H, S$
NO-MATCH $(I \rightarrow J) \ H \Longrightarrow \text{NO-MATCH } (I \vee J) \ H \Longrightarrow \text{NO-MATCH } (I \wedge J) \ H \Longrightarrow \text{NO-MATCH } (\neg J) \ H \Longrightarrow H, \neg G, S = \neg G, H, S$
NO-MATCH $(I \rightarrow J) \ H \Longrightarrow \text{NO-MATCH } (I \vee J) \ H \Longrightarrow \text{NO-MATCH } (I \wedge J) \ H \Longrightarrow \text{NO-MATCH } (\neg J) \ H \Longrightarrow \text{NO-MATCH } (\perp) \ H \Longrightarrow H, \perp, S = \perp, H, S$
NO-MATCH $(I \rightarrow J) \ H \Longrightarrow \text{NO-MATCH } (I \vee J) \ H \Longrightarrow \text{NO-MATCH } (I \wedge J) \ H \Longrightarrow \text{NO-MATCH } (\neg J) \ H \Longrightarrow \text{NO-MATCH } (\perp) \ H \Longrightarrow \text{NO-MATCH } (\text{Atom } k) \ H \Longrightarrow H, \text{Atom } l, S = \text{Atom } l, H, S$

⟨*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*: $\Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta - \{\#\perp\#\}$
<proof>

corollary *Bot-delR-simp*: $\Gamma \Rightarrow \perp, \Delta = \Gamma \Rightarrow \Delta$
<proof>

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

2.1.1 Contraction

First, we need contraction:

lemma *contract*:

$$(F, F, \Gamma \Rightarrow \Delta \longrightarrow F, \Gamma \Rightarrow \Delta) \wedge (\Gamma \Rightarrow F, F, \Delta \longrightarrow \Gamma \Rightarrow F, \Delta)$$

<proof>

corollary

shows *contractL*: $F, F, \Gamma \Rightarrow \Delta \Longrightarrow F, \Gamma \Rightarrow \Delta$

and *contractR*: $\Gamma \Rightarrow F, F, \Delta \Longrightarrow \Gamma \Rightarrow F, \Delta$

<proof>

2.1.2 Cut

Which cut rule we show is up to us:

lemma *cut-cs-cf*:

assumes *context-sharing-Cut*: $\bigwedge(A::'a \text{ formula}) \Gamma \Delta. \Gamma \Rightarrow A, \Delta \Longrightarrow A, \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta$

shows *context-free-Cut*: $\Gamma \Rightarrow (A::'a \text{ formula}), \Delta \Longrightarrow A, \Gamma' \Rightarrow \Delta' \Longrightarrow \Gamma + \Gamma' \Rightarrow \Delta + \Delta'$

<proof>

lemma *cut-cf-cs*:

assumes *context-free-Cut*: $\bigwedge(A::'a \text{ formula}) \Gamma \Gamma' \Delta \Delta'. \Gamma \Rightarrow A, \Delta \Longrightarrow A, \Gamma' \Rightarrow \Delta' \Longrightarrow \Gamma + \Gamma' \Rightarrow \Delta + \Delta'$

shows *context-sharing-Cut*: $\Gamma \Rightarrow (A::'a \text{ formula}), \Delta \Longrightarrow A, \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta$

<proof>

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

lemma *Cut-Atom-pre*: $Atom\ k, \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow Atom\ k, \Delta \Longrightarrow \Gamma \Rightarrow \Delta$

<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*: $\Gamma \Rightarrow F, \Delta \Longrightarrow F, \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta$

<proof>

corollary *cut-cf*: $\llbracket \Gamma \Rightarrow A, \Delta; A, \Gamma' \Rightarrow \Delta' \rrbracket \Longrightarrow \Gamma + \Gamma' \Rightarrow \Delta + \Delta'$

<proof>

lemma **assumes** *cut*: $\bigwedge \Gamma' \Delta' (A::'a \text{ formula}). \llbracket \Gamma' \Rightarrow A, \Delta'; A, \Gamma' \Rightarrow \Delta' \rrbracket \Longrightarrow \Gamma' \Rightarrow \Delta'$

shows *contraction-admissibility*: $F, F, \Gamma \Rightarrow \Delta \Longrightarrow (F::'a \text{ formula}), \Gamma \Rightarrow \Delta$

<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  $\Rightarrow$  'a formula multiset  $\Rightarrow$  nat  $\Rightarrow$  bool (((-
 $\Rightarrow$  / -)  $\downarrow$  -) [53,53] 53) where
  BotL:  $\perp \in \# \Gamma \Longrightarrow \Gamma \Rightarrow \Delta \downarrow \text{Suc } n$  |
  Ax: Atom  $k \in \# \Gamma \Longrightarrow \text{Atom } k \in \# \Delta \Longrightarrow \Gamma \Rightarrow \Delta \downarrow \text{Suc } n$  |
  NotL:  $\Gamma \Rightarrow F, \Delta \downarrow n \Longrightarrow \text{Not } F, \Gamma \Rightarrow \Delta \downarrow \text{Suc } n$  |
  NotR:  $F, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow \text{Not } F, \Delta \downarrow \text{Suc } n$  |
  AndL:  $F, G, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow \text{And } F G, \Gamma \Rightarrow \Delta \downarrow \text{Suc } n$  |
  AndR:  $[\Gamma \Rightarrow F, \Delta \downarrow n; \Gamma \Rightarrow G, \Delta \downarrow n] \Longrightarrow \Gamma \Rightarrow \text{And } F G, \Delta \downarrow \text{Suc } n$  |
  OrL:  $[F, \Gamma \Rightarrow \Delta \downarrow n; G, \Gamma \Rightarrow \Delta \downarrow n] \Longrightarrow \text{Or } F G, \Gamma \Rightarrow \Delta \downarrow \text{Suc } n$  |
  OrR:  $\Gamma \Rightarrow F, G, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow \text{Or } F G, \Delta \downarrow \text{Suc } n$  |
  ImpL:  $[\Gamma \Rightarrow F, \Delta \downarrow n; G, \Gamma \Rightarrow \Delta \downarrow n] \Longrightarrow \text{Imp } F G, \Gamma \Rightarrow \Delta \downarrow \text{Suc } n$  |
  ImpR:  $F, \Gamma \Rightarrow G, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow \text{Imp } F G, \Delta \downarrow \text{Suc } n$ 

```

lemma

```

shows BotL-canonical'[intro!]:  $\perp, \Gamma \Rightarrow \Delta \downarrow \text{Suc } n$ 
and Ax-canonical'[intro!]: Atom  $k, \Gamma \Rightarrow \text{Atom } k, \Delta \downarrow \text{Suc } n$ 
<proof>

```

lemma deeper: $\Gamma \Rightarrow \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow \Delta \downarrow n + m$
 <proof>

lemma deeper-suc: $\Gamma \Rightarrow \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow \Delta \downarrow \text{Suc } n$

```

thm deeper[unfolding Suc-eq-plus1[symmetric]]
<proof>

```

The equivalence is obvious.

theorem SC-SCp-eq:

