

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	7
1.3	Substitutions	11
1.4	Conjunctive Normal Forms	11
1.4.1	Going back: CNFs to formulas	17
1.4.2	Tseytin transformation	18
1.5	Implication-only formulas	26
1.6	Consistency	27
1.7	Compactness	35
1.8	Craig Interpolation using Semantics	44
2	Proof Systems	47
2.1	Sequent Calculus	47
2.1.1	Contraction	52
2.1.2	Cut	53
2.1.3	Mimicking the original	69
2.1.4	Soundness, Completeness	75
2.2	Natural Deduction	80
2.3	Hilbert Calculus	87
2.4	Resolution	94
2.4.1	Completeness	99

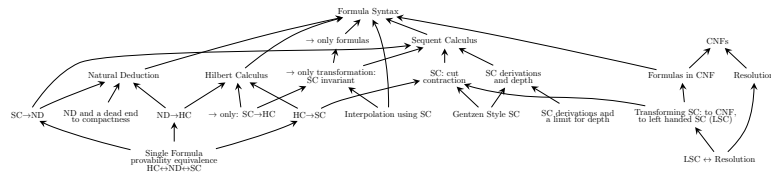


Figure 1: Overview of results considering Proof Transformation

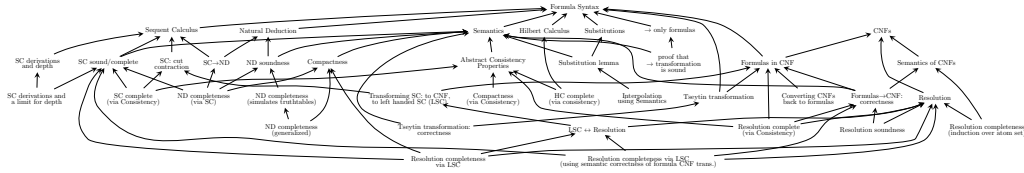


Figure 2: Overview of results considering Semantics

3	Proof Transformation	104
3.1	HC to SC	104
3.2	SC to ND	105
3.3	ND to HC	106
3.4	HC, SC, and ND	107
3.5	Transforming SC proofs into proofs of CNFs	107
3.6	Converting between Resolution and SC proofs	116
3.7	An alternate proof of ND completeness	123
3.8	SC and Implication-only formulas	126
3.8.1	SC to HC	130
3.8.2	Craig Interpolation	131

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
```

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

lemma *atoms-finite*[*simp,intro!*]: *finite (atoms F) by(induction F; simp)*

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)*
by (*induction F*) *auto*

lemma *subsubformulae*: *G \in set (subformulae F) \implies H \in set (subformulae G)*
 \implies *H \in set (subformulae F)*
by(*induction F; force*)

lemma *subformula-atoms*: *G \in set (subformulae F) \implies atoms G \subseteq atoms F*
by(*induction F*) *auto*

lemma *subformulae-self*[*simp,intro!*]: *F \in set (subformulae F)*
by(*induction F; simp*)

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)*

by(*fastforce elim: subsubformulae*)**+**

lemma *length-subformulae*: *length (subformulae F) = size F by(induction F; simp)*

1.1 Derived Connectives

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

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

lemma *top-atoms-simp*[*simp*]: *atoms $\top = \{\}$ unfolding Top-def by simp*

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

lemma *atoms-BigAnd[simp]*: $atoms (\wedge Fs) = \bigcup (atoms \text{ ` set } Fs)$
by(*induction Fs; simp*)

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

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

instance *formula* :: (countable) countable **by** *countable-datatype*

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*: $atoms F = fst (pn-atoms F) \cup snd (pn-atoms F)$
by(*induction F*) (*auto simp add: prod-unions-def split: prod.splits*)

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**

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

```

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

```

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

proof –

have [*simp*]: $\text{Suc} (\text{Suc } 0) \leq \text{size } F + \text{size } G$ **for** $F\ G :: 'a \text{ formula}$ **by** (*cases F*;
cases G; *simp*)

show *?thesis* **by** (*induction F*; *fastforce simp add: Let-def isstop-def Top-def split: formula.splits*)

qed

lemma *simplify-const*: $\text{atoms } F = \{\} \implies \text{isstop} (\text{simplify-consts } F) \vee (\text{simplify-consts } F) = \perp$

by (*induction F*; *fastforce simp add: Let-def isstop-def Top-def split: formula.splits*)

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 $K\ (\text{Suc } n) =$ ($\bigcup_{F \in \text{af}}$

let $\text{af} = \bigcup \text{set} [\text{all-formulas-of-size } K\ m. m \leftarrow [0..<\text{Suc } n]]$ in

$(\bigcup_{G \in \text{af}}$

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 $\{\}$))

$\cup \text{af}$)

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 –

have *rl*: *?r* \implies *?l*

```

proof(induction F arbitrary: n)
  case (Atom x)
    have *: Atom x ∈ all-formulas-of-size K 0 using Atom by simp
    hence **: Atom x ∈ UNION (set [0..<(Suc m)]) (all-formulas-of-size K) for
m
      by (simp; metis atLeastLessThan-iff le-zero-eq not-le)
    thus ?case using Atom by(cases n; simp)
  next
    case Bot
      have *: Bot ∈ all-formulas-of-size K 0 by simp
      hence **: Bot ∈ UNION (set [0..<(Suc m)]) (all-formulas-of-size K) for m
        by (simp; metis atLeastLessThan-iff le-zero-eq not-le)
      then show ?case using Bot by(cases n; simp)
    next
      case (Not F)
        have *: size F ≤ n using Not by simp
        then obtain m where n[simp]: n = Suc m by (metis Suc-diff-1 formula.size-neq
leD neq0-conv)
        with Not have IH: F ∈ all-formulas-of-size K m by simp
        then show ?case using * by(simp add: bexI[where x=F])
    next
      case (And F G)
        with And have *: size F + size G ≤ n by simp
        then obtain m where n[simp]: n = Suc m
          by (metis Suc-diff-1 add-is-0 formula.size-neq le-zero-eq neq0-conv)
        then obtain nF nG where nFG[simp]: size F ≤ nF size G ≤ nG n = nF +
nG
          by (metis * add.assoc nat-le-iff-add order-refl)
        then obtain mF mG where mFG[simp]: nF = Suc mF nG = Suc mG
          by (metis Suc-diff-1 formula.size-neq leD neq0-conv)
        with And have IH: F ∈ all-formulas-of-size K mF G ∈ all-formulas-of-size K
mG
          using nFG by simp+
        let ?af =  $\bigcup \text{set } [all-formulas-of-size K m. m \leftarrow [0..<Suc m]]$ 
        have r: F ∈ all-formulas-of-size K mF  $\implies mF \leq n \implies F \in \bigcup \text{set } (map$ 
(all-formulas-of-size K) [0..<Suc n])
          for F mF n by fastforce
        have af: F ∈ ?af G ∈ ?af using nFG(3) by(intro IH[THEN r]; simp)+
        have m: F ∧ G ∈ (if size F + size G ≤ Suc m then {F ∧ G, F ∨ G, F →
G} else {}) using * by simp
        from IH * show ?case using af by(simp only: n all-formulas-of-size.simps
Let-def, insert m) fast
    next
      case (Or F G) case (Imp F G) — analogous qed
    have lr: ?r if l: ?l
  proof
    have *: F ∈ all-formulas-of-size K x  $\implies F \in all-formulas-of-size K (x + n)$ 
for x n
    by(induction n; simp)

```

```

show size  $F \leq \text{Suc } n$  using  $l$ 
  by(induction  $n$ ; auto split: if-splits) (metis * le-SucI le-eq-less-or-eq le-iff-add)
show atoms  $F \subseteq K$  using  $l$ 
  proof(induction  $n$  arbitrary: F rule: less-induct)
    case (less  $x$ )
    then show ?case proof(cases  $x$ )
      case  $0$  with less show ?thesis by force
    next
      case (Suc  $y$ ) with less show ?thesis
        by(simp only: all-formulas-of-size.simps Let-def) (fastforce simp add: less-Suc-eq split: if-splits)
    qed
  qed
from lr rl show ?thesis proof qed
qed

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-semantics :: 'a valuation  $\Rightarrow$  'a formula  $\Rightarrow$  bool (infix  $\models$  51)
where

```

```

 $\mathcal{A} \models \text{Atom } k = \mathcal{A} \ k \mid$ 
 $- \models \perp = \text{False} \mid$ 
 $\mathcal{A} \models \text{Not } F = (\neg \mathcal{A} \models F) \mid$ 
 $\mathcal{A} \models \text{And } F \ G = (\mathcal{A} \models F \wedge \mathcal{A} \models G) \mid$ 
 $\mathcal{A} \models \text{Or } F \ G = (\mathcal{A} \models F \vee \mathcal{A} \models G) \mid$ 
 $\mathcal{A} \models \text{Imp } F \ G = (\mathcal{A} \models F \longrightarrow \mathcal{A} \models G)$ 

```

```

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

```

```

lemma irrelevant-atom[simp]:  $A \notin \text{atoms } F \implies (\mathcal{A}(A := V)) \models F \longleftrightarrow \mathcal{A} \models F$ 
  by (induction  $F$ ; simp)
lemma relevant-atoms-same-semantics:  $\forall k \in \text{atoms } F. \mathcal{A}_1 \ k = \mathcal{A}_2 \ k \implies \mathcal{A}_1 \models F \longleftrightarrow \mathcal{A}_2 \models F$ 
  by(induction  $F$ ; simp)

```

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-antics-alt where  
formula-antics-alt  $\mathcal{A}$  (Atom  $k$ ) =  $\mathcal{A}$   $k$  |  
formula-antics-alt  $\mathcal{A}$  (Bot) = False |  
formula-antics-alt  $\mathcal{A}$  (Not  $a$ ) = (if formula-antics-alt  $\mathcal{A}$   $a$  then False else  
True) |  
formula-antics-alt  $\mathcal{A}$  (And  $a$   $b$ ) = (if formula-antics-alt  $\mathcal{A}$   $a$  then formula-antics-alt  
 $\mathcal{A}$   $b$  else False) |  
formula-antics-alt  $\mathcal{A}$  (Or  $a$   $b$ ) = (if formula-antics-alt  $\mathcal{A}$   $a$  then True else  
formula-antics-alt  $\mathcal{A}$   $b$ ) |  
formula-antics-alt  $\mathcal{A}$  (Imp  $a$   $b$ ) = (if formula-antics-alt  $\mathcal{A}$   $a$  then formula-antics-alt  
 $\mathcal{A}$   $b$  else True)  
private lemma formula-antics-alt  $\mathcal{A}$   $F \longleftrightarrow \mathcal{A} \models F$   
by(induction  $F$ ; simp)
```

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-antics-tt where  
formula-antics-tt  $\mathcal{A}$  (Atom  $k$ ) =  $\mathcal{A}$   $k$  |  
formula-antics-tt  $\mathcal{A}$  (Bot) = False |  
formula-antics-tt  $\mathcal{A}$  (Not  $a$ ) = (case formula-antics-tt  $\mathcal{A}$   $a$  of True  $\Rightarrow$  False  
| False  $\Rightarrow$  True) |  
formula-antics-tt  $\mathcal{A}$  (And  $a$   $b$ ) = (case (formula-antics-tt  $\mathcal{A}$   $a$ , formula-antics-tt  
 $\mathcal{A}$   $b$ ) of  
  (False, False)  $\Rightarrow$  False  
| (False, True)  $\Rightarrow$  False  
| (True, False)  $\Rightarrow$  False  
| (True, True)  $\Rightarrow$  True) |  
formula-antics-tt  $\mathcal{A}$  (Or  $a$   $b$ ) = (case (formula-antics-tt  $\mathcal{A}$   $a$ , formula-antics-tt  
 $\mathcal{A}$   $b$ ) of  
  (False, False)  $\Rightarrow$  False  
| (False, True)  $\Rightarrow$  True  
| (True, False)  $\Rightarrow$  True  
| (True, True)  $\Rightarrow$  True) |  
formula-antics-tt  $\mathcal{A}$  (Imp  $a$   $b$ ) = (case (formula-antics-tt  $\mathcal{A}$   $a$ , formula-antics-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-antics-tt  $A$   $F$   
by(induction  $F$ ; simp split: prod.splits bool.splits)  
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-semantics[simp,intro!]*: $A \models \top$ **unfolding** *Top-def* **by** *simp*

lemma *BigAnd-semantics[simp]*: $A \models \bigwedge F \longleftrightarrow (\forall f \in \text{set } F. A \models f)$ **by** (*induction F; simp*)

lemma *BigOr-semantics[simp]*: $A \models \bigvee F \longleftrightarrow (\exists f \in \text{set } F. A \models f)$ **by** (*induction F; simp*)

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*: $\Gamma \models \perp \longleftrightarrow \neg \text{sat } \Gamma$

unfolding *sat-def entailment-def* **by** *simp*

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 (*induction F arbitrary: n p*)

case (*Imp F G*)

from *Imp.prem*s **have** *prems*:

$p \notin \text{fst } (\text{pn-atoms } F) \quad p \notin \text{snd } (\text{pn-atoms } G)$

$n \notin \text{snd } (\text{pn-atoms } F) \quad n \notin \text{fst } (\text{pn-atoms } G)$

by (*simp-all add: prod-unions-def split: prod.splits*)

have *IH1*: $M \models F \implies M(n := \text{True}) \models F \quad M \models F \implies M(p := \text{False}) \models F \quad \neg M \models F \implies \neg M(p := \text{True}) \models F \quad \neg M \models F \implies \neg M(n := \text{False}) \models F$

using *Imp.IH(1)[OF prems(3) prems(1)]* **by** *blast+*

have *IH2*: $M \models G \implies M(p := \text{True}) \models G \quad M \models G \implies M(n := \text{False}) \models G \quad \neg M \models G \implies \neg M(n := \text{True}) \models G \quad \neg M \models G \implies \neg M(p := \text{False}) \models G$

using *Imp.IH(2)[OF prems(2) prems(4)]* **by** *blast+*

show *?case* **proof** (*intro conjI; intro impI*)

assume $M \models F \rightarrow G$

then consider $\neg M \models F \mid M \models G$ **by** *auto*

thus $M(p := \text{True}) \models F \rightarrow G \wedge M(n := \text{False}) \models F \rightarrow G$ **using** *IH1(3,4)*

IH2(1,2) **by** *cases simp-all*

next

assume $\neg(M \models F \rightarrow G)$

hence $M \models F \quad \neg M \models G$ **by** *simp-all*

thus $\neg M(n := \text{True}) \models F \rightarrow G \wedge \neg M(p := \text{False}) \models F \rightarrow G$ **using**

IH1(1,2) IH2(3,4) **by** *simp*

qed

next

case (*And F G*)

from *And.prem*s **have** *prems*:

$p \notin \text{snd } (pn\text{-atoms } F)$ $p \notin \text{snd } (pn\text{-atoms } G)$
 $n \notin \text{fst } (pn\text{-atoms } F)$ $n \notin \text{fst } (pn\text{-atoms } G)$
by (*simp-all add: prod-unions-def split: prod.splits*)
have $IH1: M \models F \implies M(p := \text{True}) \models F$ $M \models F \implies M(n := \text{False}) \models F$ $\neg M \models F \implies \neg M(n := \text{True}) \models F$ $\neg M \models F \implies \neg M(p := \text{False}) \models F$
using $\text{And.IH}(1)[OF \text{ prems}(1) \text{ prems}(3)]$ **by** *blast+*
have $IH2: M \models G \implies M(p := \text{True}) \models G$ $M \models G \implies M(n := \text{False}) \models G$ $\neg M \models G \implies \neg M(n := \text{True}) \models G$ $\neg M \models G \implies \neg M(p := \text{False}) \models G$
using $\text{And.IH}(2)[OF \text{ prems}(2) \text{ prems}(4)]$ **by** *blast+*
show $?case$ **proof** (*intro conjI; intro impI*)
assume $\neg M \models F \wedge G$
then consider $\neg M \models F \mid \neg M \models G$ **by** *auto*
thus $\neg M(n := \text{True}) \models F \wedge G \wedge \neg M(p := \text{False}) \models F \wedge G$ **using** $IH1$ $IH2$
by *cases simp-all*
next
assume $M \models F \wedge G$
hence $M \models F$ $M \models G$ **by** *simp-all*
thus $M(p := \text{True}) \models F \wedge G \wedge M(n := \text{False}) \models F \wedge G$ **using** $IH1$ $IH2$ **by** *simp*
qed
next
case (*Or F G*)
from *Or.prems* **have** *prems*:
 $p \notin \text{snd } (pn\text{-atoms } F)$ $p \notin \text{snd } (pn\text{-atoms } G)$
 $n \notin \text{fst } (pn\text{-atoms } F)$ $n \notin \text{fst } (pn\text{-atoms } G)$
by (*simp-all add: prod-unions-def split: prod.splits*)
have $IH1: M \models F \implies M(p := \text{True}) \models F$ $M \models F \implies M(n := \text{False}) \models F$ $\neg M \models F \implies \neg M(n := \text{True}) \models F$ $\neg M \models F \implies \neg M(p := \text{False}) \models F$
using $\text{Or.IH}(1)[OF \text{ prems}(1) \text{ prems}(3)]$ **by** *blast+*
have $IH2: M \models G \implies M(p := \text{True}) \models G$ $M \models G \implies M(n := \text{False}) \models G$ $\neg M \models G \implies \neg M(n := \text{True}) \models G$ $\neg M \models G \implies \neg M(p := \text{False}) \models G$
using $\text{Or.IH}(2)[OF \text{ prems}(2) \text{ prems}(4)]$ **by** *blast+*
show $?case$ **proof** (*intro conjI; intro impI*)
assume $M \models F \vee G$
then consider $M \models F \mid M \models G$ **by** *auto*
thus $M(p := \text{True}) \models F \vee G \wedge M(n := \text{False}) \models F \vee G$ **using** $IH1$ $IH2$ **by** *cases simp-all*
next
assume $\neg M \models F \vee G$
hence $\neg M \models F$ $\neg M \models G$ **by** *simp-all*
thus $\neg M(n := \text{True}) \models F \vee G \wedge \neg M(p := \text{False}) \models F \vee G$ **using** $IH1$ $IH2$
by *simp*
qed
qed *simp-all*

lemma *const-simplifier-correct*: $\mathcal{A} \models \text{simplify-consts } F \iff \mathcal{A} \models F$
by (*induction F*) (*auto simp add: Let-def isstop-def Top-def split: formula.splits*)

end

1.3 Substitutions

theory *Substitution*
imports *Formulas*
begin

primrec *subst* **where**

$subst\ A\ F\ (Atom\ B) = (if\ A = B\ then\ F\ else\ Atom\ B) \mid$
 $subst\ -\ -\ \perp = \perp \mid$
 $subst\ A\ F\ (G \vee H) = (subst\ A\ F\ G \vee subst\ A\ F\ H) \mid$
 $subst\ A\ F\ (G \wedge H) = (subst\ A\ F\ G \wedge subst\ A\ F\ H) \mid$
 $subst\ A\ F\ (G \rightarrow H) = (subst\ A\ F\ G \rightarrow subst\ A\ F\ H) \mid$
 $subst\ A\ F\ (\neg H) = (\neg (subst\ A\ F\ H))$

term *subst*

abbreviation *subst-syntax* $((-[/(-'//)-]) [70,70] 69)$ **where**
 $A[B / C] \equiv subst\ C\ B\ A$

lemma *no-subst[simp]*: $k \notin atoms\ F \implies F[G / k] = F$ **by**(*induction F; simp*)

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

proof(*induction F*)

case (*And F G*) **thus** *?case* **by**(*cases k \in atoms F; force*) **next**
case (*Or F G*) **thus** *?case* **by**(*cases k \in atoms F; force*) **next**
case (*Imp F G*) **thus** *?case* **by**(*cases k \in atoms F; force*) **next**

qed *simp-all*

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

by(*simp add: subst-atoms*)

end

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

lemma *substitution-lemma*: $\mathcal{A} \models F[G / n] \iff \mathcal{A}(n := \mathcal{A} \models G) \models F$ **by**(*induction F; simp*)

end

1.4 Conjunctive Normal Forms

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

datatype *'a literal* = *Pos 'a* $((-^+)$ [1000] 999) \mid *Neg 'a* $((-^{-1})$ [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 = atoms-of-lit ' \bigcup c
lemma atoms-of-cnf-alt: atoms-of-cnf c = $\bigcup ((op \text{ ' atoms-of-lit) ' c)$
unfolding atoms-of-cnf-def **by** blast

lemma atoms-of-cnf-Un: atoms-of-cnf (S \cup T) = atoms-of-cnf S \cup atoms-of-cnf T
unfolding atoms-of-cnf-def **by** auto

term {0⁺}::nat clause
translations
{x} <= 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)
by(pat-completeness) auto

private fun nnf-cost **where**
nnf-cost (Atom -) = 42 |

$nnf\text{-cost } \perp = 42$ |
 $nnf\text{-cost } (Not\ F) = Suc\ (nnf\text{-cost } F)$ |
 $nnf\text{-cost } (And\ F\ G) = Suc\ (nnf\text{-cost } F + nnf\text{-cost } G)$ |
 $nnf\text{-cost } (Or\ F\ G) = Suc\ (nnf\text{-cost } F + nnf\text{-cost } G)$ |
 $nnf\text{-cost } (Imp\ F\ G) = Suc\ (Suc\ (nnf\text{-cost } F + nnf\text{-cost } G))$

termination nnf by (relation measure $(\lambda F. nnf\text{-cost } F)$; simp)

lemma $nnf\ ((Atom\ (k::nat)) \rightarrow (Not\ ((Atom\ l) \vee (Not\ (Atom\ m)))))) = \neg (Atom\ k) \vee (\neg (Atom\ l) \wedge Atom\ m)$
 by code-simp

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

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

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

fun $is\text{-disj}$ where

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

fun $is\text{-cnf}$ where

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

fun $is\text{-nnf}$ where

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

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

by (induction F rule: $nnf.induct$; simp)

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

by (induction F rule: $nnf.induct$; simp)

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

by (induction F rule: $is\text{-nnf.induct}$; simp add: $is\text{-lit-plus-cases}$ split: $formula.splits$; elim $disjE$ $conjE$; simp)

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

by (cases F ; simp)

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

$cnf\ (Atom\ k) = \{\{ k^+ \}\}$ |
 $cnf\ (Not\ (Atom\ k)) = \{\{ k^{-1} \}\}$ |
 $cnf\ \perp = \{\square\}$ |
 $cnf\ (Not\ \perp) = \{\}$ |
 $cnf\ (And\ F\ G) = cnf\ F \cup cnf\ G$ |

$cnf (Or F G) = \{C \cup D \mid C D. C \in (cnf F) \wedge D \in (cnf G)\}$

lemma *cnf-fin*:

assumes *is-nnf F*

shows *finite (cnf F) C ∈ cnf F ⇒ finite C*

proof –

have *finite (cnf F) ∧ (C ∈ cnf F → finite C)* **using** *assms*

by (*induction F arbitrary: C rule: cnf.induct; clarsimp simp add: finite-image-set2*)

thus *finite (cnf F) C ∈ cnf F ⇒ finite C* **by** *simp+*

qed

fun *cnf-lists* :: 'a formula ⇒ 'a literal list list **where**

cnf-lists (Atom k) = [[k⁺]] |

cnf-lists (Not (Atom k)) = [[k⁻¹]] |

cnf-lists ⊥ = [[]] |

cnf-lists (Not ⊥) = [] |

cnf-lists (And F G) = cnf-lists F @ cnf-lists G |

cnf-lists (Or F G) = [f @ g. f ← (cnf-lists F), g ← (cnf-lists G)]

primrec *form-of-lit* **where**

form-of-lit (Pos k) = Atom k |

form-of-lit (Neg k) = ¬(Atom k)

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

definition *disj-of-clause* $c \equiv \bigvee [form-of-lit l. l \leftarrow c]$

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 [] = ⊥

disj-of-clause (F # FF) = form-of-lit F ∨ disj-of-clause FF

by (*simp-all add: disj-of-clause-def*)

lemma *is-cnf-BigAnd: is-cnf (∧ls) ↔ (∀ l ∈ set ls. is-cnf l)*

by (*induction ls; simp*)

private lemma *BigOr-is-not-cnf'*: *is-cnf (∨ls) ⇒ (∀ l ∈ set ls. is-lit-plus l)*

proof (*induction ls*)

case (*Cons l ls*)

from *Cons.prem*s **have** *is-cnf (∨ ls)*

by (*metis BigOr.simps is-cnf.simps(3,5) is-disj.simps(1) list.exhaust*)

thus *?case* **using** *Cons* **by** *simp*

qed *simp*

private lemma *BigOr-is-not-cnf'*: *(∀ l ∈ set ls. is-lit-plus l) ⇒ is-cnf (∨ls)*

by (*induction ls; simp*) (*metis BigOr.simps(1, 2) formula.distinct(25) is-cnf.elims(2) is-cnf.simps(3) list.exhaust*)

lemma *BigOr-is-not-cnf*: $is\text{-}cnf (\bigvee ls) \longleftrightarrow (\forall l \in set\ ls. is\text{-}lit\text{-}plus\ l)$
using *BigOr-is-not-cnf'* *BigOr-is-not-cnf''* **by** *blast*

lemma *is-nnf-BigAnd[simp]*: $is\text{-}nnf (\bigwedge ls) \longleftrightarrow (\forall l \in set\ ls. is\text{-}nnf\ l)$
by (*induction* *ls*; *simp*)

lemma *is-nnf-BigOr[simp]*: $is\text{-}nnf (\bigvee ls) \longleftrightarrow (\forall l \in set\ ls. is\text{-}nnf\ l)$
by (*induction* *ls*; *simp*)

lemma *form-of-lit-is-nnf[simp,intro!]*: $is\text{-}nnf (form\text{-}of\text{-}lit\ x)$
by (*cases* *x*; *simp*)

lemma *form-of-lit-is-lit[simp,intro!]*: $is\text{-}lit\text{-}plus (form\text{-}of\text{-}lit\ x)$
by (*cases* *x*; *simp*)

lemma *disj-of-clause-is-nnf[simp,intro!]*: $is\text{-}nnf (disj\text{-}of\text{-}clause\ F)$
unfolding *disj-of-clause-def* **by** *simp*

lemma *cnf-form-of-is*: $is\text{-}nnf\ F \implies is\text{-}cnf (cnf\text{-}form\text{-}of\ F)$
by (*cases* *F*) (*auto* *simp*: *cnf-form-of-defs* *is-cnf-BigAnd* *BigOr-is-not-cnf*)

lemma *nnf-cnf-form*: $is\text{-}nnf\ F \implies is\text{-}nnf (cnf\text{-}form\text{-}of\ F)$
by (*cases* *F*) (*auto* *simp* *add*: *cnf-form-of-defs*)

lemma *cnf-BigAnd*: $cnf (\bigwedge ls) = (\bigcup x \in set\ ls. cnf\ x)$
by (*induction* *ls*; *simp*)

lemma *cnf-BigOr*: $cnf (\bigvee (x @ y)) = \{f \cup g \mid f g. f \in cnf (\bigvee x) \wedge g \in cnf (\bigvee y)\}$
by (*induction* *x* *arbitrary*: *y*; *simp*) (*metis* (*no-types*, *hide-lams*) *sup.assoc*)

lemma *cnf-cnf*: $is\text{-}nnf\ F \implies cnf (cnf\text{-}form\text{-}of\ F) = cnf\ F$
by (*induction* *F* *rule*: *cnf.induct*;
fastforce *simp* *add*: *cnf-form-of-defs* *cnf-BigAnd* *cnf-BigOr*)

lemma *is-nnf-nnf-id*: $is\text{-}nnf\ F \implies nnf\ F = F$
proof (*induction* *rule*: *is-nnf.induct*)
fix *v* **assume** $is\text{-}nnf (\neg v)$
thus $nnf (\neg v) = \neg v$ **by** (*cases* *v* *rule*: *is-lit-plus.cases*; *simp*)
qed *simp-all*

lemma *disj-of-clause-is*: $is\text{-}disj (disj\text{-}of\text{-}clause\ R)$
by (*induction* *R*; *simp*)

lemma *form-of-cnf-is-nnf*: $is\text{-}nnf (form\text{-}of\text{-}cnf\ R)$
unfolding *form-of-cnf-def* **by** *simp*

lemma *cnf-disj*: $cnf (disj\text{-}of\text{-}clause\ R) = \{set\ R\}$
by (*induction* *R*; *simp* *add*: *form-of-lit-cases* *split*: *literal.splits*)

lemma *cnf-disj-ex*: $is\text{-}disj\ F \implies \exists R. cnf\ F = \{R\} \vee cnf\ F = \{\}$
by (*induction* *F* *rule*: *is-disj.induct*; *clarsimp* *simp*: *is-lit-plus-cases* *split*: *formula.splits*)

```

lemma cnf-form-of-cnf:  $cnf (form-of-cnf S) = set (map set S)$ 
  unfolding form-of-cnf-def by (simp add: cnf-BigAnd cnf-disj) blast

lemma disj-is-nnf:  $is-disj F \implies is-nnf F$ 
  by(induction F rule: is-disj.induct; simp add: is-lit-plus-cases split: formula.splits)

lemma nnf-BigAnd:  $nnf (\bigwedge F) = \bigwedge(map\ nnf\ F)$ 
  by(induction F; simp)

end

end
theory CNF-Sema
imports CNF
begin

primrec lit-semantics :: ('a  $\Rightarrow$  bool)  $\Rightarrow$  'a literal  $\Rightarrow$  bool where
  lit-semantics  $\mathcal{A} (k^+) = \mathcal{A} k \mid$ 
  lit-semantics  $\mathcal{A} (k^{-1}) = (\neg \mathcal{A} k)$ 
case-of-simps lit-semantic-cases: lit-semantic.simps
definition clause-semantics where
  clause-semantics  $\mathcal{A} C \equiv \exists L \in C. lit-semantic\ \mathcal{A}\ L$ 
definition cnf-semantics where
  cnf-semantics  $\mathcal{A} S \equiv \forall C \in S. clause-semantic\ \mathcal{A}\ C$ 

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

lemma nnf-semantics:  $\mathcal{A} \models nnf\ F \iff \mathcal{A} \models F$ 
  by(induction F rule: nnf.induct; simp)

lemma cnf-semantics:  $is-nnf\ F \implies cnf-semantic\ \mathcal{A}\ (cnf\ F) \iff \mathcal{A} \models F$ 
  by(induction F rule: cnf.induct; simp add: cnf-semantic-def clause-semantic-def
ball-Un; metis Un-iff)

lemma cnf-form-semantics:
  fixes  $F :: 'a\ formula$ 
  assumes nnf:  $is-nnf\ F$ 
  shows  $\mathcal{A} \models cnf-form-of\ F \iff \mathcal{A} \models F$ 
proof –
  def cnf-semantic-list  $\equiv \lambda \mathcal{A} S. \forall s \in set\ S. \exists l \in set\ s. lit-semantic\ \mathcal{A}\ (l :: 'a\ literal)$ 
  have tcn:  $cnf\ F = set (map\ set (cnf-lists\ F))$  using nnf

```



```

  by(induction F rule: cnf.induct) (auto simp add: image-UN image-comp comp-def
)
  have  $\mathcal{A} \models F \iff \text{cnf-semantic } \mathcal{A} (\text{cnf } F)$  using cnf-semantic[OF nnf] by
simp
  also have ... = cnf-semantic  $\mathcal{A}$  (set (map set (cnf-lists F))) unfolding tcn ..
  also have ... = cnf-semantic-list  $\mathcal{A}$  (cnf-lists F)
  unfolding cnf-semantic-def clause-semantic-def cnf-semantic-list-def by
fastforce
  also have ... =  $\mathcal{A} \models (\text{cnf-form-of } F)$  using nnf
  by(induction F rule: cnf-lists.induct;
simp add: cnf-semantic-list-def cnf-form-of-defs ball-Un bex-Un)
  finally show ?thesis by simp
qed

```

```

corollary  $\exists G. \mathcal{A} \models F \iff \mathcal{A} \models G \wedge \text{is-cnf } G$ 
using cnf-form-of-is cnf-form-semantic is-nnf-nnf nnf-semantic by blast