```

fixes  $\Gamma \Delta$  :: 'a formula multiset
shows  $(\exists n. \Gamma \Rightarrow \Delta \downarrow n) \longleftrightarrow \Gamma \Rightarrow \Delta$  (is ?c  $\longleftrightarrow$  ?p)
<proof>

```

lemma *no-0-SC[dest!]*: $\Gamma \Rightarrow \Delta \downarrow 0 \Longrightarrow False$
 ⟨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 *Atom C → Atom D → Atom E*,

Atom k → Atom C ∧ Atom D,

Atom k, {#}

$\Rightarrow \{ \# \text{ Atom E } \# \} \downarrow \text{Suc} (\text{Suc} (\text{Suc} (\text{Suc} (\text{Suc } 0))))$

⟨proof⟩

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

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

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

lemma *OrL-inv'*:

assumes *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 *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 $\Gamma \Rightarrow \text{Imp } F \text{ } G, \Delta \downarrow n$

shows $F, \Gamma \Rightarrow G, \Delta \downarrow n$

⟨proof⟩

lemma *AndR-inv'*:

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

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

lemma *NotR-inv'*: $\Gamma \Rightarrow \text{Not } F, \Delta \downarrow n \Longrightarrow F, \Gamma \Rightarrow \Delta \downarrow n$
 ⟨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*: $\text{Atom } k, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow \text{Atom } k, \Delta \downarrow m \Longrightarrow \Gamma \Rightarrow \Delta \downarrow n + m$
 ⟨proof⟩

primrec *cut-bound* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{'a formula} \Rightarrow \text{nat}$ **where**

$\text{cut-bound } n \ m \ (\text{Atom } -) = m + n \mid$
 $\text{cut-bound } n \ m \ \text{Bot} = n \mid$
 $\text{cut-bound } n \ m \ (\text{Not } F) = \text{cut-bound } m \ n \ F \mid$
 $\text{cut-bound } n \ m \ (\text{And } F \ G) = \text{cut-bound } n \ (\text{cut-bound } n \ m \ F) \ G \mid$
 $\text{cut-bound } n \ m \ (\text{Or } F \ G) = \text{cut-bound } (\text{cut-bound } n \ m \ F) \ m \ G \mid$
 $\text{cut-bound } n \ m \ (\text{Imp } F \ G) = \text{cut-bound } (\text{cut-bound } m \ n \ F) \ m \ G$

theorem *cut-bound*: $\Gamma \Rightarrow F, \Delta \downarrow n \Longrightarrow F, \Gamma \Rightarrow \Delta \downarrow m \Longrightarrow \Gamma \Rightarrow \Delta \downarrow \text{cut-bound } n \ m \ F$
 ⟨proof⟩

context begin

private primrec *cut-bound'* :: $\text{nat} \Rightarrow \text{'a formula} \Rightarrow \text{nat}$ **where**

$\text{cut-bound}' \ n \ (\text{Atom } -) = 2 * n \mid$
 $\text{cut-bound}' \ n \ \text{Bot} = n \mid$
 $\text{cut-bound}' \ n \ (\text{Not } F) = \text{cut-bound}' \ n \ F \mid$
 $\text{cut-bound}' \ n \ (\text{And } F \ G) = \text{cut-bound}' \ (\text{cut-bound}' \ n \ F) \ G \mid$
 $\text{cut-bound}' \ n \ (\text{Or } F \ G) = \text{cut-bound}' \ (\text{cut-bound}' \ n \ F) \ G \mid$
 $\text{cut-bound}' \ n \ (\text{Imp } F \ G) = \text{cut-bound}' \ (\text{cut-bound}' \ n \ F) \ G$

private lemma *cut-bound'-mono*: $a \leq b \Longrightarrow \text{cut-bound}' \ a \ F \leq \text{cut-bound}' \ b \ F$

⟨proof⟩ **lemma** *cut-bound-mono*: $a \leq c \Longrightarrow b \leq d \Longrightarrow \text{cut-bound } a \ b \ F \leq \text{cut-bound } c \ d \ F$

⟨proof⟩ **lemma** *cut-bound-max*: $\max \ n \ (\text{cut-bound}' \ (\max \ n \ m) \ F) = \text{cut-bound}' \ (\max \ n \ m) \ F$

⟨proof⟩ **lemma** *cut-bound-max'*: $\max \ n \ (\text{cut-bound}' \ n \ F) = \text{cut-bound}' \ n \ F$

$\langle \text{proof} \rangle$ **lemma** *cut-bound-'*: $\text{cut-bound } n \ m \ F \leq \text{cut-bound}' (\max \ n \ m) \ F$
 $\langle \text{proof} \rangle$

primrec *depth* :: 'a formula \Rightarrow nat **where**

$\text{depth } (\text{Atom } -) = 0 \mid$
 $\text{depth } \text{Bot} = 0 \mid$
 $\text{depth } (\text{Not } F) = \text{Suc } (\text{depth } F) \mid$
 $\text{depth } (\text{And } F \ G) = \text{Suc } (\max (\text{depth } F) (\text{depth } G)) \mid$
 $\text{depth } (\text{Or } F \ G) = \text{Suc } (\max (\text{depth } F) (\text{depth } G)) \mid$
 $\text{depth } (\text{Imp } F \ G) = \text{Suc } (\max (\text{depth } F) (\text{depth } G))$

private primrec *cbnd* **where**

$\text{cbnd } k \ 0 = 2 * k \mid$
 $\text{cbnd } k \ (\text{Suc } n) = \text{cbnd } (\text{cbnd } k \ n) \ n$

private lemma *cbnd-grow*: $(k :: \text{nat}) \leq \text{cbnd } k \ d$

$\langle \text{proof} \rangle$ **lemma** *cbnd-mono*: **assumes** $b \leq d$ **shows** $\text{cbnd } (a :: \text{nat}) \ b \leq \text{cbnd } a \ d$
 $\langle \text{proof} \rangle$ **lemma** *cut-bound'-cbnd*: $\text{cut-bound}' \ n \ F \leq \text{cbnd } n \ (\text{depth } F)$
 $\langle \text{proof} \rangle$

value $\text{map } (\text{cbnd } (0 :: \text{int})) \ [0,1,2,3,4]$

value $\text{map } (\text{cbnd } (1 :: \text{int})) \ [0,1,2,3,4]$

value $\text{map } (\text{cbnd } (2 :: \text{int})) \ [0,1,2,3,4]$

value $\text{map } (\text{cbnd } (3 :: \text{int})) \ [0,1,2,3,4]$

value $[\text{nbe}] \ \text{map } (\text{int } \circ (\lambda n. \ n \ \text{div } 3)) \ \circ \ \text{cut-bound } 3 \ 3 \ \circ (\lambda n. \ ((\lambda F. \ \text{And } F \ F) \ \wedge \wedge \ n) \ (\text{Atom } 0)) \ [0,1,2,3,4,5,6,7]$

value $[\text{nbe}] \ \text{map } (\text{int } \circ (\lambda n. \ n \ \text{div } 3)) \ \circ \ \text{cut-bound}' \ 3 \ \circ (\lambda n. \ ((\lambda F. \ \text{And } F \ F) \ \wedge \wedge \ n) \ (\text{Atom } 0)) \ [0,1,2,3,4]$

value $[\text{nbe}] \ \text{map } (\text{int } \circ (\lambda n. \ n \ \text{div } 3)) \ \circ \ \text{cut-bound } 3 \ 3 \ \circ (\lambda n. \ ((\lambda F. \ \text{Imp } (\text{Or } F \ F) \ (\text{And } F \ F)) \ \wedge \wedge \ n) \ (\text{Atom } 0)) \ [0,1,2]$

value $[\text{nbe}] \ \text{map } (\text{int } \circ (\lambda n. \ n \ \text{div } 3)) \ \circ \ \text{cut-bound}' \ 3 \ \circ (\lambda n. \ ((\lambda F. \ \text{Imp } (\text{Or } F \ F) \ (\text{And } F \ F)) \ \wedge \wedge \ n) \ (\text{Atom } 0)) \ [0,1,2]$

value $[\text{nbe}] \ (\lambda F. \ \text{And } (\text{Or } F \ F) \ (\text{Or } F \ F)) \ \wedge \wedge \ 2$

lemma $n + ((n + m) * 2 \wedge (\text{size } F - \text{Suc } 0) +$

$(n + (n + m + (n + m) * 2 \wedge (\text{size } F - \text{Suc } 0))) * 2 \wedge (\text{size } G - \text{Suc } 0))$

$\leq (n + (m :: \text{nat})) * 2 \wedge (\text{size } F + \text{size } G)$

$\langle \text{proof} \rangle$

lemma *cut-bound* $(n :: \text{nat}) \ m \ F \leq (n + m) * (2 \wedge (\text{size } F - 1) + 1)$

$\langle \text{proof} \rangle$ **lemma** *cbnd-comm*: $\text{cbnd } (l * k :: \text{nat}) \ n = l * \text{cbnd } (k :: \text{nat}) \ n$

$\langle \text{proof} \rangle$ **lemma** *cbnd-closed*: $\text{cbnd } (k :: \text{nat}) \ n = k * 2 \wedge (2 \wedge n)$

$\langle \text{proof} \rangle$

theorem *cut'*: **assumes** $\Gamma \Rightarrow F, \Delta \downarrow n \ F, \Gamma \Rightarrow \Delta \downarrow n$ **shows** $\Gamma \Rightarrow \Delta \downarrow n * 2 \wedge (2 \wedge \text{depth } F)$

$\langle \text{proof} \rangle$

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  $\Rightarrow$  'a formula list  $\Rightarrow$  bool (infix  $\Rightarrow$  30) where
Anfang:  $[\mathcal{D}] \Rightarrow [\mathcal{D}] \mid$ 
FalschA:  $[\perp] \Rightarrow [] \mid$ 
VerduennungA:  $\Gamma \Rightarrow \Theta \Longrightarrow \mathcal{D}\#\Gamma \Rightarrow \Theta \mid$ 
VerduennungS:  $\Gamma \Rightarrow \Theta \Longrightarrow \Gamma \Rightarrow \mathcal{D}\#\Theta \mid$ 
ZusammenziehungA:  $\mathcal{D}\#\mathcal{D}\#\Gamma \Rightarrow \Theta \Longrightarrow \mathcal{D}\#\Gamma \Rightarrow \Theta \mid$ 
ZusammenziehungS:  $\Gamma \Rightarrow \mathcal{D}\#\mathcal{D}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathcal{D}\#\Theta \mid$ 
VertauschungA:  $\Delta@\mathcal{D}\#\mathcal{E}\#\Gamma \Rightarrow \Theta \Longrightarrow \Delta@\mathcal{E}\#\mathcal{D}\#\Gamma \Rightarrow \Theta \mid$ 
VertauschungS:  $\Gamma \Rightarrow \Theta@\mathcal{E}\#\mathcal{D}\#\Lambda \Longrightarrow \Gamma \Rightarrow \Theta@\mathcal{D}\#\mathcal{E}\#\Lambda \mid$ 
Schnitt:  $[\Gamma \Rightarrow \mathcal{D}\#\Theta; \mathcal{D}\#\Delta \Rightarrow \Lambda] \Longrightarrow \Gamma@\Delta \Rightarrow \Theta@\Lambda \mid$ 
UES:  $[\Gamma \Rightarrow \mathcal{A}\#\Theta; \Gamma \Rightarrow \mathcal{B}\#\Theta] \Longrightarrow \Gamma \Rightarrow \mathcal{A}\wedge\mathcal{B}\#\Theta \mid$ 
UEA1:  $\mathcal{A}\#\Gamma \Rightarrow \Theta \Longrightarrow \mathcal{A}\wedge\mathcal{B}\#\Gamma \Rightarrow \Theta \mid$  UEA2:  $\mathcal{B}\#\Gamma \Rightarrow \Theta \Longrightarrow \mathcal{A}\wedge\mathcal{B}\#\Gamma \Rightarrow \Theta \mid$ 
OEA:  $[\mathcal{A}\#\Gamma \Rightarrow \Theta; \mathcal{B}\#\Gamma \Rightarrow \Theta] \Longrightarrow \mathcal{A}\vee\mathcal{B}\#\Gamma \Rightarrow \Theta \mid$ 
OES1:  $\Gamma \Rightarrow \mathcal{A}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathcal{A}\vee\mathcal{B}\#\Theta \mid$  OES2:  $\Gamma \Rightarrow \mathcal{B}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathcal{A}\vee\mathcal{B}\#\Theta \mid$ 
FES:  $\mathcal{A}\#\Gamma \Rightarrow \mathcal{B}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathcal{A}\rightarrow\mathcal{B}\#\Theta \mid$ 
FEA:  $[\Gamma \Rightarrow \mathcal{A}\#\Theta; \mathcal{B}\#\Delta \Rightarrow \Lambda] \Longrightarrow \mathcal{A}\rightarrow\mathcal{B}\#\Gamma@\Delta \Rightarrow \Theta@\Lambda \mid$ 
NES:  $\mathcal{A}\#\Gamma \Rightarrow \Theta \Longrightarrow \Gamma \Rightarrow \neg\mathcal{A}\#\Theta \mid$ 
NEA:  $\Gamma \Rightarrow \mathcal{A}\#\Theta \Longrightarrow \neg\mathcal{A}\#\Gamma \Rightarrow \Theta$ 
```

Nota bene: E here stands for “Einführung”, which is introduction and not elimination.

The rule for \perp 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:  $\mathcal{D}\#\Delta@\Gamma \Rightarrow \Theta \Longrightarrow \Delta@\mathcal{D}\#\Gamma \Rightarrow \Theta$ 
  <proof>
lemma AnfangTauschS:  $\Gamma \Rightarrow \mathcal{D}\#\Delta@\Theta \Longrightarrow \Gamma \Rightarrow \Delta@\mathcal{D}\#\Theta$ 
  <proof>
lemma MittenTauschA:  $\Delta@\mathcal{D}\#\Gamma \Rightarrow \Theta \Longrightarrow \mathcal{D}\#\Delta@\Gamma \Rightarrow \Theta$ 
  <proof>
lemma MittenTauschS:  $\Gamma \Rightarrow \Delta@\mathcal{D}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathcal{D}\#\Delta@\Theta$ 
```

<proof>

lemma *BotLe*: $\perp \in \text{set } \Gamma \implies \Gamma \Rightarrow \Delta$
<proof>

lemma *Axe*: $A \in \text{set } \Gamma \implies A \in \text{set } \Delta \implies \Gamma \Rightarrow \Delta$
<proof>

lemma *VerduennungListeA*: $\Gamma \Rightarrow \Theta \implies \Gamma @ \Gamma \Rightarrow \Theta$
<proof>

lemma *VerduennungListeS*: $\Gamma \Rightarrow \Theta \implies \Gamma \Rightarrow \Theta @ \Theta$
<proof>

lemma *ZusammenziehungListeA*: $\Gamma @ \Gamma \Rightarrow \Theta \implies \Gamma \Rightarrow \Theta$
<proof>

lemma *ZusammenziehungListeS*: $\Gamma \Rightarrow \Theta @ \Theta \implies \Gamma \Rightarrow \Theta$
<proof>

theorem *gentzen-sc-eq*: $\text{mset } \Gamma \Rightarrow \text{mset } \Delta \longleftrightarrow \Gamma \Rightarrow \Delta$ *<proof>*

end

2.1.4 Soundness, Completeness

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

definition *sequent-semantics* :: 'a valuation \Rightarrow 'a formula multiset \Rightarrow 'a formula multiset \Rightarrow bool ((- \models (- \Rightarrow / -)) [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 \Rightarrow 'a formula multiset \Rightarrow bool ((\models (- \Rightarrow / -)) [53,53] 53) **where**

$\models \Gamma \Rightarrow \Delta \equiv \forall A. A \models \Gamma \Rightarrow \Delta$

abbreviation *sequent-nonvalid* :: 'a valuation \Rightarrow 'a formula multiset \Rightarrow 'a formula multiset \Rightarrow bool ((- $\not\models$ (- \Rightarrow / -)) [53, 53,53] 53) **where**

$\mathcal{A} \not\models \Gamma \Rightarrow \Delta \equiv \neg \mathcal{A} \models \Gamma \Rightarrow \Delta$

lemma *sequent-intuitionistic-semantics*: $\models \Gamma \Rightarrow \{\#\delta\# \} \longleftrightarrow \text{set-mset } \Gamma \models \delta$
<proof>

lemma *SC-soundness*: $\Gamma \Rightarrow \Delta \implies \models \Gamma \Rightarrow \Delta$
<proof>

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

function(*sequential*)

$sc :: 'a \text{ formula list} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ formula list} \Rightarrow 'a \text{ list} \Rightarrow ('a \text{ list} \times 'a \text{ list}) \text{ set}$

where

$sc (\perp \# \Gamma) A \Delta B = \{\}$ |
 $sc [] A [] B = (\text{if } \text{set } A \cap \text{set } B = \{\} \text{ then } \{\text{remdups } A, \text{remdups } B\} \text{ else } \{\})$ |
 $sc (\text{Atom } k \# \Gamma) A \Delta B = sc \Gamma (k\#A) \Delta B$ |
 $sc (\text{Not } F \# \Gamma) A \Delta B = sc \Gamma A (F\#\Delta) B$ |
 $sc (\text{And } F G \# \Gamma) A \Delta B = sc (F\#G\#\Gamma) A \Delta B$ |
 $sc (\text{Or } F G \# \Gamma) A \Delta B = sc (F\#\Gamma) A \Delta B \cup sc (G\#\Gamma) A \Delta B$ |
 $sc (\text{Imp } F G \# \Gamma) A \Delta B = sc \Gamma A (F\#\Delta) B \cup sc (G\#\Gamma) A \Delta B$ |
 $sc \Gamma A (\perp\#\Delta) B = sc \Gamma A \Delta B$ |
 $sc \Gamma A (\text{Atom } k \# \Delta) B = sc \Gamma A \Delta (k\#B)$ |
 $sc \Gamma A (\text{Not } F \# \Delta) B = sc (F\#\Gamma) A \Delta B$ |
 $sc \Gamma A (\text{And } F G \# \Delta) B = sc \Gamma A (F\#\Delta) B \cup sc \Gamma A (G\#\Delta) B$ |
 $sc \Gamma A (\text{Or } F G \# \Delta) B = sc \Gamma A (F\#G\#\Delta) B$ |
 $sc \Gamma A (\text{Imp } F G \# \Delta) B = sc (F\#\Gamma) A (G\#\Delta) B$
 ⟨*proof*⟩

definition *list-sequent-cost* $\Gamma \Delta = 2 * \text{sum-list } (\text{map size } (\Gamma @ \Delta)) + \text{length } (\Gamma @ \Delta)$

termination sc ⟨*proof*⟩

lemma $sc [] [] [((\text{Atom } 0 \rightarrow \text{Atom } 1) \rightarrow \text{Atom } 0) \rightarrow \text{Atom } 1] [] = \{([0], [1 :: \text{nat}])\}$

⟨*proof*⟩

lemma *sc-sim*:

fixes $\Gamma \Delta :: 'a \text{ formula list}$ **and** $G D :: 'a \text{ list}$

assumes $sc \Gamma A \Delta B = \{\}$

shows $\text{image-mset Atom } (\text{mset } A) + \text{mset } \Gamma \Rightarrow \text{image-mset Atom } (\text{mset } B) + \text{mset } \Delta$

⟨*proof*⟩

lemma *scc-ce-distinct*:

$(C, E) \in sc \Gamma G \Delta D \Longrightarrow \text{set } C \cap \text{set } E = \{\}$

⟨*proof*⟩

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

theorem *SC-counterexample*:

$(C, D) \in sc \Gamma A \Delta B \Longrightarrow$

$(\lambda a. a \in \text{set } C) \neg \models \text{image-mset Atom } (\text{mset } A) + \text{mset } \Gamma \Rightarrow \text{image-mset Atom } (\text{mset } B) + \text{mset } \Delta$

⟨*proof*⟩

corollary *SC-counterexample'*:

assumes $(C, D) \in sc \Gamma [] \Delta []$

shows $(\lambda k. k \in \text{set } C) \neg \models \text{mset } \Gamma \Rightarrow \text{mset } \Delta$

<proof>

theorem *SC-sound-complete*: $\Gamma \Rightarrow \Delta \longleftrightarrow \models \Gamma \Rightarrow \Delta$
<proof>

theorem $\models \Gamma \Rightarrow \Delta \implies \Gamma \Rightarrow \Delta$
<proof>

end

theory *SC-Depth-Limit*

imports *SC-Sema SC-Depth*

begin

lemma *SC-completeness*: $\models \Gamma \Rightarrow \Delta \implies \Gamma \Rightarrow \Delta \downarrow$ *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*:

$\forall \Gamma'. F \triangleright$ *set-mset* $\Gamma =$ *set-mset* $\Gamma' \longrightarrow P \Gamma' \implies P (F, \Gamma)$

$\forall \Gamma'. F \triangleright G \triangleright$ *set-mset* $\Gamma =$ *set-mset* $\Gamma' \longrightarrow P \Gamma' \implies P (F, G, \Gamma)$ *<proof>*

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

<proof>

end

lemma
fixes $\Gamma \Delta :: 'a :: \text{countable formula multiset}$
shows $\models \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta$
 $\langle \text{proof} \rangle$

end