```

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 :: (ord) ord
begin

```

definition

```

literal-less-def:  $xs < ys \iff$  (
  if atoms-of-lit  $xs = \text{atoms-of-lit } ys$ 
  then (case  $xs$  of Neg  $\Rightarrow$  (case  $ys$  of Pos  $\Rightarrow$  True |  $\Rightarrow$  False) |  $\Rightarrow$  False)
  else atoms-of-lit  $xs < \text{atoms-of-lit } ys$ )

```

definition

```

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

```

```

instance ..

```

end

```

instance literal :: (linorder) linorder
by standard (auto simp add: literal-less-def literal-le-def split: literal.splits if-splits)

```

definition formula-of-cnf **where**

formula-of-cnf S \equiv *form-of-cnf (sorted-list-of-set (sorted-list-of-set ‘ S))*

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

lemma *simplify-consts* (*formula-of-cnf* ($\{\{Pos\ 0\}\} :: nat\ clause\ set$)) = *Atom 0*
by *code-simp*

lemma *cnf-formula-of-cnf*:
assumes *finite S* $\forall C \in S. *finite C*
shows *cnf (formula-of-cnf S) = S*
using *assms* **by**(*simp add: cnf-BigAnd formula-of-cnf-def form-of-cnf-def cnf-disj*)$

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 (*case p of (k,v) \Rightarrow fun-upd f k v*)

lemma *fold-pair-upd-triv*: *A* \notin *fst ‘ set U* \implies *foldl pair-fun-upd F U A = F A*
by(*induction U arbitrary: F; simp*)
(metis fun-upd-apply pair-fun-upd-def prod.simps(2) surjective-pairing)

lemma *distinct-pair-update-one*: $(k,v) \in set\ U \implies distinct\ (map\ fst\ U) \implies foldl\ pair-fun-upd\ F\ U\ k = v$
by(*induction U arbitrary: F; clarsimp simp add: pair-fun-upd-def fold-pair-upd-triv split: prod.splits*)
(insert fold-pair-upd-triv, fastforce)

lemma *distinct-zipunzip*: *distinct xs* $\implies distinct\ (map\ fst\ (zip\ xs\ ys))$ **by** (*simp add: distinct-conv-nth*)

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

by(*unfold fun-eq-iff; induction U arbitrary: F; clarsimp split: option.splits simp: pair-fun-upd-def rev-image-eqI*)

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

by(*unfold fun-eq-iff comp-def; induction M; simp*)

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

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

unfolding *biimp-def* **by** *auto*

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

by(*simp add: biimp-def*)

locale *freshstuff* =

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

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

begin

primrec *nfresh* **where**

nfresh $S \ 0 = []$ |

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

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

by(*induction n arbitrary: S; simp add: Let-def*)

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

by(*induction n arbitrary: S; auto simp add: Let-def isfresh*)

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

by(*induction n arbitrary: S; simp add: Let-def; meson disjoint-iff-not-equal finite-insert insertI1 nfresh-isfresh*)

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

unfolding *tseytin-assmt-def* **using** *nfresh-distinct* **by** (*simp add: Let-def length-nfresh*)

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

proof –

assume $G \in \text{set } (\text{subformulae } F)$

then have $\exists n. G = \text{subformulae } F \ ! \ n \wedge n < \text{length } (\text{subformulae } F)$

by (*simp add: set-conv-nth*)

then have $\exists a. (a, G) \in \text{set } (\text{zip } (\text{nfresh } (\text{atoms } F) \ (\text{length } (\text{subformulae } F))))$
(*subformulae* F)

by (*metis (no-types) in-set-zip length-nfresh prod.sel(1) prod.sel(2)*)

thus *?thesis* **by**(*simp add: tseytin-assmt-def Let-def (metis fst-conv in-set-conv-nth*)

in-set-zip length-nfresh set-remdups snd-conv)
qed

lemma *tseytin-assmt-new-atoms*: $(k,l) \in \text{set } (tseytin\text{-assmt } F) \implies k \notin \text{atoms } F$
unfolding *tseytin-assmt-def Let-def using nfresh-isfresh by (fastforce dest: set-zip-leftD)*

primrec *tseytin-tran1* **where**

tseytin-tran1 $S \text{ Atom } k = [\text{Atom } k \leftrightarrow S (\text{Atom } k)] \mid$
tseytin-tran1 $S \perp = [\perp \leftrightarrow S \perp] \mid$
tseytin-tran1 $S (\text{Not } F) = [S (\text{Not } F) \leftrightarrow \text{Not } (S F)] @ tseytin\text{-tran1 } S F \mid$
tseytin-tran1 $S (\text{And } F G) = [S (\text{And } F G) \leftrightarrow \text{And } (S F) (S G)] @ tseytin\text{-tran1 } S F @ tseytin\text{-tran1 } S G \mid$
tseytin-tran1 $S (\text{Or } F G) = [S (\text{Or } F G) \leftrightarrow \text{Or } (S F) (S G)] @ tseytin\text{-tran1 } S F @ tseytin\text{-tran1 } S G \mid$
tseytin-tran1 $S (\text{Imp } F G) = [S (\text{Imp } F G) \leftrightarrow \text{Imp } (S F) (S G)] @ tseytin\text{-tran1 } S F @ tseytin\text{-tran1 } S G$

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

definition *tseytin-tran* $F \equiv \bigwedge (\text{let } S = \text{tseytin-toatom } F \text{ in } S F \# tseytin\text{-tran1 } S F)$

lemma *distinct-snd-tseytin-assmt*: *distinct (map snd (tseytin-assmt F))*
unfolding *tseytin-assmt-def by (simp add: Let-def length-nfresh)*

lemma *tseytin-assmt-backlookup*: **assumes** $J \in \text{set } (\text{subformulae } F)$

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

proof –

have 1: *distinct (map snd M) $\implies J \in \text{snd } \text{'set } M \implies (\text{the } (\text{map-of } (\text{map } (\lambda(a, b). (b, a))) M) J), J \in \text{set } M$ for $J M$*

by (*induction M; clarsimp split: prod.splits*)

have 2: $J \in \text{set } (\text{subformulae } F) \implies J \in \text{snd } \text{'set } (tseytin\text{-assmt } F)$ **for** J
using *image-iff tseytin-assmt-has by fastforce*

from 1 [*OF distinct-snd-tseytin-assmt 2, OF assms*] **show** *?thesis* .

qed

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

proof –

have 3: $\text{card } S \leq 2 \implies \text{card } (a \triangleright S) \leq 3$ **for** $a S$

by (*cases finite S; simp add: card-insert-le-m1*)

have 2: $\text{card } S \leq 1 \implies \text{card } (a \triangleright S) \leq 2$ **for** $a S$

by (*cases finite S; simp add: card-insert-le-m1*)

have 1: $\text{card } S \leq 0 \implies \text{card } (a \triangleright S) \leq 1$ **for** $a S$

by (*cases finite S; simp add: card-insert-le-m1*)

have *: $\llbracket G \in \text{set } (tseytin\text{-tran1 } (\text{Atom} \circ S) F); C \in \text{cnf } (\text{nnf } G) \rrbracket \implies \text{card } C \leq 3$ **for** $G C S$

by (*induction F arbitrary: G; simp add: biimp-def; (elim disjE exE conjE | intro 1 2 3 | simp)+*)

```

show ?thesis
  unfolding tseytin-tran-def tseytin-toatom-def Let-def
  by(clarsimp simp add: cnf-BigAnd nnf-BigAnd comp-assoc *)
qed

lemma tseytin-tran-few-clauses: card (cnf (nnf (tseytin-tran F))) ≤ 3 * size F +
1
proof –
  have size Bot = 1 by simp
  have ws: {c ▷ D | D. D = {c1} ∨ D = {c2}} = {{c,c1},{c,c2}} for c1 c2 c by
auto
  have grr: Suc (card S) ≤ c ⇒ card (a ▷ S) ≤ c for a S c
  by(cases finite S; simp add: card-insert-le-m1)
  have *: card (⋃ a∈set (tseytin-tran1 (Atom ◦ S) F). cnf (nnf a)) ≤ 3 * size F
for S
  by(induction F; simp add: biimp-def; ((intro grr card-Un-le[THEN le-trans] |
simp add: ws)+)?)
  show ?thesis
  unfolding tseytin-tran-def tseytin-toatom-def Let-def
  by(clarsimp simp: nnf-BigAnd cnf-BigAnd; intro grr; simp add: comp-assoc *)
qed

lemma tseytin-tran-new-atom-count: card (atoms (tseytin-tran F)) ≤ size F +
card (atoms F)
proof –
  have tseytin-tran1-atoms: H ∈ set (tseytin-tran1 (tseytin-toatom F) G) ⇒ G
∈ set (subformulae F) ⇒
atoms H ⊆ atoms F ∪ (⋃ I ∈ set (subformulae F). atoms (tseytin-toatom F
I)) for G H
  proof(induction G arbitrary: H)
  case (Atom k)
  hence k ∈ atoms F
  by simp (meson formula.set-intros(1) rev-subsetD subformula-atoms)
  with Atom show ?case by simp blast
  next
  case Bot then show ?case by simp blast
  next
  case (Not G)
  show ?case by(insert Not.prem(1,2);
frule subformulas-in-subformulas; simp; elim disjE; (elim Not.IH | force))
  next
  case (And G1 G2)
  show ?case by(insert And.prem(1,2);
frule subformulas-in-subformulas; simp; elim disjE; (elim And.IH; simp |
force))
  next
  case (Or G1 G2)
  show ?case by(insert Or.prem(1,2);
frule subformulas-in-subformulas; simp; elim disjE; (elim Or.IH; simp |

```

```

force))
next
  case (Imp G1 G2)
  show ?case by(insert Imp.prem1,2);
             frule subformulas-in-subformulas; simp; elim disjE; (elim Imp.IH; simp |
force))
qed
have tseytin-tran1-atoms: (⋃ G∈set (tseytin-tran1 (tseytin-toatom F) F). atoms
G) ⊆
  atoms F ∪ (⋃ I∈set (subformulae F). atoms (tseytin-toatom F I))
using tseytin-tran1-atoms[OF - subformulae-self] by blast
have 1: card (atoms (tseytin-tran F)) ≤
  card (atoms (tseytin-toatom F F) ∪ (⋃ x∈set (tseytin-tran1 (tseytin-toatom F)
F). atoms x))
unfolding tseytin-tran-def by(simp add: Let-def tseytin-tran1-atoms)
have 2: atoms (tseytin-toatom F F) ∪ (⋃ x∈set (tseytin-tran1 (tseytin-toatom
F) F). atoms x) ⊆
  (atoms F ∪ (⋃ I∈set (subformulae F). atoms (tseytin-toatom F I)))
using tseytin-tran1-atoms by blast
have twofin: finite (atoms F ∪ (⋃ I∈set (subformulae F). atoms (tseytin-toatom
F I))) by simp
have card-subformulae: card (set (subformulae F)) ≤ size F using length-subformulae
by (metis card-length)
have card-singleton-union: finite S ⇒ card (⋃ x∈S. {f x}) ≤ card S for f S
by(induction S rule: finite-induct; simp add: card-insert-if)
have 3: card (⋃ I∈set (subformulae F). atoms (tseytin-toatom F I)) ≤ size F
unfolding tseytin-toatom-def using le-trans[OF card-singleton-union card-subformulae]
by simp fast
have 4: card (atoms (tseytin-tran F)) ≤ card (atoms F) + card (⋃ f∈set
(subformulae F). atoms (tseytin-toatom F f))
using le-trans[OF 1 card-mono[OF twofin 2]] card-Un-le le-trans by blast
show ?thesis using 3 4 by linarith
qed

```

end

definition freshnat $S \equiv \text{Suc } (0 \triangleright S)$

primrec nfresh-natcode **where**

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

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

interpretation freshnats: freshstuff freshnat **unfolding** freshnat-def **by** standard
(meson Max-ge Suc-n-not-le-n finite-insert insertCI)

lemma [code-unfold]: freshnats.nfresh = nfresh-natcode

proof –

have freshnats.nfresh $S \ n = \text{nfresh-natcode } S \ n$ for $S \ n$ by(induction n arbitrary:
 S ; simp)

thus ?thesis by auto

qed

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 \leftrightarrow *Atom 3* \rightarrow *Atom 4*,
Atom 0 \leftrightarrow *Atom 3*,
Atom 4 \leftrightarrow \neg (*Atom 5*),
Atom 1 \leftrightarrow *Atom 5*

] (**is** ?*l* = ?*r*)

proof –

have *cnf* (*nnf* ?*r*) =
 {{*Pos 2*},
 {*Neg 4*, *Pos 2*}, {*Pos 3*, *Pos 2*}, {*Neg 2*, *Neg 3*, *Pos 4*},
 {*Neg 3*, *Pos 0*}, {*Neg 0*, *Pos 3*},
 {*Pos 5*, *Pos 4*}, {*Neg 4*, *Neg 5*},
 {*Neg 5*, *Pos 1*}, {*Neg 1*, *Pos 5*}} **by** *eval*

have ?*thesis* **by** *eval*

show ?*thesis* **by** *code-simp*

qed

end

theory *Tseytin-Sema*

imports *Sema Tseytin*

begin

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

unfolding *biimp-def* **by** *auto*

locale *freshstuff-sema* = *freshstuff*

begin

definition *tseytin-update* $\mathcal{A} F \equiv$ (*let* *U* = *map* (*apsnd* (*formula-semantic* \mathcal{A}))
 (*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 –

assume $G \in \text{set}(\text{subformulae } F)$

hence *sub*: *atoms* *G* \subseteq *atoms* *F* **by** (*fact* *subformula-atoms*)

have *natoms*: $k \in \text{atoms } F \implies k \notin \text{fst 'set}(\text{tseytin-assmt } F)$ **for** *k l*

using *tseytin-assmt-new-atoms* **by** *force*

have $k \in \text{atoms } F \implies \text{tseytin-update } \mathcal{A} F k = \mathcal{A} k$ **for** *k*

unfolding *tseytin-update-def* *Let-def*

by (*force* *intro!*: *fold-pair-upd-triv* *dest!*: *natoms*)

thus ?*thesis* **using** *relevant-atoms-same-semantic* *sub* **by** (*metis* *subsetCE*)

qed

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(induction F arbitrary: G)
  case (Atom x)
  then show ?case by(simp add: tseytin-update-def tseytin-assmt-def Let-def pair-fun-upd-def)
next
  case Bot
  then show ?case by(simp add: tseytin-update-def tseytin-assmt-def Let-def pair-fun-upd-def)
next
  case (Not F)
  then show ?case
  oops

```

```

lemma tseytin-updates:  $(k, G) \in \text{set } (tseytin-assmt F) \implies tseytin-update \mathcal{A} F k$ 
 $\longleftrightarrow tseytin-update \mathcal{A} F \models G$ 
  apply(subst tseytin-update-def)
  apply(simp add: tseytin-assmt-def Let-def foldl-pair-fun-upd-map-of map-of-map-apsnd
distinct-zipunzip[OF nfresh-distinct[OF atoms-finite]])
  apply(subst tseyting-update-keep-subformula-sema)
  apply(erule in-set-zipE; simp; fail)
  ..