2.2 Natural Deduction

theory *ND*
imports *Formulas*
begin

inductive *ND* :: $'a \text{ formula set} \Rightarrow 'a \text{ formula} \Rightarrow \text{bool}$ (**infix** \vdash 30) **where**
 $Ax: F \in \Gamma \Longrightarrow \Gamma \vdash F \mid$
 $NotE: \llbracket \Gamma \vdash Not\ F; \Gamma \vdash F \rrbracket \Longrightarrow \Gamma \vdash \perp \mid$
 $NotI: F \triangleright \Gamma \vdash \perp \Longrightarrow \Gamma \vdash Not\ F \mid$
 $CC: Not\ F \triangleright \Gamma \vdash \perp \Longrightarrow \Gamma \vdash F \mid$
 $AndE1: \Gamma \vdash And\ F\ G \Longrightarrow \Gamma \vdash F \mid$
 $AndE2: \Gamma \vdash And\ F\ G \Longrightarrow \Gamma \vdash G \mid$
 $AndI: \llbracket \Gamma \vdash F; \Gamma \vdash G \rrbracket \Longrightarrow \Gamma \vdash And\ F\ G \mid$
 $OrI1: \Gamma \vdash F \Longrightarrow \Gamma \vdash Or\ F\ G \mid$
 $OrI2: \Gamma \vdash G \Longrightarrow \Gamma \vdash Or\ F\ G \mid$
 $OrE: \llbracket \Gamma \vdash Or\ F\ G; F \triangleright \Gamma \vdash H; G \triangleright \Gamma \vdash H \rrbracket \Longrightarrow \Gamma \vdash H \mid$
 $ImpI: F \triangleright \Gamma \vdash G \Longrightarrow \Gamma \vdash Imp\ F\ G \mid$
 $ImpE: \llbracket \Gamma \vdash Imp\ F\ G; \Gamma \vdash F \rrbracket \Longrightarrow \Gamma \vdash G$

lemma *Weaken*: $\llbracket \Gamma \vdash F; \Gamma \subseteq \Gamma' \rrbracket \Longrightarrow \Gamma' \vdash F$
 $\langle \text{proof} \rangle$

lemma *BotE* : $\Gamma \vdash \perp \Longrightarrow \Gamma \vdash F$
 $\langle \text{proof} \rangle$

lemma *Not2E*: $Not(Not\ F) \triangleright \Gamma \vdash F$
 $\langle \text{proof} \rangle$

lemma *Not2I*: $F \triangleright \Gamma \vdash Not(Not\ F)$
 $\langle \text{proof} \rangle$

lemma *Not2IE*: $F \triangleright \Gamma \vdash G \Longrightarrow Not\ (Not\ F) \triangleright \Gamma \vdash G$
 $\langle \text{proof} \rangle$

lemma *NDtrans*: $\Gamma \vdash F \Longrightarrow F \triangleright \Gamma \vdash G \Longrightarrow \Gamma \vdash G$
 $\langle \text{proof} \rangle$

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

$\langle proof \rangle$

lemma *NotSwap*: $Not\ F \triangleright \Gamma \vdash G \implies Not\ G \triangleright \Gamma \vdash F$

$\langle proof \rangle$

lemma *AndR-sim*: $\llbracket Not\ F \triangleright \Gamma \vdash H; Not\ G \triangleright \Gamma \vdash H \rrbracket \implies Not(And\ F\ G) \triangleright \Gamma \vdash H$

$\langle proof \rangle$

lemma *OrL-sim*: $\llbracket F \triangleright \Gamma \vdash H; G \triangleright \Gamma \vdash H \rrbracket \implies F \vee G \triangleright \Gamma \vdash H$

$\langle proof \rangle$

lemma *OrR-sim*: $\llbracket \neg F \triangleright \neg G \triangleright \Gamma \vdash \perp \rrbracket \implies \neg(G \vee F) \triangleright \Gamma \vdash \perp$

$\langle proof \rangle$

lemma *ImpL-sim*: $\llbracket \neg F \triangleright \Gamma \vdash \perp; G \triangleright \Gamma \vdash \perp \rrbracket \implies F \rightarrow G \triangleright \Gamma \vdash \perp$

$\langle proof \rangle$

lemma *ImpR-sim*: $\llbracket \neg G \triangleright F \triangleright \Gamma \vdash \perp \rrbracket \implies \neg(F \rightarrow G) \triangleright \Gamma \vdash \perp$

$\langle proof \rangle$

lemma *ND-lem*: $\{\} \vdash Not\ F \vee F$

$\langle proof \rangle$

lemma *ND-caseDistinction*: $\llbracket F \triangleright \Gamma \vdash H; Not\ F \triangleright \Gamma \vdash H \rrbracket \implies \Gamma \vdash H$

$\langle proof \rangle$

lemma $\llbracket \neg F \triangleright \Gamma \vdash H; G \triangleright \Gamma \vdash H \rrbracket \implies F \rightarrow G \triangleright \Gamma \vdash H$

$\langle proof \rangle$

lemma *ND-deMorganAnd*: $\{\neg(F \wedge G)\} \vdash \neg F \vee \neg G$

$\langle proof \rangle$

lemma *ND-deMorganOr*: $\{\neg(F \vee G)\} \vdash \neg F \wedge \neg G$

$\langle proof \rangle$

lemma *sim-sim*: $F \triangleright \Gamma \vdash H \implies G \triangleright \Gamma \vdash F \implies G \triangleright \Gamma \vdash H$

$\langle proof \rangle$

thm *sim-sim*[**where** $\Gamma = \{\}$, *rotated*, *no-vars*]

lemma *Top-provable*[*simp,intro!*]: $\Gamma \vdash \top \langle proof \rangle$

lemma *NotBot-provable*[*simp,intro!*]: $\Gamma \vdash \neg \perp \langle proof \rangle$

lemma *Top-useless*: $\Gamma \vdash F \implies \Gamma - \{\top\} \vdash F$

$\langle proof \rangle$

lemma *AssmBigAnd*: $set\ G \vdash F \iff \{\} \vdash (\bigwedge G \rightarrow F)$

$\langle proof \rangle$

end

theory *ND-Sound*
imports *ND Sema*
begin

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

lemma *ND-sound*: $\Gamma \vdash F \implies \Gamma \models F$
 ⟨*proof*⟩

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

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

definition *turn-true* $\mathcal{A} F \equiv \text{if } \mathcal{A} \models F \text{ then } F \text{ else } (\text{Not } F)$

lemma *lemma0[simp,intro!]*: $\mathcal{A} \models \text{turn-true } \mathcal{A} F$ ⟨*proof*⟩

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

definition *line-assm* :: 'a valuation \Rightarrow 'a set \Rightarrow 'a formula set **where**
line-assm $\mathcal{A} \equiv \text{op } ' (\lambda k. \text{turn-true } \mathcal{A} (\text{Atom } k))$

definition *line-suitable* :: 'a set \Rightarrow 'a formula \Rightarrow bool **where**
line-suitable $Z F \equiv (\text{atoms } F \subseteq Z)$

lemma *line-suitable-junctors[simp]*:
 $\text{line-suitable } \mathcal{A} (\text{Not } F) = \text{line-suitable } \mathcal{A} F$
 $\text{line-suitable } \mathcal{A} (\text{And } F G) = (\text{line-suitable } \mathcal{A} F \wedge \text{line-suitable } \mathcal{A} G)$
 $\text{line-suitable } \mathcal{A} (\text{Or } F G) = (\text{line-suitable } \mathcal{A} F \wedge \text{line-suitable } \mathcal{A} G)$
 $\text{line-suitable } \mathcal{A} (\text{Imp } F G) = (\text{line-suitable } \mathcal{A} F \wedge \text{line-suitable } \mathcal{A} G)$
 ⟨*proof*⟩

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

lemma *NotD*: $\Gamma \vdash \neg F \implies F \triangleright \Gamma \vdash \perp$ ⟨*proof*⟩

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

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

theorem *ND-complete*:
assumes $\text{taut}: \models F$
shows $\{\} \vdash F$
 $\langle \text{proof} \rangle$

corollary *ND-sound-complete*: $\{\} \vdash F \longleftrightarrow \models F$
 $\langle \text{proof} \rangle$

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

theorem
fixes $\Gamma :: 'a :: \text{countable formula set}$
shows $\Gamma \models F \implies \Gamma \vdash F$
 $\langle \text{proof} \rangle$

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* :: $'a \text{ formula set} \Rightarrow 'a \text{ formula} \Rightarrow \text{bool}$ (**infix** \vdash_H 30) **for** $\Gamma :: 'a \text{ formula set}$ **where**
 $Ax: F \in \Gamma \implies \Gamma \vdash_H F \mid$
 $MP: \Gamma \vdash_H F \implies \Gamma \vdash_H F \rightarrow G \implies \Gamma \vdash_H G$

.

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 $\{F \rightarrow (G \rightarrow F), \dots\}$. Instead, defining it as an inductive set with no induction is a good idea.

inductive-set *AX0* **where**

$F \rightarrow (G \rightarrow F) \in AX0 \mid$
 $(F \rightarrow (G \rightarrow H)) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H)) \in AX0$
inductive-set $AX10$ **where**
 $F \in AX0 \implies F \in AX10 \mid$
 $F \rightarrow (F \vee G) \in AX10 \mid$
 $G \rightarrow (F \vee G) \in AX10 \mid$
 $(F \rightarrow H) \rightarrow ((G \rightarrow H) \rightarrow ((F \vee G) \rightarrow H)) \in AX10 \mid$
 $(F \wedge G) \rightarrow F \in AX10 \mid$
 $(F \wedge G) \rightarrow G \in AX10 \mid$
 $F \rightarrow (G \rightarrow (F \wedge G)) \in AX10 \mid$
 $(F \rightarrow \perp) \rightarrow \neg F \in AX10 \mid$
 $\neg F \rightarrow (F \rightarrow \perp) \in AX10 \mid$
 $(\neg F \rightarrow \perp) \rightarrow F \in AX10$
lemmas $HC\text{-intros}[\text{intro!}] =$
 $AX0.\text{intros}[\text{THEN } HC.\text{intros}(1)]$
 $AX0.\text{intros}[\text{THEN } AX10.\text{intros}(1), \text{THEN } HC.\text{intros}(1)]$
 $AX10.\text{intros}(2-)[\text{THEN } HC.\text{intros}(1)]$

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

inductive-set AXH **where**
 $(F \rightarrow (G \rightarrow F)) \in AXH \mid$
 $(F \rightarrow (F \rightarrow G)) \rightarrow (F \rightarrow G) \in AXH \mid$
 $(F \rightarrow (G \rightarrow H)) \rightarrow (G \rightarrow (F \rightarrow H)) \in AXH \mid$
 $(G \rightarrow H) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H)) \in AXH$

lemma $HC\text{-mono}$: $S \vdash_H F \implies S \subseteq T \implies T \vdash_H F$
 $\langle \text{proof} \rangle$

lemma $AX010$: $AX0 \subseteq AX10$
 $\langle \text{proof} \rangle$

lemma $AX100[\text{simp}]$: $AX0 \cup AX10 = AX10$ $\langle \text{proof} \rangle$

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

lemma $hilbert\text{-folgeaxiome-as-strong-as-AX0}$: $AX0 \cup \Gamma \vdash_H F \iff AXH \cup \Gamma \vdash_H F$
 $\langle \text{proof} \rangle$

lemma $AX0 \vdash_H F \rightarrow F$ $\langle \text{proof} \rangle$

lemma $imp\text{-self}$: $AX0 \vdash_H F \rightarrow F$ $\langle \text{proof} \rangle$

theorem $Deduction\text{-theorem}$: $AX0 \cup \text{insert } F \Gamma \vdash_H G \implies AX0 \cup \Gamma \vdash_H F \rightarrow G$
 $\langle \text{proof} \rangle$

end

end

theory *HC-Compl-Consistency*
imports *Consistency HC*
begin

context begin

private lemma *dt*: $F \triangleright \Gamma \cup AX10 \vdash_H G \implies \Gamma \cup AX10 \vdash_H F \rightarrow G$
 $\langle proof \rangle$

lemma *sim*: $\Gamma \cup AX10 \vdash_H F \implies F \triangleright \Gamma \cup AX10 \vdash_H G \implies \Gamma \cup AX10 \vdash_H G$
 $\langle proof \rangle$

lemma *sim-conj*: $F \triangleright G \triangleright \Gamma \cup AX10 \vdash_H H \implies \Gamma \cup AX10 \vdash_H F \implies \Gamma \cup AX10 \vdash_H G \implies \Gamma \cup AX10 \vdash_H H$
 $\langle proof \rangle$

lemma *sim-disj*: $\llbracket F \triangleright \Gamma \cup AX10 \vdash_H H; G \triangleright \Gamma \cup AX10 \vdash_H H; \Gamma \cup AX10 \vdash_H F \vee G \rrbracket \implies \Gamma \cup AX10 \vdash_H H$

$\langle proof \rangle$ **lemma** *someax*: $\Gamma \cup AX10 \vdash_H F \rightarrow \neg F \rightarrow \perp$
 $\langle proof \rangle$

lemma *lem*: $\Gamma \cup AX10 \vdash_H \neg F \vee F$
 $\langle proof \rangle$

lemma *exchg*: $\Gamma \cup AX10 \vdash_H F \vee G \implies \Gamma \cup AX10 \vdash_H G \vee F$
 $\langle proof \rangle$

lemma *lem2*: $\Gamma \cup AX10 \vdash_H F \vee \neg F$ $\langle proof \rangle$

lemma *imp-sim*: $\Gamma \cup AX10 \vdash_H F \rightarrow G \implies \Gamma \cup AX10 \vdash_H \neg F \vee G$
 $\langle proof \rangle$

lemma *inpcp*: $\Gamma \cup AX10 \vdash_H \perp \implies \Gamma \cup AX10 \vdash_H F$
 $\langle proof \rangle$

lemma *HC-case-distinction*: $\Gamma \cup AX10 \vdash_H F \rightarrow G \implies \Gamma \cup AX10 \vdash_H \neg F \rightarrow G \implies \Gamma \cup AX10 \vdash_H G$
 $\langle proof \rangle$

lemma *nand-sim*: $\Gamma \cup AX10 \vdash_H \neg (F \wedge G) \implies \Gamma \cup AX10 \vdash_H \neg F \vee \neg G$
 $\langle proof \rangle$

lemma *HC-contrapos-nn*:

$\llbracket \Gamma \cup AX10 \vdash_H \neg F; \Gamma \cup AX10 \vdash_H G \rightarrow F \rrbracket \implies \Gamma \cup AX10 \vdash_H \neg G$
 $\langle proof \rangle$

lemma *nor-sim*:

assumes $\Gamma \cup AX10 \vdash_H \neg (F \vee G)$

shows $\Gamma \cup AX10 \vdash_H \neg F \quad \Gamma \cup AX10 \vdash_H \neg G$
 $\langle proof \rangle$

lemma *HC-contrapos-np*:

$[[\Gamma \cup AX10 \vdash_H \neg F; \Gamma \cup AX10 \vdash_H \neg G \rightarrow F]] \implies \Gamma \cup AX10 \vdash_H G$
<proof>

lemma *not-imp*: $\Gamma \cup AX10 \vdash_H \neg F \rightarrow F \rightarrow G$

<proof>

lemma *HC-consistent*: *pcp* $\{\Gamma \mid \Gamma. \neg(\Gamma \cup AX10 \vdash_H \perp)\}$

<proof>

end

corollary *HC-complete*:

fixes $F :: 'a :: \text{countable formula}$

shows $\models F \implies AX10 \vdash_H F$

<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* \Rightarrow *'a literal set* \Rightarrow *bool* (*-* *+* *-* [30] 28)

where

Ass: $C \in S \implies S \vdash C$ |

R: $S \vdash C \implies S \vdash D \implies k^+ \in C \implies k^{-1} \in D \implies S \vdash (C - \{k^+\}) \cup (D - \{k^{-1}\})$

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*: $S \vdash D \implies T \cup S \vdash D$

<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* \longleftrightarrow *?r*)

<proof>