```

```

lemma tseytin-tran1:  $G \in \text{set } (subformulae F) \implies H \in \text{set } (tseytin-tran1 S G)$ 
 $\implies \forall J \in \text{set } (subformulae F). tseytin-update \mathcal{A} F \models J \longleftrightarrow tseytin-update \mathcal{A} F$ 
 $\models (S J) \implies tseytin-update \mathcal{A} F \models H$ 
proof(induction G arbitrary: H)
  case Bot thus ?case by auto
next
  case (Atom k) thus ?case by fastforce
next
  case (Not G)
  consider  $H = S (\neg G) \leftrightarrow \neg (S G) \mid H \in \text{set } (tseytin-tran1 S G)$  using
Not.prem1(2) by auto
  then show ?case proof cases
    case 1 then show ?thesis using Not.prem1(3)
    by (metis Not.prem1(1) biimp-simp formula-anticsimps(3) set-subset-Cons
subformulae.simps(3) subformulae-self subsetCE subsubformulae)
  next
  have  $D: \neg G \in \text{set } (subformulae F) \implies G \in \text{set } (subformulae F)$ 
  by (elim subsubformulae; simp)
  case 2 then show ?thesis using  $D$  Not.IH Not.prem1(3) by blast
  qed
next
  case (And G1 G2)
  have  $el: G1 \in \text{set } (subformulae F) G2 \in \text{set } (subformulae F)$  using subsubfor-
mulae And.prem1(1) by fastforce+
  with And.IH And.prem1(3) have  $IH: H \in \text{set } (tseytin-tran1 S G1) \implies tseytin-update$ 
 $\mathcal{A} F \models H$ 
   $H \in \text{set } (tseytin-tran1 S G2) \implies tseytin-update \mathcal{A} F$ 
 $\models H$  for  $H$ 

```



```

    by blast+
    show ?case using And.prem1 IH el by (simp; elim disjE; simp; insert And.prem1)
    formula-semantics.simps(4), blast)
next
  case (Or G1 G2)
  have el: G1 ∈ set (subformulae F) G2 ∈ set (subformulae F) using subsubfor-
  mulae Or.prem1 by fastforce+
  with Or.IH Or.prem3 have IH: H ∈ set (tseytin-tran1 S G1) ⇒ tseytin-update
  A F ⊨ H
  H ∈ set (tseytin-tran1 S G2) ⇒ tseytin-update A F
  ⊨ H for H
  by blast+
  show ?case using Or.prem3(2) IH el by (simp; elim disjE; simp;metis Or.prem1)
  formula-semantics.simps(5))
next
  case (Imp G1 G2)
  have el: G1 ∈ set (subformulae F) G2 ∈ set (subformulae F) using subsubfor-
  mulae Imp.prem1 by fastforce+
  with Imp.IH Imp.prem3 have IH: H ∈ set (tseytin-tran1 S G1) ⇒ tseytin-update
  A F ⊨ H
  H ∈ set (tseytin-tran1 S G2) ⇒ tseytin-update A F
  ⊨ H for H
  by blast+
  show ?case using Imp.prem3(2) IH el by (simp; elim disjE; simp;metis
  Imp.prem1) formula-semantics.simps(6))
qed

```

lemma *all-tran-formulas-validated*: $\forall J \in \text{set}(\text{subformulae } F). \text{tseytin-update } \mathcal{A} F \models J \iff \text{tseytin-update } \mathcal{A} F \models (\text{tseytin-toatom } F J)$
 apply (simp add: tseytin-toatom-def)
 apply (intro ballI)
 apply (subst tseytin-updates)
 apply (erule tseytin-assmt-backlookup)
 ..

lemma *tseytin-tran-equisat*: $\mathcal{A} \models F \iff \text{tseytin-update } \mathcal{A} F \models (\text{tseytin-tran } F)$
 using *all-tran-formulas-validated* *tseytin-tran1* *all-tran-formulas-validated* *tseyting-update-keep-subformula-se*
 by (simp add: tseytin-tran-def Let-def) blast

lemma *tseytin-tran1-orig-connection*: $G \in \text{set}(\text{subformulae } F) \implies (\forall H \in \text{set}(\text{tseytin-tran1}(\text{tseytin-toatom } F) G). \mathcal{A} \models H) \implies \mathcal{A} \models G \iff \mathcal{A} \models \text{tseytin-toatom } F G$
 by (induction G; simp; drule subformulas-in-subformulas; simp)

lemma *tseytin-untran*: $\mathcal{A} \models (\text{tseytin-tran } F) \implies \mathcal{A} \models F$

proof –

have 1: $\llbracket \mathcal{A} \models \text{tseytin-toatom } F F; \mathcal{A} \models F \rrbracket \implies \text{tseytin-update } \mathcal{A} F \models \text{tseytin-toatom } F F$

```

using all-tran-formulas-validated tseyting-update-keep-subformula-sema by blast
let  $?C = \lambda \mathcal{A}. (\forall H \in \text{set } (tseytin-tran1 \ (tseytin-toatom \ F) \ F). \ \mathcal{A} \models H)$ 
have  $?2: ?C \ \mathcal{A} \implies ?C \ (tseytin-update \ \mathcal{A} \ F)$ 
  using all-tran-formulas-validated tseytin-tran1 by blast
assume  $\mathcal{A} \models tseytin-tran \ F$ 
hence  $tseytin-update \ \mathcal{A} \ F \models tseytin-tran \ F$ 
  unfolding tseytin-tran-def
  apply(simp add: Let-def)
  apply(intro conjI)
  apply(elim conjE)
  apply(drule tseytin-tran1-orig-connection[OF subformulae-self])
  apply(clarsimp simp add: tseytin-assmt-distinct foldl-pair-fun-upd-map-of 1
 $2$ )+)
  done
thus  $?thesis$  using tseytin-tran-equisat by blast
qed
lemma tseytin-tran-equiunsatisfiable:  $\models \neg F \longleftrightarrow \models \neg (tseytin-tran \ F)$  (is  $?l \longleftrightarrow ?r$ )
proof(rule iffI; erule contrapos-pp)
  assume  $\neg ?l$ 
  then obtain  $\mathcal{A}$  where  $\mathcal{A} \models F$  by auto
  hence  $tseytin-update \ \mathcal{A} \ F \models (tseytin-tran \ F)$  using tseytin-tran-equisat by simp
  thus  $\neg ?r$  by simp blast
next
  assume  $\neg ?r$ 
  then obtain  $\mathcal{A}$  where  $\mathcal{A} \models tseytin-tran \ F$  by auto
  thus  $?l$  using tseytin-untran by simp blast
qed
end

interpretation freshsemanats: freshstuff-sema freshnat
  by (simp add: freshnats.freshstuff-axioms freshstuff-sema-def)
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)

by(*induction* F ; *simp*)

lemma *mini-to-mini*: *is-mini-formula* $F \implies$ *to-mini-formula* $F = F$

by(*induction* F ; *simp*)

corollary *mini-mini*[*simp*]: *to-mini-formula* (*to-mini-formula* F) = *to-mini-formula* F

using *mini-to-mini*[*OF to-mini-is-mini*].

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

by(*induction* F ; *simp*)

(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 $A \models F \iff A \models$ *to-mini-formula* F

by(*induction* F) *auto*

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

unfolding *Hintikka-def* **by** *simp*

theorem *Hintikkas-lemma*:

assumes H : *Hintikka* S

shows *sat* S

proof –

from H [*unfolded Hintikka-def*]

have H' : $\perp \notin S$

$\text{Atom } k \in S \implies \neg (\text{Atom } k) \in S \implies \text{False}$

$F \wedge G \in S \implies F \in S \wedge G \in S$

$F \vee G \in S \implies F \in S \vee G \in S$

$F \rightarrow G \in S \implies \neg F \in S \vee G \in S$

$\neg (\neg F) \in S \implies F \in S$

$\neg (F \wedge G) \in S \implies \neg F \in S \vee \neg G \in S$

$\neg (F \vee G) \in S \implies \neg F \in S \wedge \neg G \in S$

$\neg (F \rightarrow G) \in S \implies F \in S \wedge \neg G \in S$

for $k F G$ **by** *blast+*

let $?M = \lambda k. \text{Atom } k \in S$

have $(F \in S \longrightarrow (?M \models F)) \wedge (\neg F \in S \longrightarrow (\neg (?M \models F)))$ **for** F

by(*induction* F) (*auto simp: H'(1) dest!: H'(2-)*)

thus *thesis* **unfolding** *sat-def* **by** *blast*

qed

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

$\perp \notin S$

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

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

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

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

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

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

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

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

$)$

lemma *pcp* {} *pcp* {{{}} *pcp* {{Atom 0}} **by** (*simp add: pcp-def*)+
lemma *pcp* {{(¬ (Atom 1)) → Atom 2},
 {(¬ (Atom 1)) → Atom 2}, ¬(¬ (Atom 1))},
 {(¬ (Atom 1)) → Atom 2}, ¬(¬ (Atom 1)), Atom 1}} **by** (*auto simp add:*
pcp-def)

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

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

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

lemma *Con* (Not (Not F)) F F *Dis* (Not (Not F)) F F **by**(*intro Con.intros*
Dis.intros)+

lemma *con-dis-simps*:

Con a1 a2 a3 = (a1 = a2 ∧ a3 ∨ (∃ F G. a1 = ¬ (F ∨ G) ∧ a2 = ¬ F ∧ a3 =
 = ¬ G) ∨ (∃ G. a1 = ¬ (a2 → G) ∧ a3 = ¬ G) ∨ a1 = ¬ (¬ a2) ∧ a3 = a2)
Dis a1 a2 a3 = (a1 = a2 ∨ a3 ∨ (∃ F G. a1 = F → G ∧ a2 = ¬ F ∧ a3 =
 = G) ∨ (∃ F G. a1 = ¬ (F ∧ G) ∧ a2 = ¬ F ∧ a3 = ¬ G) ∨ a1 = ¬ (¬ a2) ∧
 a3 = a2)
by(*simp-all add: Con.simps Dis.simps*)

lemma *Hintikka-alt*: *Hintikka* S = (
 ⊥ ∉ S
 ∧ (∀ k. Atom k ∈ S → ¬ (Atom k) ∈ S → False)
 ∧ (∀ F G H. Con F G H → F ∈ S → G ∈ S ∧ H ∈ S)
 ∧ (∀ F G H. Dis F G H → F ∈ S → G ∈ S ∨ H ∈ S)
)
apply(*simp add: Hintikka-def con-dis-simps*)
apply(*rule iffI*)

subgoal **by** *blast*
subgoal **by** *safe metis*+
done

lemma *pcp-alt*: *pcp* C = (∀ S ∈ C.

```

  ⊥ ∉ S
  ∧ (∀ k. Atom k ∈ S → ¬ (Atom k) ∈ S → False)
  ∧ (∀ F G H. Con F G H → F ∈ S → G ▷ H ▷ S ∈ C)
  ∧ (∀ F G H. Dis F G H → F ∈ S → G ▷ S ∈ C ∨ H ▷ S ∈ C)
)
  apply(simp add: pcp-def con-dis-simps)
  apply(rule iffI; unfold Ball-def; elim all-forward)
  by (auto simp add: insert-absorb split: formula.splits)

definition subset-closed C ≡ (∀ S ∈ C. ∀ s ⊆ S. s ∈ C)
definition finite-character C ≡ (∀ S. S ∈ C ↔ (∀ s ⊆ S. finite s → s ∈ C))

lemma ex1: pcp C ⇒ ∃ C'. C ⊆ C' ∧ pcp C' ∧ subset-closed C'
proof(intro exI[of - {s . ∃ S ∈ C. s ⊆ S}] conjI)
  let ?E = {s. ∃ S ∈ C. s ⊆ S}
  show C ⊆ ?E by blast
  show subset-closed ?E unfolding subset-closed-def by blast
  assume C: ⟨pcp C⟩
  show pcp ?E using C unfolding pcp-alt
  by (intro ballI conjI; simp; meson insertI1 rev-subsetD subset-insertI subset-insertI2)
qed

lemma sallI: (∧ s. s ⊆ S ⇒ P s) ⇒ ∀ s ⊆ S. P s by simp

lemma ex2:
  assumes fc: finite-character C
  shows subset-closed C
  unfolding subset-closed-def
proof (intro ballI sallI)
  fix s S
  assume e: ⟨S ∈ C⟩ and s: ⟨s ⊆ S⟩
  hence *: t ⊆ s ⇒ t ⊆ S for t by simp
  from fc have t ⊆ S ⇒ finite t ⇒ t ∈ C for t unfolding finite-character-def
using e by blast
  hence t ⊆ s ⇒ finite t ⇒ t ∈ C for t using * by simp
  with fc show ⟨s ∈ C⟩ unfolding finite-character-def by blast
qed

lemma
  assumes C: pcp C
  assumes S: subset-closed C
  shows ex3: ∃ C'. C ⊆ C' ∧ pcp C' ∧ finite-character C'
proof(intro exI[of - C ∪ {S. ∀ s ⊆ S. finite s → s ∈ C}] conjI)
  let ?E = {S. ∀ s ⊆ S. finite s → s ∈ C}
  show C ⊆ C ∪ ?E by blast
  from S show finite-character (C ∪ ?E) unfolding finite-character-def subset-closed-def
by blast
  note C'' = C[unfolded pcp-alt, THEN bspec]

```

have $CON: G \triangleright H \triangleright S \in C \cup \{S. \forall s \subseteq S. \text{finite } s \longrightarrow s \in C\}$ **if** $si: \bigwedge s. \llbracket s \subseteq S; \text{finite } s \rrbracket \implies s \in C$ **and**
 $un: Con\ F\ G\ H$ **and** $el: F \in S$ **for** $F\ G\ H\ S$ **proof** –
have $k: \forall s \subseteq S. \text{finite } s \longrightarrow F \in s \longrightarrow G \triangleright H \triangleright s \in C$
using $si\ un\ C''$ **by** $simp$
have $G \triangleright H \triangleright S \in ?E$
unfolding $mem\text{-}Collect\text{-}eq\ Un\text{-}iff$ **proof** $safe$
fix s
assume $s \subseteq G \triangleright H \triangleright S$ **and** $f: \text{finite } s$
hence $F \triangleright (s - \{G, H\}) \subseteq S$ **using** el **by** $blast$
with $k\ f$ **have** $G \triangleright H \triangleright F \triangleright (s - \{G, H\}) \in C$ **by** $simp$
hence $F \triangleright G \triangleright H \triangleright s \in C$ **using** $insert\text{-}absorb$ **by** $fastforce$
thus $s \in C$ **using** S **unfolding** $subset\text{-}closed\text{-}def$ **by** $fast$
qed
thus $G \triangleright H \triangleright S \in C \cup ?E$ **by** $simp$
qed
have $DIS: G \triangleright S \in C \cup \{S. \forall s \subseteq S. \text{finite } s \longrightarrow s \in C\} \vee H \triangleright S \in C \cup \{S. \forall s \subseteq S. \text{finite } s \longrightarrow s \in C\}$
if $si: \bigwedge s. s \subseteq S \implies \text{finite } s \implies s \in C$ **and** $un: Dis\ F\ G\ H$ **and** $el: F \in S$
for $F\ G\ H\ S$ **proof** –
have $l: \exists I \in \{G, H\}. I \triangleright s1 \in C \wedge I \triangleright s2 \in C$
if $s1 \subseteq S$ $\text{finite } s1$ $F \in s1$
 $s2 \subseteq S$ $\text{finite } s2$ $F \in s2$ **for** $s1\ s2$
proof –
let $?s = s1 \cup s2$
have $?s \subseteq S$ $\text{finite } ?s$ **using** $that$ **by** $simp\text{-}all$
with si **have** $?s \in C$ **by** $simp$
moreover **have** $F \in ?s$ **using** $that$ **by** $simp$
ultimately **have** $\exists I \in \{G, H\}. I \triangleright ?s \in C$
using $C''\ un$ **by** $simp$
thus $\exists I \in \{G, H\}. I \triangleright s1 \in C \wedge I \triangleright s2 \in C$
by $(meson\ S[\text{unfolded } subset\text{-}closed\text{-}def, THEN\ bspec])\ insert\text{-}mono\ sup.\text{cobounded}2\ sup\text{-}ge1)$
qed
have $m: \llbracket s1 \subseteq S; \text{finite } s1; F \in s1; G \triangleright s1 \notin C; s2 \subseteq S; \text{finite } s2; F \in s2; H \triangleright s2 \notin C \rrbracket \implies False$ **for** $s1\ s2$
using l **by** $blast$
have $False$ **if** $s1 \subseteq S$ $\text{finite } s1$ $G \triangleright s1 \notin C$ $s2 \subseteq S$ $\text{finite } s2$ $H \triangleright s2 \notin C$ **for** $s1\ s2$
proof –
have $*$: $F \triangleright s1 \subseteq S$ $\text{finite } (F \triangleright s1)$ $F \in F \triangleright s1$ **if** $s1 \subseteq S$ $\text{finite } s1$ **for** $s1$
using $that\ el$ **by** $simp\text{-}all$
have $G \triangleright F \triangleright s1 \notin C$ $H \triangleright F \triangleright s2 \notin C$
by $(meson\ S\ insert\text{-}mono\ subset\text{-}closed\text{-}def\ subset\text{-}insertI\ that(3,6))+$
from $m[OF\ *[OF\ that(1-2)]\ this(1)\ *[OF\ that(4-5)]\ this(2)]$
show $False$.
qed
hence $G \triangleright S \in ?E \vee H \triangleright S \in ?E$

unfolding *mem-Collect-eq Un-iff*
by (*metis (no-types, lifting) finite-Diff insert-Diff si subset-insert-iff*)
thus $G \triangleright S \in C \cup ?E \vee H \triangleright S \in C \cup ?E$ **by** *blast*
qed

have CON' : $\bigwedge f2\ g2\ h2\ F2\ G2\ S2. [\bigwedge s. [s \in C; h2\ F2\ G2 \in s]] \implies f2\ F2 \triangleright s \in C \vee g2\ G2 \triangleright s \in C;$
 $\forall s \subseteq S2. \text{finite } s \longrightarrow s \in C; h2\ F2\ G2 \in S2; \text{False}]$
 $\implies f2\ F2 \triangleright S2 \in C \cup \{S. \forall s \subseteq S. \text{finite } s \longrightarrow s \in C\} \vee g2\ G2 \triangleright S2 \in C \cup \{S. \forall s \subseteq S. \text{finite } s \longrightarrow s \in C\}$
by *fast*

show $pcp (C \cup ?E)$ **unfolding** *pcp-alt*
apply(*intro ballI conjI; elim UnE; (unfold mem-Collect-eq)?*)
subgoal using C'' **by** *blast*
subgoal using C'' **by** *blast*
subgoal using C'' **by** (*simp; fail*)
subgoal by (*meson C'' empty-subsetI finite.emptyI finite-insert insert-subset subset-insertI*)
subgoal using C'' **by** *simp*
subgoal using CON **by** *simp*
subgoal using C'' **by** *blast*
subgoal using DIS **by** *simp*
done
qed

primrec *pcp-seq* **where**
 $pcp-seq\ C\ S\ 0 = S$ |
 $pcp-seq\ C\ S\ (Suc\ n) = ($
 $\text{let } Sn = pcp-seq\ C\ S\ n; Sn1 = \text{from-nat } n \triangleright Sn \text{ in}$
 $\text{if } Sn1 \in C \text{ then } Sn1 \text{ else } Sn$
 $)$

lemma *pcp-seq-in*: $pcp\ C \implies S \in C \implies pcp-seq\ C\ S\ n \in C$
proof(*induction n*)
case ($Suc\ n$)
hence $pcp-seq\ C\ S\ n \in C$ **by** *simp*
thus *?case* **by**(*simp add: Let-def*)
qed *simp*

lemma *pcp-seq-mono*: $n \leq m \implies pcp-seq\ C\ S\ n \subseteq pcp-seq\ C\ S\ m$
proof(*induction m*)
case ($Suc\ m$)
thus *?case* **by**(*cases n = Suc m; simp add: Let-def; blast*)
qed *simp*

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

proof(*induction m*)
case (*Suc m*)
have $\{f\ n \mid n. n \leq \text{Suc } m\} = f\ (\text{Suc } m) \triangleright \{f\ n \mid n. n \leq m\}$ **for** f **using** *le-Suc-eq*
by *auto*
hence $\{pcp\text{-seq } C\ S\ n \mid n. n \leq \text{Suc } m\} = pcp\text{-seq } C\ S\ (\text{Suc } m) \triangleright \{pcp\text{-seq } C\ S\ n \mid n. n \leq m\}$.
hence $\bigcup \{pcp\text{-seq } C\ S\ n \mid n. n \leq \text{Suc } m\} = \bigcup \{pcp\text{-seq } C\ S\ n \mid n. n \leq m\} \cup pcp\text{-seq } C\ S\ (\text{Suc } m)$ **by** *auto*
thus *?case* **using** *Suc pcp-seq-mono* **by** *blast*
qed *simp*

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

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

oops

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

lemma *pcp-seq-sub*: $pcp\text{-seq } C\ S\ n \subseteq pcp\text{-lim } C\ S$
unfolding *pcp-lim-def* **by**(*induction n; blast*)

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

lemma *pcp-lim-in*:
assumes $c: pcp\ C$
and $el: S \in C$
and $sc: \text{subset-closed } C$
and $fc: \text{finite-character } C$
shows $pcp\text{-lim } C\ S \in C$ (**is** $?cl \in C$)

proof –

from *pcp-seq-in*[*OF c el, THEN allI*] **have** $\forall n. pcp\text{-seq } C\ S\ n \in C$.
hence $\forall m. \bigcup \{pcp\text{-seq } C\ S\ n \mid n. n \leq m\} \in C$ **unfolding** *pcp-seq-UN* .

have $\forall s \subseteq ?cl. \text{finite } s \longrightarrow s \in C$

proof *safe*

fix $s :: 'a \text{ formula set}$

have $pcp\text{-seq } C\ S\ (\text{Suc } (\text{Max } (\text{to-nat } 's))) \subseteq pcp\text{-lim } C\ S$ **using** *pcp-seq-sub*

by *blast*

assume $\langle \text{finite } s \rangle \langle s \subseteq pcp\text{-lim } C\ S \rangle$

hence $\exists k. s \subseteq pcp\text{-seq } C\ S\ k$

proof(*induction s rule: finite-induct*)

case (*insert x s*)

hence $\exists k. s \subseteq pcp\text{-seq } C\ S\ k$ **by** *fast*

then *guess k1 ..*

moreover **obtain** $k2$ **where** $x \in pcp\text{-seq } C\ S\ k2$

by (*meson pcp-lim-inserted-at-ex insert.premis insert-subset*)

ultimately **have** $x \triangleright s \subseteq pcp\text{-seq } C\ S\ (\text{max } k1\ k2)$

by (meson pcp-seq-mono dual-order.trans insert-subset max.bounded-iff
 order-refl subsetCE)
 thus ?case by blast
 qed simp
 with pcp-seq-in[OF c el] sc
 show $s \in C$ unfolding subset-closed-def by blast
 qed
 thus ?cl $\in C$ using fc unfolding finite-character-def by blast
 qed

lemma cl-max:

assumes c: pcp C
 assumes sc: subset-closed C
 assumes el: $K \in C$
 assumes su: pcp-lim C S $\subseteq K$
 shows pcp-lim C S = K (is ?e)
 proof (rule ccontr)
 assume $\neg ?e$
 with su have pcp-lim C S $\subset K$ by simp
 then obtain F where $e: F \in K$ and $ne: F \notin \text{pcp-lim } C S$ by blast
 from ne have $F \notin \text{pcp-seq } C S$ (Suc (to-nat F)) using pcp-seq-sub by fast
 hence 1: $F \triangleright \text{pcp-seq } C S$ (to-nat F) $\notin C$ by (simp add: Let-def split: if-splits)
 have $F \triangleright \text{pcp-seq } C S$ (to-nat F) $\subseteq K$ using pcp-seq-sub e su by blast
 hence $F \triangleright \text{pcp-seq } C S$ (to-nat F) $\in C$ using sc unfolding subset-closed-def
 using el by blast
 with 1 show False ..
 qed

lemma cl-max':

assumes c: pcp C
 assumes sc: subset-closed C
 shows $F \triangleright \text{pcp-lim } C S \in C \implies F \in \text{pcp-lim } C S$
 $F \triangleright G \triangleright \text{pcp-lim } C S \in C \implies F \in \text{pcp-lim } C S \wedge G \in \text{pcp-lim } C S$
 using cl-max[OF assms] by blast+

lemma pcp-lim-Hintikka:

assumes c: pcp C
 assumes sc: subset-closed C
 assumes fc: finite-character C
 assumes el: $S \in C$
 shows Hintikka (pcp-lim C S)
 proof –
 let ?cl = pcp-lim C S
 have ?cl $\in C$ using pcp-lim-in[OF c el sc fc] .
 from c[unfolded pcp-alt, THEN bspec, OF this]
 have d: $\perp \notin ?cl$
 $\text{Atom } k \in ?cl \implies \neg (\text{Atom } k) \in ?cl \implies \text{False}$
 $\text{Con } F G H \implies F \in ?cl \implies G \triangleright H \triangleright ?cl \in C$
 $\text{Dis } F G H \implies F \in ?cl \implies G \triangleright ?cl \in C \vee H \triangleright ?cl \in C$

for $k F G H$ **by** *blast+*
have
Con $F G H \implies F \in ?cl \implies G \in ?cl \wedge H \in ?cl$
Dis $F G H \implies F \in ?cl \implies G \in ?cl \vee H \in ?cl$
for $F G H$
by(*auto dest: d(3-) cl-max'[OF c sc]*)
with $d(1,2)$ **show** *?thesis unfolding Hintikka-alt by fast*
qed

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 –
note $[[\text{show-types}]]$
from c **obtain** Ce **where** $C \subseteq Ce$ *pcp Ce subset-closed Ce finite-character Ce*
using $ex1[\text{where } 'a='a]$ $ex2[\text{where } 'a='a]$ $ex3[\text{where } 'a='a]$
by (*meson dual-order.trans ex2*)
have $S \in Ce$ **using** $\langle C \subseteq Ce \rangle el ..$
with *pcp-lim-Hintikka* $\langle \text{pcp } Ce \rangle \langle \text{subset-closed } Ce \rangle \langle \text{finite-character } Ce \rangle$
have *Hintikka* $(\text{pcp-lim } Ce S)$.
with *Hintikkas-lemma* **have** $\text{sat } (\text{pcp-lim } Ce S)$.
moreover **have** $S \subseteq \text{pcp-lim } Ce S$ **using** *pcp-seq.simps(1) pcp-seq-sub by fast*
ultimately show *?thesis unfolding sat-def by fast*
qed

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 (*rule ccontr*)

assume $fs: \text{fin-sat } S$
assume $nfs: \neg (\text{fin-sat } (\text{insert } F S) \vee \text{fin-sat } (\text{insert } (\neg F) S))$
from nfs **obtain** $s1$ **where** $s1: s1 \subseteq \text{insert } F S$ *finite s1 $\neg \text{sat } s1$ unfolding fin-sat-def by blast*
from nfs **obtain** $s2$ **where** $s2: s2 \subseteq \text{insert } (\text{Not } F) S$ *finite s2 $\neg \text{sat } s2$ unfolding fin-sat-def by blast*
let $?u = (s1 - \{F\}) \cup (s2 - \{\text{Not } F\})$
have $?u \subseteq S$ *finite ?u using s1 s2 by auto*
hence $\text{sat } ?u$ **using** fs **unfolding** *fin-sat-def by blast*
then obtain A **where** $A: \forall F \in ?u. A \models F$ **unfolding** *sat-def by blast*
have $A \models F \vee A \models \neg F$ **by** *simp*

hence $\text{sat } s1 \vee \text{sat } s2$ **using** A **unfolding** sat-def **by** $(\text{fastforce intro!}: \text{exI}[\text{where } x=A])$
thus False **using** $s1(\beta) s2(\beta)$ **by** simp
qed

context
begin

lemma fin-sat-antimono : $\text{fin-sat } F \implies G \subseteq F \implies \text{fin-sat } G$ **unfolding** fin-sat-def
by simp
lemmas $\text{fin-sat-insert} = \text{fin-sat-antimono}[\text{OF - subset-insertI}]$

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

private lemma extender-fin-sat : $\text{fin-sat } S \implies \text{fin-sat } (\text{extender } n S)$
proof $(\text{induction } n \text{ arbitrary: } S)$
case $(\text{Suc } n)$
note $mIH = \text{Suc.IH}[\text{OF Suc.prem}]$
show $?case$ **proof** $(\text{cases } \text{fin-sat } (\text{insert } (\text{Not } (\text{from-nat } n)) (\text{extender } n S)))$
case True **thus** $?thesis$ **by** simp
next
case False
hence $\text{fin-sat } (\text{insert } ((\text{from-nat } n)) (\text{extender } n S))$ **using** $mIH \text{fin-sat-extend}$
by auto
thus $?thesis$ **by** $(\text{simp add: Let-def})$
qed
qed simp

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

lemma extended-max : $F \in \text{extended } S \vee \text{Not } F \in \text{extended } S$
proof $-$
obtain n **where** $[\text{simp}]$: $F = \text{from-nat } n$ **by** $(\text{metis from-nat-to-nat})$
have $F \in \text{extender } (\text{Suc } n) S \vee \text{Not } F \in \text{extender } (\text{Suc } n) S$ **by** $(\text{simp add: Let-def})$
thus $?thesis$ **unfolding** extended-def **by** blast
qed

private lemma extender-Sucset : $\text{extender } k S \subseteq \text{extender } (\text{Suc } k) S$ **by** $(\text{force simp add: Let-def})$

private lemma extender-deeper : $F \in \text{extender } k S \implies k \leq l \implies F \in \text{extender } l S$ **using** $\text{extender-Sucset le-Suc-eq}$

by(*induction l*) (*auto simp del: extender.simps*)
private lemma *extender-subset*: $S \subseteq \text{extender } k \ S$
proof –
from *extender-deeper*[*OF - le0*] **have** $F \in \text{extender } 0 \ Sa \implies F \in \text{extender } l \ Sa$
for $Sa \ l \ F$.
thus *?thesis* **by** *auto*
qed

lemma *extended-fin-sat*:
assumes *fin-sat S*
shows *fin-sat (extended S)*
proof –
have *assm*: $\llbracket s \subseteq \text{extender } n \ S; \text{finite } s \rrbracket \implies \text{sat } s \ \text{for } s \ n$
using *extender-fin-sat*[*OF assms*] **unfolding** *fin-sat-def* **by** *presburger*
hence $\text{sat } s$ **if** $su: s \subseteq \bigcup \{ \text{extender } n \ S \mid n. \text{True} \}$ **and** *fin*: *finite s for s* **proof**
–
{ **fix** x **assume** $e: x \in s$
with su **have** $x \in \bigcup \{ \text{extender } n \ S \mid n. \text{True} \}$ **by** *blast*
hence $\exists n. x \in \text{extender } n \ S$ **unfolding** *Union-eq* **by** *blast* }
hence $\forall x \in s. \exists n. x \in \text{extender } n \ S$ **by** *blast*
from *finite-set-choice*[*OF fin this*] **obtain** f **where** $cf: \forall x \in s. x \in \text{extender } (f \ x) \ S ..$
have $\exists k. s \subseteq \bigcup \{ \text{extender } n \ S \mid n. n \leq k \}$ **proof**(*intro exI subsetI*)
fix x **assume** $e: x \in s$
with cf **have** $x \in \text{extender } (f \ x) \ S ..$
hence $x \in \text{extender } (\text{Max } (f \ ' \ s)) \ S$ **by**(*elim extender-deeper; simp add: e fin*)
thus $x \in \bigcup \{ \text{extender } n \ S \mid n. n \leq \text{Max } (f \ ' \ s) \}$ **by** *blast*
qed
moreover **have** $\bigcup \{ \text{extender } n \ S \mid n. n \leq k \} = \text{extender } k \ S$ **for** k **proof**(*induction k*)
case (*Suc k*)
moreover **have** $\bigcup \{ \text{extender } n \ S \mid n. n \leq \text{Suc } k \} = \bigcup \{ \text{extender } n \ S \mid n. n \leq k \} \cup \text{extender } (\text{Suc } k) \ S$
unfolding *Union-eq le-Suc-eq*
using *le-Suc-eq* **by**(*auto simp del: extender.simps*)
ultimately show *?case* **using** *extender-Sucset* **by**(*force simp del: extender.simps*)
qed *simp*
ultimately show $\text{sat } s$ **using** *assm fin* **by** *auto*
qed
thus *?thesis* **unfolding** *extended-def fin-sat-def* **by** *presburger*
qed

lemma *extended-superset*: $S \subseteq \text{extended } S$ **unfolding** *extended-def* **using** *extender.simps(1)* **by** *blast*

lemma *extended-complem*:
assumes *fs: fin-sat S*