lemma *Resolution-taint-assumptions*: $S \cup T \vdash C \implies \exists R \subseteq D. (op \cup D \text{ ' } S) \cup T \vdash R \cup C$

<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 $\exists S' \subseteq S. \text{Ball } S' \text{ finite} \wedge \text{finite } S' \wedge (S' \vdash R)$

<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 \Rightarrow 'a clause set **where**

res $S =$

$(\bigcup C1 \in S. \bigcup C2 \in S. \bigcup L1 \in C1. \bigcup L2 \in C2.$

$(\text{case } (L1, L2) \text{ of } (Pos\ i, Neg\ j) \Rightarrow \text{if } i=j \text{ then } \{(C1 - \{Pos\ i\}) \cup (C2 - \{Neg\ j\})\} \text{ else } \{$

$| - \Rightarrow \{\}\})$

private definition *ex1* $\equiv \{\{Neg\ (0::int)\}, \{Pos\ 0, Pos\ 1, Neg\ 2\}, \{Pos\ 0, Pos\ 1, Pos\ 2\}, \{Pos\ 0, Neg\ 1\}\}$

value *res* *ex1*

definition *Rwhile* :: 'a clause set \Rightarrow 'a clause set option **where**

Rwhile = *while-option* $(\lambda S. \square \notin S \wedge \neg \text{res } S \subseteq S) (\lambda S. \text{res } S \cup S)$

value [*code*] *Rwhile* *ex1*

lemma $\square \in \text{the } (Rwhile\ ex1)$ *<proof>*

lemma *Rwhile-sound*: **assumes** *Rwhile* $S = \text{Some } S'$

shows $\forall C \in S'. \text{Resolution } S\ C$

<proof>

definition *all-clauses* $S = \{s. s \subseteq \{\text{Pos } k \mid k. k \in \text{atoms-of-cnf } S\} \cup \{\text{Neg } k \mid k. k \in \text{atoms-of-cnf } S\}\}$

lemma *s-sub-all-clauses*: $S \subseteq \text{all-clauses } S$

<proof>

lemma *atoms-res*: $\text{atoms-of-cnf } (\text{res } S) \subseteq \text{atoms-of-cnf } S$

<proof>

lemma *exlitE*: $(\bigwedge x. xe = \text{Pos } x \implies P x) \implies (\bigwedge x. xe = \text{Neg } x \implies \text{False}) \implies \exists x. xe = \text{Pos } x \wedge P x$

<proof>

lemma *res-in-all-clauses*: $\text{res } S \subseteq \text{all-clauses } S$

<proof>

lemma *Res-in-all-clauses*: $\text{res } S \cup S \subseteq \text{all-clauses } S$

<proof>

lemma *all-clauses-Res-inv*: $\text{all-clauses } (\text{res } S \cup S) = \text{all-clauses } S$

<proof>

lemma *all-clauses-finite*: $\text{finite } S \wedge (\forall C \in S. \text{finite } C) \implies \text{finite } (\text{all-clauses } S)$

<proof>

lemma *finite-res*: $\forall C \in S. \text{finite } C \implies \forall C \in \text{res } S. \text{finite } C$

<proof>

lemma *finite T*: $\text{finite } T \implies S \subseteq T \implies \text{card } S < \text{Suc } (\text{card } T)$

<proof>

lemma *finite S*: $\text{finite } S \wedge (\forall C \in S. \text{finite } C) \implies \exists T. \text{Rwhile } S = \text{Some } T$

<proof>

partial-function(*option*) *Res where*

Res S = (let R = res S in if R = S then Some S else Res R)

declare *Res.simps*[*code*]

value [*code*] *Res ex1*

lemma $\square \in \text{the } (\text{Res } \text{ex1})$ *<proof>*

lemma *res*: $C \in \text{res } S \implies S \vdash C$

<proof>

lemma *Res-sound*: $\text{Res } S = \text{Some } S' \implies (\forall C \in S'. S \vdash C)$

<proof>

lemma *Res-terminates*: $\text{finite } S \implies \forall C \in S. \text{finite } C \implies \exists T. \text{Res } S = \text{Some } T$

<proof>

code-pred *Resolution* *<proof>*

print-theorems

end

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

lemma *Resolution-insert*: $S \vdash R \implies \text{cnf-semantic } \mathcal{A} S \implies \text{cnf-semantic } \mathcal{A} \{R\}$
 $\langle \text{proof} \rangle$

lemma $S \vdash R \implies \text{cnf-semantic } \mathcal{A} S \longleftrightarrow \text{cnf-semantic } \mathcal{A} (R \triangleright S)$
 $\langle \text{proof} \rangle$

corollary *Resolution-cnf-sound*: **assumes** $S \vdash \square$ **shows** $\neg \text{cnf-semantic } \mathcal{A} S$
 $\langle \text{proof} \rangle$

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

end

2.4.1 Completeness

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

Completeness proof following Schönig [9].

definition *make-lit* $v a \equiv \text{case } v \text{ of } \text{True} \Rightarrow \text{Pos } a \mid \text{False} \Rightarrow \text{Neg } a$

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

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

lemma *cnf-substitution-lemma*:
 $\text{cnf-semantic } A (\text{restrict-cnf-atom } a v S) = \text{cnf-semantic } (A(a := v)) S$
 $\langle \text{proof} \rangle$

lemma *finite-restrict*: $\text{finite } S \implies \text{finite } (\text{restrict-cnf-atom } a v S)$
 $\langle \text{proof} \rangle$

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*:

($\lambda c. \text{if } \{\text{make-lit } (\neg v) a\} \cup c \in S \text{ then } \{\text{make-lit } (\neg v) a\} \cup c \text{ else } c$) ‘*restrict-cnf-atom*
 $a \vee S \subseteq S$
 ⟨*proof*⟩

lemma *can-cope-with-unrestrict-effects*:

assumes *pr*: $S \vdash \square$
assumes *su*: $S \subseteq T$
shows $\exists R \subseteq \{\text{make-lit } v a\}. (\lambda c. \text{if } c \in n \text{ then } \{\text{make-lit } v a\} \cup c \text{ else } c)$ ‘*T* \vdash
 R
 ⟨*proof*⟩

lemma *unrestrict'*:

fixes $R :: \text{'a clause}$
assumes *rp*: *restrict-cnf-atom* $a \vee S \vdash \square$
shows $\exists R \subseteq \{\text{make-lit } (\neg v) a\}. S \vdash R$
 ⟨*proof*⟩

lemma *Resolution-complete-standalone-finite*:

assumes *ns*: $\forall \mathcal{A}. \neg \text{cnf-semantic } \mathcal{A} S$
assumes *fin*: *finite (atoms-of-cnf S)*
shows $S \vdash \square$
 ⟨*proof*⟩

What you might actually want is $\forall \mathcal{A}. \neg \text{cnf-semantic } \mathcal{A} S \implies S \vdash \square$. 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'*: $(\neg P \implies Q) \implies P \vee Q$ ⟨*proof*⟩

lemma *atomD*: $\text{Atom } k \in S \implies \{\text{Pos } k\} \in \text{UNION } S \text{ cnf Not } (\text{Atom } k) \in S \implies \{\text{Neg } k\} \in \text{UNION } S \text{ cnf}$ ⟨*proof*⟩

lemma *pcp-disj*:

$\llbracket F \vee G \in \Gamma; (\forall xa. (xa = F \vee xa \in \Gamma) \longrightarrow \text{is-cnf } xa) \longrightarrow (\text{cnf } F \cup (\bigcup x \in \Gamma. \text{cnf } x) \vdash \square); (\forall xa. (xa = G \vee xa \in \Gamma) \longrightarrow \text{is-cnf } xa) \longrightarrow (\text{cnf } G \cup (\bigcup x \in \Gamma. \text{cnf } x) \vdash \square); \forall x \in \Gamma. \text{is-cnf } x \rrbracket$