```

shows  $F \in \text{extended } S \neq (\text{Not } F \in \text{extended } S)$ 
proof –
  note  $fs = fs[\text{THEN extended-fin-sat}]$ 
  show ?thesis proof(cases  $F \in \text{extended } S$ )
    case False with extended-max show ?thesis by blast
  next
    case True have  $\text{Not } F \notin \text{extended } S$  proof
      assume False:  $\text{Not } F \in \text{extended } S$ 
      with True have  $\{F, \text{Not } F\} \subseteq \text{extended } S$  by blast
      moreover have finite  $\{F, \text{Not } F\}$  by simp
      ultimately have sat  $\{F, \text{Not } F\}$  using fs unfolding fin-sat-def by blast
      thus False unfolding sat-def by simp
    qed
  with True show ?thesis by blast
qed
qed

```

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 –
  from assms[unfolded fin-sat-def, simplified] obtain  $s :: 'a :: \text{countable formula set}$ 
  where finite  $s \neg \text{sat } s$  by clarify
  from this(2)[unfolded sat-def, simplified] have  $\exists x \in s. \neg A \models x$  for  $A ..$ 
  have  $nfs: \neg \text{fin-sat } (\text{insert } x (\text{extender } n S))$  for  $n x$ 
    apply(rule notI)
    apply(drule fin-sat-insert)
    apply(drule fin-sat-antimono)
    apply(rule extender-subset)
    apply(erule notE[rotated])
    apply(fact assms)
  done
  have  $x \in \bigcup \{\text{extender } n S \mid n. \text{True}\}$  for  $x$  proof cases
    assume  $x \in S$  thus ?thesis by (metis extended-def extended-superset insert-absorb insert-subset)
  next
    assume  $x \notin S$ 
    have  $x \in \text{extender } (\text{Suc } (\text{to-nat } x)) S$ 
      unfolding extender.simps Let-def
      unfolding if-not-P[OF nfs] by simp
    thus ?thesis by blast
  qed
thus ?thesis unfolding extended-def by auto
qed

```

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.

oops

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

proof –

have *ex-then-min*: $\exists x :: \text{nat}. P x \implies P (\text{LEAST } x. P x)$ **for** P **using** *LeastI2-wellorder* **by** *auto*

def $P \equiv \lambda x. \text{let } F = (\text{from-nat } x :: 'a \text{ formula}) \text{ in } (\exists A. \neg A \models F) \wedge (\exists A. A \models F)$

def $x \equiv \text{LEAST } n. P n$

hence $\exists n. P n$ **unfolding** *P-def* **Let-def** **by**(*auto intro!*: *exI*[**where** $x=\text{to-nat}$ (*Atom undefined* :: 'a formula)])

from *ex-then-min*[*OF this*] **have** $Px: P x$ **unfolding** *x-def* .

have *lessx*: $n < x \implies \neg P n$ **for** n **unfolding** *x-def* **using** *not-less-Least* **by** *blast*

let $?S = \{\} :: 'a \text{ formula set}$ **let** $?T = \{\text{from-nat } x :: 'a \text{ formula}\}$

have $s: \text{fin-sat } ?S \text{ fin-sat } ?T$ **using** Px **unfolding** *P-def* *fin-sat-def* *sat-def* *Let-def* **by** *fastforce+*

have *reject*: $Q A \implies \forall A. \neg Q A \implies \text{False}$ **for** $A Q$ **by** *simp*

have $y \leq x \implies F \in \text{extender } y ?S \implies \models F$ **for** $F y$

proof(*induction y arbitrary: F*)

case (*Suc y*)

have $*$: $F \in \text{extender } y \{\} \implies \models F$ **for** $F :: 'a \text{ formula}$ **using** *Suc.IH* *Suc.prem(1)* **by** *auto*

let $?Y = \text{from-nat } y :: 'a \text{ formula}$

have *ex*: $(\forall A. \neg A \models ?Y) \vee \models ?Y$ **unfolding** *formula-antics.simps* **by** (*meson P-def Suc.prem(1) Suc-le-lessD lessx*)

have $1: \forall A. \neg A \models ?Y$ **if** *fin-sat* (*Not ?Y* \triangleright *extender y ?S*)

proof –

note[[*show-types*]]

from *that* **have** $\exists A. A \models \text{Not } ?Y$ **unfolding** *fin-sat-def* *sat-def* **by**(*elim allE*[**where** $x=\{\text{Not } ?Y\}$]) *simp*

hence $\neg \models ?Y$ **by** *simp*

hence $\forall A. \neg A \models ?Y$ **using** *ex* **by** *argo*

thus *thesis* **by** *simp*

qed

have $2: \neg \text{fin-sat } (\text{Not } ?Y \triangleright \text{extender } y ?S) \implies \models ?Y$

proof(*erule contrapos-mp*)

assume $\neg \models ?Y$

hence $\forall A. \neg A \models ?Y$ **using** *ex* **by** *argo*

hence $\models \neg ?Y$ **by** *simp*

thus *fin-sat* ($\neg ?Y \triangleright \text{extender } y ?S$) **unfolding** *fin-sat-def* *sat-def*

by(*auto intro!*: *exI*[**where** $x=\lambda-$:: '*a. False*] *dest!*: *rev-subsetD*[*rotated*] *)
 qed
 show ?*case* **using** *Suc.prem*s(2) **by**(*simp add*: *Let-def split: if-splits; elim disjE*;
simp add: * 1 2)
 qed *simp*
 hence *fin-sat* (\neg (*from-nat* x) \triangleright *extender* x ?*S*) **using** *Px* **unfolding** *P-def*
Let-def
 by (*clarsimp simp*: *fin-sat-def sat-def*) (*insert formula-antics.simps*(3),
blast)
 hence *Not* (*from-nat* x) \in *extender* (*Suc* x) ?*S* **by**(*simp*)
 hence *Not* (*from-nat* x) \in *extended* ?*S* **unfolding** *extended-def* **by** *blast*
 moreover **have** *Not* (*from-nat* x) \notin *extended* ?*T* **using** *extended-complem*
extended-superset s(2) **by** *blast*
 ultimately show ?*thesis* **using** *s* **by** *blast*
 qed

private lemma *not-in-extended-FE*: *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(*goal-cases*)
 case 1
 hence *Not* $F \in \text{extended } S$ **using** *extended-max* **by** *blast*
 thus *False* **using** 1 *extended-fin-sat fin-sat-def*
 by (*metis Diff-eq-empty-iff finite.insertI insert-Diff-if*)
 qed

lemma *extended-id*: *extended* (*extended* S) = *extended* S
using *extended-complem extended-fin-sat extended-max extended-superset not-fin-sat-extended-UNIV*

 by(*intro equalityI*[*rotated*] *extended-superset*) *blast*

lemma *ext-model*:
assumes r : *fin-sat* S
shows ($\lambda k. \text{Atom } k \in \text{extended } S$) $\models F \longleftrightarrow F \in \text{extended } S$
proof –
 note $fs = r[\text{THEN } \text{extended-fin-sat}]$
have *Elim*: $F \in S \wedge G \in S \implies \{F, G\} \subseteq S \implies \{F\} \subseteq S$ **for** $F G S$ **by**
simp+
show ?*thesis*
proof(*induction* F)
 case *Atom* **thus** ?*case* **by**(*simp*)
next
 case *Bot*
have *False* **if** $\perp \in \text{extended } S$ **proof** –
have *finite* $\{\perp\}$ **by** *simp*
moreover from that **have** $\{\perp\} \subseteq \text{extended } S$ **by** *simp*
ultimately have $\exists A. A \models \perp$ **using** fs **unfolding** *fin-sat-def sat-def*
by(*elim allE*[*of* - $\{\perp\}$]) *simp*


```

      thus False by simp
    qed
  thus ?case by auto
next
  case (Not F)
  moreover have  $A \models F \neq A \models \neg F$  for  $A F$  by simp
  ultimately show ?case using extended-complem[OF r] by blast
next
  case (And F G)
  have  $(F \in \text{extended } S \wedge G \in \text{extended } S) = (F \wedge G \in \text{extended } S)$  proof –
    have *:  $\neg \text{sat } \{\neg (F \wedge G), F, G\} \neg \text{sat } \{\neg F, (F \wedge G)\} \neg \text{sat } \{\neg G, (F \wedge G)\}$ 
  } unfolding sat-def by auto
  show ?thesis by(intro iffI; rule ccontr) (auto intro: *[THEN not-in-extended-FE[OF r]]])
  qed
  thus ?case by(simp add: And)
next
  case (Or F G)
  have  $(F \in \text{extended } S \vee G \in \text{extended } S) = (F \vee G \in \text{extended } S)$  proof –
    have  $\neg \text{sat } \{\neg (F \vee G), F\} \neg \text{sat } \{\neg (F \vee G), G\}$  unfolding sat-def by auto
    from this[THEN not-in-extended-FE[OF r]] have 1:  $\llbracket F \in \text{extended } S \vee G \in \text{extended } S; F \vee G \notin \text{extended } S \rrbracket \implies \text{False}$  by auto
    have  $\neg \text{sat } \{\neg F, \neg G, F \vee G\}$  unfolding sat-def by auto
    hence 2:  $\llbracket F \vee G \in \text{extended } S; F \notin \text{extended } S; G \notin \text{extended } S \rrbracket \implies \text{False}$ 
  using extended-max not-in-extended-FE[OF r] by fastforce
  show ?thesis by(intro iffI; rule ccontr) (auto intro: 1 2)
  qed
  thus ?case by(simp add: Or)
next
  case (Imp F G)
  have  $(F \in \text{extended } S \longrightarrow G \in \text{extended } S) = (F \rightarrow G \in \text{extended } S)$  proof –
    have  $\neg \text{sat } \{\neg G, F, F \rightarrow G\}$  unfolding sat-def by auto
    hence 1:  $\llbracket F \rightarrow G \in \text{extended } S; F \in \text{extended } S; G \notin \text{extended } S \rrbracket \implies \text{False}$ 
  using extended-max not-in-extended-FE[OF r] by blast
  have  $\neg \text{sat } \{\neg F, \neg(F \rightarrow G)\}$  unfolding sat-def by auto
  hence 2:  $\llbracket F \rightarrow G \notin \text{extended } S; F \notin \text{extended } S \rrbracket \implies \text{False}$  using extended-max not-in-extended-FE[OF r] by blast
  have  $\neg \text{sat } \{\neg(F \rightarrow G), G\}$  unfolding sat-def by auto
  hence 3:  $\llbracket F \rightarrow G \notin \text{extended } S; G \in \text{extended } S \rrbracket \implies \text{False}$  using extended-max not-in-extended-FE[OF r] by blast
  show ?thesis by(intro iffI; rule ccontr) (auto intro: 1 2 3)
  qed
  thus ?case by(simp add: Imp)
  qed
  qed

```

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

```

  shows  $sat\ S \longleftrightarrow fin\text{-}sat\ S$  (is ?l = ?r)
proof
  assume ?l thus ?r unfolding sat-def fin-sat-def by blast
next
  assume r: ?r
  note ext-model[OF r, THEN iffD2]
  hence  $\forall F \in S. (\lambda k. Atom\ k \in extended\ S) \models F$  using extended-superset by blast
  thus ?l unfolding sat-def by blast
qed

```

corollary compact-entailment:

```

  fixes  $F :: 'a :: countable\ formula$ 
  assumes fent:  $\Gamma \models F$ 
  shows  $\exists \Gamma'. finite\ \Gamma' \wedge \Gamma' \subseteq \Gamma \wedge \Gamma' \models F$ 
proof -
  have ND-sem:  $\Gamma \models F \longleftrightarrow \neg sat\ (insert\ (\neg F)\ \Gamma)$ 
  for  $\Gamma\ F$  unfolding sat-def entailment-def by auto
  obtain  $\Gamma'$  where 0:  $finite\ \Gamma'\ \Gamma' \models F\ \Gamma' \subseteq \Gamma$  proof(goal-cases)
  from fent[unfolded ND-sem compactness] have  $\neg fin\text{-}sat\ (insert\ (\neg F)\ \Gamma)$ .
  from this[unfolded fin-sat-def] obtain  $s$  where  $s: s \subseteq insert\ (\neg F)\ \Gamma\ finite\ s$ 
 $\neg sat\ s$  by blast
  have 2:  $finite\ (s - \{\neg F\})$  using  $s$  by simp
  have 3:  $s - \{\neg F\} \models F$  unfolding ND-sem using  $s(3)$  unfolding sat-def
  by blast
  have 4:  $s - \{\neg F\} \subseteq \Gamma$  using  $s$  by blast
  case 1 from 2 3 4 show ?case by(intro 1[of  $s - \{\neg F\}$ ])
  qed
  thus ?thesis by blast
qed

```

corollary compact-to-formula:

```

  fixes  $F :: 'a :: countable\ formula$ 
  assumes fent:  $\Gamma \models F$ 
  obtains  $\Gamma'$  where  $set\ \Gamma' \subseteq \Gamma \models (\bigwedge \Gamma') \rightarrow F$ 
proof goal-cases
  case 1
  note compact-entailment[OF assms]
  then guess  $\Gamma'$  ..
  moreover then obtain  $\Gamma''$  where  $\Gamma' = set\ \Gamma''$  using finite-list by auto
  ultimately show thesis by(intro 1) (blast, simp add: entailment-def)
qed

```

end

end

theory Compactness-Consistency

imports Consistency

begin

theorem $\text{sat } S \longleftrightarrow \text{fin-sat } (S :: 'a :: \text{countable formula set})$ (is ?l = ?r)

proof

assume 0: ?r

let ?C = { W :: 'a formula set. fin-sat W }

have 1: $S \in ?C$ **using** 0 **unfolding** mem-Collect-eq .

have 2: pcp ?C **proof** –

 { **fix** S :: 'a formula set

assume S ∈ ?C

hence a: $\forall s \subseteq S. \text{finite } s \longrightarrow (\exists \mathcal{A}. \forall F \in s. \mathcal{A} \models F)$ **by** (simp add: fin-sat-def sat-def)

have conj: $\llbracket h \ F \ G \in S; s \subseteq f \ F \triangleright g \ G \triangleright S; \text{finite } s; \bigwedge \mathcal{A}. \mathcal{A} \models h \ F \ G \implies \mathcal{A} \models f \ F \wedge \mathcal{A} \models g \ G \rrbracket \implies \exists \mathcal{A}. \forall F \in s. \mathcal{A} \models F$

for F G s **and** f g :: 'a formula \implies 'a formula **and** h :: 'a formula \implies 'a formula

proof goal-cases

case 1

have h F G $\triangleright s - \{f \ F, g \ G\} \subseteq S$ finite (h F G $\triangleright s - \{f \ F, g \ G\}$) **using** 1 **by** auto

then obtain A **where** 2: $\forall H \in h \ F \ G \triangleright s - \{f \ F, g \ G\}. A \models H$ **using** a **by** presburger

hence A $\models f \ F$ A $\models g \ G$ **using** 1(4) **by** simp-all

with 2 **have** $\forall H \in h \ F \ G \triangleright s. A \models H$ **by** blast

thus ?case **by** blast

qed

have disj: $\llbracket h \ F \ G \in S; s1 \subseteq f \ F \triangleright S; s2 \subseteq g \ G \triangleright S; \text{finite } s1; \forall \mathcal{A}. \exists x \in s1. \neg \mathcal{A} \models x; \text{finite } s2; \forall \mathcal{A}. \exists x \in s2. \neg \mathcal{A} \models x; \bigwedge \mathcal{A}. \mathcal{A} \models h \ F \ G \implies \mathcal{A} \models f \ F \vee \mathcal{A} \models g \ G \rrbracket \implies \text{False}$

for F G s1 s2 **and** f g :: 'a formula \implies 'a formula **and** h :: 'a formula \implies 'a formula

proof goal-cases

case 1

let ?U = h F G $\triangleright (s1 - \{f \ F\}) \cup (s2 - \{g \ G\})$

have ?U $\subseteq S$ finite ?U **using** 1 **by** auto

with a **obtain** A **where** 2: $H \in ?U \implies A \models H$ **for** H **by** meson

with 1(1) 1(8) **have** A $\models f \ F \vee A \models g \ G$ **by** force

hence $(\forall H \in s1. A \models H) \vee (\forall H \in s1. A \models H)$ **using** 1(7) 2

by (metis DiffI Diff-empty Diff-iff UnCI insert-iff)

thus ?case **using** 1 **by** fast

qed

have 1: $\perp \notin S$ **using** a **by** (meson empty-subsetI finite.emptyI finite.insertI formula-anticsimps(2) insertI1 insert-subset)

have 2: $\text{Atom } k \in S \longrightarrow \neg (\text{Atom } k) \in S \longrightarrow \text{False}$ **for** k **using** a[THEN spec[of - {Atom k, $\neg(\text{Atom } k)$]}] **by** auto

have 3: $F \wedge G \in S \longrightarrow F \triangleright G \triangleright S \in \text{Collect fin-sat}$ **for** F G **unfolding** fin-sat-def sat-def mem-Collect-eq **using** conj[] **by** fastforce

have 4: $F \vee G \in S \longrightarrow F \triangleright S \in \text{Collect fin-sat} \vee G \triangleright S \in \text{Collect fin-sat}$ **for** F G

unfolding fin-sat-def sat-def mem-Collect-eq **using** disj[of Or F G - $\lambda k. k - \lambda k. k$] **by** (metis formula-anticsimps(5))

```

have 5:  $F \rightarrow G \in S \longrightarrow \neg F \triangleright S \in \text{Collect fin-sat} \vee G \triangleright S \in \text{Collect fin-sat}$ 
for  $F G$ 
  unfolding fin-sat-def sat-def mem-Collect-eq using disj[of Imp F G - Not -  $\lambda k. k$ ] by (metis formula-semantic.simps(3,6))
  have 6:  $\neg(\neg F) \in S \longrightarrow F \triangleright S \in \text{Collect fin-sat}$  for  $F$  unfolding fin-sat-def sat-def mem-Collect-eq using conj[of  $\lambda F G. \text{Not}(\text{Not } F) F F - \lambda k. k \lambda k. k$ ] by simp
  have 7:  $\neg(F \wedge G) \in S \longrightarrow \neg F \triangleright S \in \text{Collect fin-sat} \vee \neg G \triangleright S \in \text{Collect fin-sat}$  for  $F G$ 
  unfolding fin-sat-def sat-def mem-Collect-eq using disj[of  $\lambda F G. \text{Not}(\text{And } F G) F G - \text{Not} - \text{Not}$ ] by (metis formula-semantic.simps(3,4))
  have 8:  $\forall F G. \neg(F \vee G) \in S \longrightarrow \neg F \triangleright \neg G \triangleright S \in \text{Collect fin-sat}$ 
unfolding fin-sat-def sat-def mem-Collect-eq using conj[] by fastforce
  have 9:  $\forall F G. \neg(F \rightarrow G) \in S \longrightarrow F \triangleright \neg G \triangleright S \in \text{Collect fin-sat}$  unfolding fin-sat-def sat-def mem-Collect-eq using conj[] by fastforce
  note 1 2 3 4 5 6 7 8 9
}
thus pcp ?C unfolding pcp-def by auto
qed
from pcp-sat 2 1 show ?l .
next
  assume ?l thus ?r unfolding sat-def fin-sat-def by blast
qed

```

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  $\mathcal{A} \models ((F[\top / n]) \vee (F[\perp / n]))$ 
using assms by(cases A n; simp add: substitution-lemma fun-upd-idem)

```

theorem *interpolation*:

```

assumes  $\models \Gamma \rightarrow \Delta$ 
obtains  $\varrho$  where
   $\models \Gamma \rightarrow \varrho \models \varrho \rightarrow \Delta$ 
   $\text{atoms } \varrho \subseteq \text{atoms } \Gamma$ 
   $\text{atoms } \varrho \subseteq \text{atoms } \Delta$ 
proof(goal-cases)
  let  $?as = \text{atoms } \Gamma - \text{atoms } \Delta$ 
  have fas: finite ?as by simp
  from fas assms have  $\exists \varrho. ((\models \Gamma \rightarrow \varrho) \wedge (\models \varrho \rightarrow \Delta) \wedge (\text{atoms } \varrho \subseteq \text{atoms } \Gamma) \wedge (\text{atoms } \varrho \subseteq \text{atoms } \Delta))$ 

```

```

proof(induction ?as arbitrary:  $\Gamma$  rule: finite-induct)
  case empty
    from  $\langle \{\} = \text{atoms } \Gamma - \text{atoms } \Delta \rangle$  have  $\text{atoms } \Gamma \subseteq \text{atoms } \Delta$  by blast
    with  $\langle \models \Gamma \rightarrow \Delta \rangle$  show ?case by(intro exI[where x= $\Gamma$ ]) simp
  next
    case (insert a A)
    hence  $e: a \in \text{atoms } \Gamma \wedge a \notin \text{atoms } \Delta$  by auto
    def  $\Gamma' \equiv (\Gamma[\top / a]) \vee (\Gamma[\perp / a])$ 
    have  $su: \text{atoms } \Gamma' \subseteq \text{atoms } \Gamma$  unfolding  $\Gamma'$ -def by(cases a  $\in$  atoms  $\Gamma$ ; simp
add: subst-atoms Diff-subset)
    from  $\langle \models \Gamma \rightarrow \Delta \rangle$   $e$  have  $\models \Gamma' \rightarrow \Delta$  by (auto simp add: substitution-lemma
 $\Gamma'$ -def)
    from  $\langle a \triangleright A = \text{atoms } \Gamma - \text{atoms } \Delta \rangle$   $\langle a \notin A \rangle$   $e$  have  $A = \text{atoms } \Gamma' - \text{atoms}$ 
 $\Delta$  by(simp add: subst-atoms  $\Gamma'$ -def) blast
    from insert.hyps(3)[OF this  $\langle \models \Gamma' \rightarrow \Delta \rangle$ ] obtain  $\varrho$  where  $\varrho: \models \Gamma' \rightarrow \varrho \models \varrho$ 
 $\rightarrow \Delta$   $\text{atoms } \varrho \subseteq \text{atoms } \Gamma'$   $\text{atoms } \varrho \subseteq \text{atoms } \Delta$  by clarify
    have  $\models \Gamma \rightarrow \varrho$  using  $\varrho(1)$  subst-true-false unfolding  $\Gamma'$ -def by fastforce
    with  $\varrho$   $su$  show ?case by(intro exI[where x= $\varrho$ ]) simp
  qed
  moreover case 1
  ultimately show thesis by blast
qed

```

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

```

function interpolate where
  interpolate F H = (
  let K = atoms F - atoms H in
  if K = {}
  then F
  else (
  let k = Min K
  in interpolate ((F[\top / k]) \vee (F[\perp / k])) H
  )
  ) by pat-completeness simp

```

Showing termination is slightly technical...

```

termination interpolate
  apply(relation measure ( $\lambda(F,H). \text{card} (\text{atoms } F - \text{atoms } H)$ ))

  subgoal by simp
  apply (simp add: subst-atoms-simp)
  apply(intro conjI impI)
  apply(intro psubset-card-mono)
  subgoal by simp
  apply(subgoal-tac Min (atoms F - atoms H)  $\notin$  atoms H)
  subgoal by blast
  apply (meson atoms-finite Diff-eq-empty-iff Diff-iff Min-in finite-Diff)+
done

```

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)) =
 ($\top \wedge \text{Atom } 1$) \vee ($\perp \wedge \text{Atom } 1$) **by** *simp*
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) = $23 \wedge \text{simplify-consts } I = \text{Atom } 2 \rightarrow \text{Atom } 0$
by *code-simp*

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 –

let $?as = \text{atoms } F - \text{atoms } H$

have *fas*: *finite* $?as$ **by** *simp*

hence $?t1 \wedge ?t2 \wedge ?s$ **using** *assms*

proof(*induction* *card* $?as$ *arbitrary*: $F \ H$)

case (*Suc* n)

let $?inf = \text{Min } (\text{atoms } F - \text{atoms } H)$

def $G \equiv (F[\top / ?inf]) \vee (F[\perp / ?inf])$

have e : $\text{Min } (\text{atoms } F - \text{atoms } H) \in \text{atoms } F - \text{atoms } H$ **by** (*metis* *Min-in* *Suc.hyyps*(2) *Suc.prem*(1) *card-empty* *nat.simps*(3))

with *Suc*(2) **have** $n = \text{card } (\text{atoms } G - \text{atoms } H)$ **unfolding** *G-def* *subst-atoms-simp*

proof –

assume $a1$: $\text{Suc } n = \text{card } (\text{atoms } F - \text{atoms } H)$

assume $\text{Min } (\text{atoms } F - \text{atoms } H) \in \text{atoms } F - \text{atoms } H$

hence $a2$: $\text{Min } (\text{atoms } F - \text{atoms } H) \in \text{atoms } F \wedge \text{Min } (\text{atoms } F - \text{atoms } H) \notin \text{atoms } H$ **by** *simp*

have $n = \text{card } (\text{atoms } F - \text{atoms } H) - 1$

using $a1$ **by** *presburger*

hence $n = \text{card } (\text{atoms } (F[\top / \text{Min } (\text{atoms } F - \text{atoms } H)]) \cup \text{atoms } (F[\perp / \text{Min } (\text{atoms } F - \text{atoms } H)])) - \text{atoms } H$

using $a2$ **by** (*metis* (*full-types*) *formula.set*(2) *Diff-insert* *Diff-insert2* *Suc.prem*(1) *Un-absorb* *Un-empty-right* *card-Diff-singleton* e *subst-atoms* *top-atoms-simp*)

thus $n = \text{card } (\text{atoms } ((F[\top / \text{Min } (\text{atoms } F - \text{atoms } H)]) \vee (F[\perp / \text{Min } (\text{atoms } F - \text{atoms } H)])) - \text{atoms } H)$ **by** *simp*

qed

moreover **have** *finite* $(\text{atoms } G - \text{atoms } H) \models G \rightarrow H$ **using** *Suc*(3-) e

by(*auto* *simp*: *G-def* *substitution-lemma*)

ultimately **have** IH : $\models G \rightarrow \text{interpolate } G \ H \models \text{interpolate } G \ H \rightarrow H$
 $\text{atoms } (\text{interpolate } G \ H) \subseteq \text{atoms } G \cap \text{atoms } H$ **using** *Suc* **by** *blast+*

```

moreover have  $\models F \rightarrow G$  unfolding G-def
  using subst-true-false by fastforce
moreover {
  assume a1: atoms (interpolate ((F[ $\top$ /Min (atoms F - atoms H))]  $\vee$ 
(F[ $\perp$ /Min (atoms F - atoms H)])) H)  $\subseteq$  atoms (F[ $\top$ /Min (atoms F - atoms H))]  $\cup$ 
atoms (F[ $\perp$ /Min (atoms F - atoms H)]))  $\wedge$  atoms (interpolate ((F[ $\top$ /Min
(atoms F - atoms H))]  $\vee$  (F[ $\perp$ /Min (atoms F - atoms H)])) H)  $\subseteq$  atoms H
  have f2: atoms (( $\perp$ ::'a formula)  $\rightarrow$   $\perp$ ) = atoms  $\perp$ 
  by simp
  then have f3: atoms F - {Min (atoms F - atoms H)} = atoms (F[ $\top$ /Min
(atoms F - atoms H)]))
  by (metis (no-types) DiffD1 Top-def Un-empty-right e formula.simps(91)
subst-atoms)
  have atoms (F[ $\perp$ /Min (atoms F - atoms H))] = atoms (F[ $\top$ /Min (atoms F -
atoms H)]))
  using f2 by (metis (no-types) DiffD1 Top-def e subst-atoms)
  then have  $\neg$  atoms F  $\subseteq$  atoms H  $\longrightarrow$  atoms (interpolate ((F[ $\top$ /Min (atoms F -
atoms H)]))  $\vee$  (F[ $\perp$ /Min (atoms F - atoms H)])) H)  $\subseteq$  atoms F
  using f3 a1 by blast
} ultimately show ?case
  by (intro conjI; subst interpolate.simps; simp del: interpolate.simps add: Let-def G-def;
blast?)
qed auto
thus ?t1 ?t2 ?s by simp-all
qed

```

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 ( $\cdot$ , - [56,56] 56) where  $x, M ==$  add-mset x M

```

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

```

inductive SCp :: 'a formula multiset  $\Rightarrow$  'a formula multiset  $\Rightarrow$  bool (( $\cdot \Rightarrow / \cdot$ ) [53]
53) where
  BotL:  $\perp \in \# \Gamma \Longrightarrow \Gamma \Rightarrow \Delta$  |
  Ax: Atom k  $\in \# \Gamma \Longrightarrow$  Atom k  $\in \# \Delta \Longrightarrow \Gamma \Rightarrow \Delta$  |
  NotL:  $\Gamma \Rightarrow F, \Delta \Longrightarrow$  Not F,  $\Gamma \Rightarrow \Delta$  |

```

NotR: $F, \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \text{Not } F, \Delta \mid$
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* $k, \Gamma \Rightarrow \text{Atom } k, \Delta$
by (*meson SCp.intros union-single-eq-member*)⁺

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

by (*metis (no-types, lifting) add-eq-conv-ex add-mset-remove-trivial add-mset-remove-trivial-eq*)

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

using *lem1* **by** *fastforce*

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*→*J*) *H* $\Longrightarrow H, F \rightarrow G, S = F \rightarrow G, H, S$
NO-MATCH (*I*→*J*) *H* $\Longrightarrow \text{NO-MATCH } (\text{IN } J) \ H \Longrightarrow H, F \vee G, S = F \vee G, H, S$
NO-MATCH (*I*→*J*) *H* $\Longrightarrow \text{NO-MATCH } (\text{IN } J) \ H \Longrightarrow \text{NO-MATCH } (\text{I} \wedge J) \ H$
 $\Longrightarrow H, F \wedge G, S = F \wedge G, H, S$
NO-MATCH (*I*→*J*) *H* $\Longrightarrow \text{NO-MATCH } (\text{IN } J) \ H \Longrightarrow \text{NO-MATCH } (\text{I} \wedge J) \ H$
 $\Longrightarrow \text{NO-MATCH } (\neg J) \ H \Longrightarrow H, \neg G, S = \neg G, H, S$
NO-MATCH (*I*→*J*) *H* $\Longrightarrow \text{NO-MATCH } (\text{IN } J) \ H \Longrightarrow \text{NO-MATCH } (\text{I} \wedge J) \ H$
 $\Longrightarrow \text{NO-MATCH } (\neg J) \ H \Longrightarrow \text{NO-MATCH } (\perp) \ H \Longrightarrow H, \perp, S = \perp, H, S$
NO-MATCH (*I*→*J*) *H* $\Longrightarrow \text{NO-MATCH } (\text{IN } J) \ H \Longrightarrow \text{NO-MATCH } (\text{I} \wedge J) \ H$
 $\Longrightarrow \text{NO-MATCH } (\neg J) \ H \Longrightarrow \text{NO-MATCH } (\perp) \ H \Longrightarrow \text{NO-MATCH } (\text{Atom } k)$
 $H \Longrightarrow H, \text{Atom } l, S = \text{Atom } l, H, S$

by *simp-all*

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*: $F, G, H, \Gamma \Rightarrow \Delta \Longrightarrow G, H, F, \Gamma \Rightarrow \Delta$ **by** (*simp-all add: add-mset-commute*)
lemmas *SCp-swap-applies*[*elim!, intro*] = *inL1 inL2 inR1 inR2*

lemma *NotL-inv*: $\text{Not } F, \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow F, \Delta$

proof (*induction* *Not* *F*, $\Gamma \ \Delta$ *arbitrary*: Γ *rule*: *SCp.induct*)

case (*NotL* $\Gamma' \ G \ \Delta$) **thus** *?case* **by** (*cases* *Not* *F* = *Not* *G*) (*auto intro!*: *SCp.intros*(3-)
dest!: *multi-member-split simp: lem1 lem2*)

qed (*auto intro!*: *SCp.intros(3-)* *dest!*: *multi-member-split simp*: *SCp.intros lem1 lem2*)