 $\implies (\bigcup x \in \Gamma. \text{cnf } x) \vdash \square$

<proof>

lemma *R-consistent*: $pcp \{ \Gamma | \Gamma. \neg((\forall \gamma \in \Gamma. is\text{-}cnf \ \gamma) \longrightarrow ((\bigcup (cnf \ ' \ \Gamma)) \vdash \square)) \}$
<proof>

theorem *Resolution-complete*:

fixes $F :: 'a :: countable \ formula$

shows $\models F \implies cnf \ (nnf \ (\neg F)) \vdash \square$

<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*: $AX10 \cup set\text{-}mset \ \Gamma \vdash_H F \implies \Gamma \Rightarrow \{ \#F \# \}$
<proof>

end

3.2 SC to ND

theory *SCND*

imports *SC ND*

begin

lemma *SCND*: $\Gamma \Rightarrow \Delta \implies (set\text{-}mset \ \Gamma) \cup Not \ ' \ (set\text{-}mset \ \Delta) \vdash \perp$
<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 $AX0 \cup (?F \triangleright ?\Gamma) \vdash_H ?G \implies AX0 \cup ?\Gamma \vdash_H ?F \rightarrow ?G$ helps with this.

theorem *NDHC*: $\Gamma \vdash F \implies AX10 \cup \Gamma \vdash_H F$
<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: $\neg(Atom\ k), Atom\ k, \Gamma \Rightarrow_n \mid$
BotL: $\perp, \Gamma \Rightarrow_n \mid$
AndL: $F, G, \Gamma \Rightarrow_n \implies F \wedge G, \Gamma \Rightarrow_n \mid$
OrL: $F, \Gamma \Rightarrow_n \implies G, \Gamma \Rightarrow_n \implies F \vee G, \Gamma \Rightarrow_n \mid$
— nnf rules:
NotOrNNF: $\neg F, \neg G, \Gamma \Rightarrow_n \implies \neg(F \vee G), \Gamma \Rightarrow_n \mid$
NotAndNNF: $\neg F, \Gamma \Rightarrow_n \implies \neg G, \Gamma \Rightarrow_n \implies \neg(F \wedge G), \Gamma \Rightarrow_n \mid$
ImpNNF: $\neg F, \Gamma \Rightarrow_n \implies G, \Gamma \Rightarrow_n \implies F \rightarrow G, \Gamma \Rightarrow_n \mid$
NotImpNNF: $F, \neg G, \Gamma \Rightarrow_n \implies \neg(F \rightarrow G), \Gamma \Rightarrow_n \mid$
NotNotNNF: $F, \Gamma \Rightarrow_n \implies \neg(\neg F), \Gamma \Rightarrow_n$
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* $\Gamma \equiv \forall x \in \text{set-mset } \Gamma. \text{is-nnf } x$

lemma $\Gamma \Rightarrow \{\#\} \Longrightarrow \text{is-nnf-mset } \Gamma \Longrightarrow \Gamma \Rightarrow_n$
<proof>

lemma *LSC-to-SC*:

shows $\Gamma \Rightarrow_n \Longrightarrow \Gamma \Rightarrow \{\#\}$
<proof>

lemma *SC-to-LSC*:

assumes $\Gamma \Rightarrow \Delta$
shows $\Gamma + (\text{image-mset Not } \Delta) \Rightarrow_n$
<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*: $\Gamma \Rightarrow_n \Longrightarrow \text{image-mset nnf } \Gamma \Rightarrow_n$
<proof>

lemma *LSC-NNF-back*: $\text{image-mset nnf } \Gamma \Rightarrow_n \Longrightarrow \Gamma \Rightarrow_n$
<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 calculi for too long, and they start staring back: Try imagining that there is a \perp on the right hand side. Also, bear in mind that provability of $\Gamma \Rightarrow_n$ and satisfiability of Γ 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*: $\Gamma \Rightarrow_n \implies F, \Gamma \Rightarrow_n$
 ⟨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 *op* \wedge and *op* \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* $k \vee S, (\neg(\text{Atom } k)) \vee T, \Gamma \Rightarrow_n$
 ⟨proof⟩

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

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

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

lemma *distrib1*: $\llbracket F, \Gamma \Rightarrow_n; \text{image-mset } \text{disj-of-clause } (\text{mset } G) + \Gamma \Rightarrow_n \rrbracket$
 $\implies \text{mset } (\text{map } (\lambda d. F \vee \text{disj-of-clause } d) G) + \Gamma \Rightarrow_n$

<proof>

lemma *mset-concat-map-cons*:

$mset (concat (map (\lambda c. F c \# G c) S)) = mset (map F S) + mset (concat (map G S))$
<proof>

lemma *distrib*:

$image-mset \ disj-of-clause (mset F) + \Gamma \Rightarrow_n \implies$
 $image-mset \ disj-of-clause (mset G) + \Gamma \Rightarrow_n \implies$
 $mset [disj-of-clause c \vee disj-of-clause d. c \leftarrow F, d \leftarrow G] + \Gamma \Rightarrow_n$
<proof>

lemma *LSC-BigAndL*: $mset F + \Gamma \Rightarrow_n \implies \bigwedge F, \Gamma \Rightarrow_n$
<proof>

lemma *LSC-Top-unused*: $\llbracket \Gamma \Rightarrow_n; is-nnf-mset \Gamma \rrbracket \implies \Gamma - \{\#\neg \perp\#\} \Rightarrow_n$
<proof>

lemma *LSC-BigAndL-inv*: $\bigwedge F, \Gamma \Rightarrow_n \implies \forall f \in set F. is-nnf f \implies is-nnf-mset \Gamma \implies mset F + \Gamma \Rightarrow_n$
<proof>

lemma *LSC-reassociate-And*: $\{\#disj-of-clause c \vee disj-of-clause d. (c, d) \in\# C\#\} + \Gamma \Rightarrow_n \implies$
 $is-nnf-mset \Gamma \implies$
 $\{\#disj-of-clause (c \textcircled{a} d). (c, d) \in\# C\#\} + \Gamma \Rightarrow_n$
<proof>

lemma *LSC-cnf*: $\Gamma \Rightarrow_n \implies is-nnf-mset \Gamma \implies image-mset \ cnf-form-of \Gamma \Rightarrow_n$
<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 $R = \square \vee R = C$
<proof>

Proof following Gallier [3].

theorem *CSC-Resolution-pre*: $\Gamma \Rightarrow_n \implies \forall \gamma \in set-mset \Gamma. is-cnf \gamma \implies (\bigcup (cnf \text{ '}$
 $set-mset \Gamma)) \vdash \square$
<proof>

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

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

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

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

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

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

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

lemma *ND-finite-assms*: $\Gamma \vdash F \implies \exists \Gamma'. \Gamma' \subseteq \Gamma \wedge \text{finite } \Gamma' \wedge (\Gamma' \vdash F)$
 ⟨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 Γ' 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 $\Gamma :: 'a :: \text{countable formula set}$
 shows $\Gamma \vdash F \longleftrightarrow \Gamma \models F$ (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* Γ
 shows $\Gamma \vdash F \longleftrightarrow \Gamma \models F$ (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*: $S \models F$
 shows $\bigcup ((\text{cnf} \circ \text{nnf}) ` (\{\neg F\} \cup S)) \vdash \square$
 ⟨proof⟩

corollary *Resolution-complete-single*:

```

assumes  $\models F$ 
shows  $\text{cnf} (\text{nnf} (\neg F)) \vdash \square$ 
 $\langle \text{proof} \rangle$ 

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

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

 $\langle \text{proof} \rangle$ 

end

```

3.8 SC and Implication-only formulas

```

theory MiniSC
imports MiniFormulas SC
begin

```

```

abbreviation is-mini-mset  $\Gamma \equiv \forall F \in \text{set-mset } \Gamma. \text{is-mini-formula } F$ 
lemma to-mini-mset-is:  $\text{is-mini-mset} (\text{image-mset to-mini-formula } \Gamma) \langle \text{proof} \rangle$ 

```

```

lemma SC-full-to-mini:
  defines  $\text{tms} \equiv \text{image-mset to-mini-formula}$ 
  assumes  $\text{op}: \Gamma \Rightarrow \Delta$ 
  shows  $\text{tms } \Gamma \Rightarrow \text{tms } \Delta$ 
 $\langle \text{proof} \rangle$ 

```

```

lemma SC-mini-to-full:
  defines  $\text{tms} \equiv \text{image-mset to-mini-formula}$ 
  assumes  $\text{op}: \text{tms } \Gamma \Rightarrow \text{tms } \Delta$ 
  shows  $\Gamma \Rightarrow \Delta$ 
 $\langle \text{proof} \rangle$ 

```

```

theorem MiniSC-eq:  $\text{image-mset to-mini-formula } \Gamma \Rightarrow \text{image-mset to-mini-formula } \Delta \iff \Gamma \Rightarrow \Delta$ 
 $\langle \text{proof} \rangle$ 

```

end

3.8.1 SC to HC

```

theory MiniSC-HC
imports MiniSC HC

```


begin

inductive-set $AX1$ **where**

$F \in AX0 \implies F \in AX1$ |

$((F \rightarrow \perp) \rightarrow \perp) \rightarrow F \in AX1$

lemma $AX01$: $AX0 \subseteq AX1$ $\langle proof \rangle$

lemma $AX1$ -away: $AX1 \cup \Gamma = AX0 \cup (\Gamma \cup AX1)$ $\langle proof \rangle$

lemma $Deduction1$: $F \triangleright (AX1 \cup \Gamma) \vdash_H \perp \longleftrightarrow (AX1 \cup \Gamma) \vdash_H (F \rightarrow \perp)$ $\langle proof \rangle$

lemma $Deduction2$: $(F \rightarrow \perp) \triangleright (AX1 \cup \Gamma) \vdash_H \perp \longleftrightarrow (AX1 \cup \Gamma) \vdash_H F$ **(is ?l**

$\longleftrightarrow ?r)$

$\langle proof \rangle$

lemma

$\Gamma \Rightarrow \Delta \implies is-mini-mset \Gamma \implies is-mini-mset \Delta \implies$

$(set-mset \Gamma \cup (\lambda F. F \rightarrow \perp)) \text{ ' } set-mset \Delta \cup AX1) \vdash_H \perp$

$\langle proof \rangle$

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:

$(\forall \Gamma \Delta \Gamma' \Delta'. \Gamma + \Gamma' \Rightarrow \Delta + \Delta' \longrightarrow (\exists F :: 'a \text{ formula}. \Gamma \Rightarrow F, \Delta \wedge F, \Gamma' \Rightarrow \Delta' \wedge atoms F \subseteq atoms$ -mset $(\Gamma + \Delta) \wedge atoms F \subseteq atoms$ -mset $(\Gamma' + \Delta'))$

\longleftrightarrow

$(\forall \Gamma \Delta. \Gamma \Rightarrow \Delta \longrightarrow (\exists F :: 'a \text{ formula}. \Gamma \Rightarrow \{\#F\# \} \wedge \{\#F\# \} \Rightarrow \Delta \wedge atoms F \subseteq atoms$ -mset $\Gamma \wedge atoms F \subseteq atoms$ -mset $\Delta))$

$\langle proof \rangle$

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 $(\Gamma' + \Delta')$

$\langle proof \rangle$

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 $patoms\text{-}mset\ S \equiv (UNION\ (set\text{-}mset\ S)\ (\lambda F. fst\ (pn\text{-}atoms\ F)))$

abbreviation $natoms\text{-}mset\ S \equiv (UNION\ (set\text{-}mset\ S)\ (\lambda F. snd\ (pn\text{-}atoms\ F)))$

theorem *SC-Craig-interpolation-pn*:

assumes $\Gamma + \Gamma' \Rightarrow \Delta + \Delta'$

obtains F **where**

$\Gamma \Rightarrow F, \Delta$

$F, \Gamma' \Rightarrow \Delta'$

$fst\ (pn\text{-}atoms\ F) \subseteq (patoms\text{-}mset\ \Gamma \cup natoms\text{-}mset\ \Delta) \cap (natoms\text{-}mset\ \Gamma' \cup patoms\text{-}mset\ \Delta')$

$snd\ (pn\text{-}atoms\ F) \subseteq (natoms\text{-}mset\ \Gamma \cup patoms\text{-}mset\ \Delta) \cap (patoms\text{-}mset\ \Gamma' \cup natoms\text{-}mset\ \Delta')$

<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.