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

proof(*induction And F G, $\Gamma \Delta$ arbitrary: Γ rule: SCp.induct*)

case (*AndL F' G' $\Gamma' \Delta$*) **thus** *?case*

by(*cases And F G = And F' G'*;

auto intro!: *SCp.intros(3-)* *dest!*: *multi-member-split simp*: *lem1 lem2*;

metis add-mset-commute)

qed (*auto intro!*: *SCp.intros(3-)* *dest!*: *multi-member-split simp*: *SCp.intros lem1 lem2 inL2*)

lemma *OrL-inv*:

assumes *Or F G, $\Gamma \Rightarrow \Delta$*

shows *F, $\Gamma \Rightarrow \Delta \wedge G, \Gamma \Rightarrow \Delta$*

proof(*insert assms, induction Or F G, $\Gamma \Delta$ arbitrary: Γ rule: SCp.induct*)

case (*OrL F' $\Gamma' \Delta G'$*) **thus** *?case*

by(*cases Or F G = Or F' G'*;

auto intro!: *SCp.intros(3-)* *dest!*: *multi-member-split simp*: *lem1 lem2*;

blast)

qed (*fastforce intro!*: *SCp.intros(3-)* *dest!*: *multi-member-split simp*: *SCp.intros lem1 lem2*)+

lemma *ImpL-inv*:

assumes *Imp F G, $\Gamma \Rightarrow \Delta$*

shows *$\Gamma \Rightarrow F, \Delta \wedge G, \Gamma \Rightarrow \Delta$*

proof(*insert assms, induction Imp F G, $\Gamma \Delta$ arbitrary: Γ rule: SCp.induct*)

case (*ImpL $\Gamma' F' \Delta G'$*) **thus** *?case*

by(*cases Or F G = Or F' G'*;

auto intro!: *SCp.intros(3-)* *dest!*: *multi-member-split simp*: *lem1 lem2*;

blast)

qed (*fastforce intro!*: *SCp.intros(3-)* *dest!*: *multi-member-split simp*: *SCp.intros lem1 lem2*)+

lemma *ImpR-inv*:

assumes *$\Gamma \Rightarrow Imp\ F\ G, \Delta$*

shows *F, $\Gamma \Rightarrow G, \Delta$*

proof(*insert assms, induction $\Gamma Imp\ F\ G, \Delta$ arbitrary: Δ rule: SCp.induct*)

case (*ImpR F' $\Gamma G' \Delta'$*) **thus** *?case*

by(*cases Or F G = Or F' G'*;

auto intro!: *SCp.intros(3-)* *dest!*: *multi-member-split simp*: *lem1 lem2*;

blast)

qed (*fastforce intro!*: *SCp.intros(3-)* *dest!*: *multi-member-split simp*: *SCp.intros lem1 lem2*)+

lemma *AndR-inv*:

shows *$\Gamma \Rightarrow And\ F\ G, \Delta \Longrightarrow \Gamma \Rightarrow F, \Delta \wedge \Gamma \Rightarrow G, \Delta$*

proof(*induction $\Gamma And\ F\ G, \Delta$ arbitrary: Δ rule: SCp.induct*)

case (*AndR $\Gamma F' \Delta' G'$*) **thus** *?case*

by(cases Or F G = Or F' G';
 auto intro!: SCp.intros(3-) dest!: multi-member-split simp: lem1 lem2;
 blast)
qed (fastforce intro!: SCp.intros(3-) dest!: multi-member-split simp: SCp.intros
 lem1 lem2)+

lemma OrR-inv: $\Gamma \Rightarrow Or F G, \Delta \Longrightarrow \Gamma \Rightarrow F, G, \Delta$
proof(induction $\Gamma Or F G, \Delta$ arbitrary: Δ rule: SCp.induct)
case (OrR $\Gamma F' G' \Delta'$) **thus** ?case
by(cases Or F G = Or F' G';
 auto intro!: SCp.intros(3-) dest!: multi-member-split simp: lem1 lem2;
 metis add-mset-commute)
qed (fastforce intro!: SCp.intros(3-) dest!: multi-member-split simp: SCp.intros
 lem1 lem2)+

lemma NotR-inv: $\Gamma \Rightarrow Not F, \Delta \Longrightarrow F, \Gamma \Rightarrow \Delta$
proof(induction $\Gamma Not F, \Delta$ arbitrary: Δ rule: SCp.induct)
case (NotR $G \Gamma \Delta'$) **thus** ?case
by(cases Not F = Not G;
 auto intro!: SCp.intros(3-) dest!: multi-member-split simp: lem1 lem2;
 metis add-mset-commute)
qed (fastforce intro!: SCp.intros(3-) dest!: multi-member-split simp: SCp.intros
 lem1 lem2)+

lemma weakenL: $\Gamma \Rightarrow \Delta \Longrightarrow F, \Gamma \Rightarrow \Delta$
by(induction rule: SCp.induct)
 (auto intro!: SCp.intros(3-) intro: SCp.intros(1,2))

lemma weakenR: $\Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow F, \Delta$
by(induction rule: SCp.induct)
 (auto intro!: SCp.intros(3-) intro: SCp.intros(1,2))

lemma weakenL-set: $\Gamma \Rightarrow \Delta \Longrightarrow F + \Gamma \Rightarrow \Delta$
by(induction F; simp add: weakenL)

lemma weakenR-set: $\Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow F + \Delta$
by(induction F; simp add: weakenR)

lemma extended-Ax: $\Gamma \cap \# \Delta \neq \{\#\} \Longrightarrow \Gamma \Rightarrow \Delta$
proof –
assume $\Gamma \cap \# \Delta \neq \{\#\}$
then obtain W **where** $W \in \# \Gamma W \in \# \Delta$ **by** force
then show ?thesis **proof**(induction W arbitrary: $\Gamma \Delta$)
case (Not W)
from Not.prem **obtain** $\Gamma' \Delta'$ **where** [simp]: $\Gamma = Not W, \Gamma' \Delta = Not W, \Delta'$
by (metis insert-DiffM)
have $W \in \# W, \Gamma' W \in \# W, \Delta'$ **by** simp-all
from Not.IH[OF this] **obtain** n **where** $W, \Gamma' \Rightarrow W, \Delta'$ **by** presburger
hence $Not W, \Gamma' \Rightarrow Not W, \Delta'$ **using** SCp.intros(3,4) add-mset-commute **by**
 metis

```

    thus  $\Gamma \Rightarrow \Delta$  by auto
  next
    case (And G H)
    from And.prem1 obtain  $\Gamma' \Delta'$  where [simp]:  $\Gamma = \text{And } G \ H, \Gamma' \Delta = \text{And } G$ 
     $H, \Delta'$  by (metis insert-DiffM)
    have  $G \in\# \ G, H, \Gamma' \ G \in\# \ G, \Delta'$  by simp-all
    with And.IH(1) have IH1:  $G, H, \Gamma' \Rightarrow G, \Delta'$ .
    have  $H \in\# \ G, H, \Gamma' \ H \in\# \ H, \Delta'$  by simp-all
    with And.IH(2) have IH2:  $G, H, \Gamma' \Rightarrow H, \Delta'$ .
    from IH1 IH2 have  $G, H, \Gamma' \Rightarrow G, \Delta' \ G, H, \Gamma' \Rightarrow H, \Delta'$  by fast+
    thus  $\Gamma \Rightarrow \Delta$  using SCp.intros(5,6) by fastforce
  next
    case (Or G H)
    case (Imp G H)

```

analogously

```

  qed (auto intro: SCp.intros)
qed

```

lemma Bot-delR: $\Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta - \{\# \perp \#\}$

proof(induction rule: SCp.induct)

```

  case BotL
  thus ?case by (simp add: SCp.BotL)
next
  case Ax
  thus ?case
  by (metis SCp.Ax diff-union-swap formula.distinct(1) insert-DiffM union-single-eq-member)
next
  case NotL
  thus ?case
  by (metis SCp.NotL diff-single-trivial diff-union-swap2)
next
  case NotR
  thus ?case by (simp add: SCp.NotR)
next
  case AndL
  thus ?case by (simp add: SCp.AndL)
next
  case AndR
  thus ?case
  by (metis SCp.AndR diff-single-trivial diff-union-swap diff-union-swap2 formula.distinct(13))
next
  case OrL
  thus ?case by (simp add: SCp.OrL)
next
  case OrR
  thus ?case
  by (metis SCp.OrR diff-single-trivial diff-union-swap2 formula.distinct(15))

```

```

insert-iff set-mset-add-mset-insert)
next
  case ImpL
  thus ?case by (metis SCp.ImpL diff-single-trivial diff-union-swap2)
next
  case ImpR
  thus ?case
    by (metis SCp.ImpR diff-single-trivial diff-union-swap diff-union-swap2 formula.distinct(17))
qed
corollary Bot-delR-simp:  $\Gamma \Rightarrow \perp, \Delta = \Gamma \Rightarrow \Delta$ 
  using Bot-delR weakenR by fastforce

```

```

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(induction F arbitrary:  $\Gamma \Delta$ )
  case (Atom k)
  have Atom k, Atom k,  $\Gamma \Rightarrow \Delta \Longrightarrow$  Atom k,  $\Gamma \Rightarrow \Delta$ 
    by(induction Atom k, Atom k,  $\Gamma \Delta$  arbitrary:  $\Gamma$  rule: SCp.induct)
    (auto dest!: multi-member-split intro!: SCp.intros(3-) simp add: lem2 lem1 SCp.intros)
  moreover have  $\Gamma \Rightarrow$  Atom k, Atom k,  $\Delta \Longrightarrow$   $\Gamma \Rightarrow$  Atom k,  $\Delta$ 
    by(induction  $\Gamma$  Atom k, Atom k,  $\Delta$  arbitrary:  $\Delta$  rule: SCp.induct)
    (auto dest!: multi-member-split intro!: SCp.intros(3-) simp add: lem2 lem1 SCp.intros)
  ultimately show ?case by blast
next
  case Bot
  have  $\perp, \perp, \Gamma \Rightarrow \Delta \Longrightarrow$   $\perp, \Gamma \Rightarrow \Delta$  by(induction  $\perp, \perp, \Gamma \Delta$  rule: SCp.induct; blast)
  moreover have ( $\Gamma \Rightarrow \perp, \perp, \Delta \Longrightarrow$   $\Gamma \Rightarrow \perp, \Delta$ )
    using Bot-delR by fastforce
  ultimately show ?case by blast
next
  case (Not F)
  then show ?case by (meson NotL-inv NotR-inv SCp.NotL SCp.NotR)
next
  case (And F1 F2) then show ?case by(auto intro!: SCp.intros(3-) dest!: AndR-inv AndL-inv) (metis add-mset-commute)
next
  case (Or F1 F2) then show ?case by(auto intro!: SCp.intros(3-) dest!: OrR-inv

```

OrL-inv (*metis add-mset-commute*)
next

case (*Imp F1 F2*) **show** ?*case* **by** (*auto dest!*: *ImpR-inv ImpL-inv intro!*: *SCp.intros(3-)*)
(*insert Imp.IH*; *blast*)**+**

qed

corollary

shows *contractL*: $F, F, \Gamma \Rightarrow \Delta \Longrightarrow F, \Gamma \Rightarrow \Delta$
and *contractR*: $\Gamma \Rightarrow F, F, \Delta \Longrightarrow \Gamma \Rightarrow F, \Delta$
using *contract* **by** *blast***+**

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*: $\Gamma \Rightarrow (A::'a \text{ formula}), \Delta \Longrightarrow A, \Gamma' \Rightarrow \Delta' \Longrightarrow \Gamma + \Gamma' \Rightarrow$
 $\Delta + \Delta'$

by (*intro context-sharing-Cut*[*of* $\Gamma + \Gamma' A \Delta + \Delta'$])

(*metis add commute union-mset-add-mset-left weakenL-set weakenR-set*)**+**

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 –

have *contract* $\Gamma\Gamma$: $\Gamma + \Gamma' \Rightarrow \Delta \Longrightarrow \Gamma' \subseteq\# \Gamma \Longrightarrow \Gamma \Rightarrow \Delta$ **for** $\Gamma \Gamma' \Delta$

proof (*induction* Γ' *arbitrary*: Γ)

case *empty* **thus** ?*case* **using** *weakenL-set* **by** *force*

next

case (*add* $x \Gamma' \Gamma$)

from *add.premis*(2) **have** $x \in\# \Gamma$ **by** (*simp add: insert-subset-eq-iff*)

then **obtain** Γ'' **where** Γ [*simp*]: $\Gamma = x, \Gamma''$ **by** (*meson multi-member-split*)

from *add.premis*(1) **have** $x, x, \Gamma'' + \Gamma' \Rightarrow \Delta$ **by** *simp*

with *contractL* **have** $x, \Gamma'' + \Gamma' \Rightarrow \Delta$.

with *add.IH*[*of* Γ] **show** ?*case* **using** Γ *add.premis*(2) *subset-mset.order.trans*

by *force*

qed

have *contract* $\Delta\Delta$: $\Gamma \Rightarrow \Delta + \Delta' \Longrightarrow \Delta' \subseteq\# \Delta \Longrightarrow \Gamma \Rightarrow \Delta$ **for** $\Gamma \Delta \Delta'$

proof (*induction* Δ' *arbitrary*: Δ)

case *empty* **thus** ?*case* **using** *weakenL-set* **by** *force*

next

case (*add* $x \Delta' \Delta$)

from *add.premis*(2) **have** $x \in\# \Delta$ **by** (*simp add: insert-subset-eq-iff*)

then **obtain** Δ'' **where** Δ [*simp*]: $\Delta = x, \Delta''$ **by** (*metis multi-member-split*)

from *add.premis*(1) **have** $\Gamma \Rightarrow x, x, \Delta'' + \Delta'$ **by** *simp*

with *contractR* **have** $\Gamma \Rightarrow x, \Delta'' + \Delta'$.

with *add.IH*[*of* Δ] **show** ?*case* **using** Δ *add.premis*(2) *subset-mset.order.trans*

by *force*

```

qed
show  $\Gamma \Rightarrow A, \Delta \Longrightarrow A, \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta$ 
  using context-free-Cut[of  $\Gamma A \Delta \Gamma \Delta$ ] contract $\Gamma\Gamma$  contract $\Delta\Delta$ 
  by blast
qed

```

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(induction Atom  $k, \Gamma \Delta$  arbitrary:  $\Gamma$  rule: SCp.induct)
  case BotL
  hence  $\perp \in \# \Gamma$  by simp
  thus ?case using SCp.BotL by blast
next
  case (Ax  $l \Delta$ )
  show ?case proof cases
    assume  $l = k$ 
    with  $\langle Atom\ l \in \# \Delta \rangle$  obtain  $\Delta'$  where  $\Delta = Atom\ k, \Delta'$  by (meson multi-member-split)
    with  $\langle \Gamma \Rightarrow Atom\ k, \Delta \rangle$  have  $\Gamma \Rightarrow \Delta$  using contractR by blast
    thus ?thesis by auto
  next
    assume  $l \neq k$ 
    with  $\langle Atom\ l \in \# Atom\ k, \Gamma \rangle$  have  $Atom\ l \in \# \Gamma$  by simp
    with  $\langle Atom\ l \in \# \Delta \rangle$  show ?thesis using SCp.Ax[of  $l$ ] by blast
  qed
next
  case NotL
  thus ?case by(auto simp: add-eq-conv-ex intro!: SCp.NotL dest!: NotL-inv)
next
  case NotR
  then show ?case by(auto intro!: SCp.NotR dest!: NotR-inv)
next
  case AndL
  thus ?case by(auto simp: add-eq-conv-ex intro!: SCp.AndL dest!: AndL-inv)
next
  case AndR
  then show ?case by(auto intro!: SCp.AndR dest!: AndR-inv)
next
  case OrL
  thus ?case by(auto simp: add-eq-conv-ex intro!: SCp.OrL dest!: OrL-inv)
next
  case OrR
  thus ?case by(auto intro!: SCp.OrR dest!: OrR-inv)
next
  case ImpL
  thus ?case by(auto simp: add-eq-conv-ex intro!: SCp.ImpL dest!: ImpL-inv)
next
  case ImpR
  then show ?case by (auto dest!: ImpR-inv intro!: SCp.ImpR)

```

qed

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(*induction F arbitrary: $\Gamma \Delta$*)

case Atom thus ?case using Cut-Atom-pre by metis

next

case Bot thus ?case using Bot-delR by fastforce

next

case Not with NotL-inv NotR-inv show ?case by blast

next

case And thus ?case by (meson AndL-inv AndR-inv weakenL)

next

case Or thus ?case by (metis OrL-inv OrR-inv weakenR)

next

case (Imp F G)

from ImpR-inv $\langle \Gamma \Rightarrow F \rightarrow G, \Delta \rangle$ have R: $F, \Gamma \Rightarrow G, \Delta$ by blast

from ImpL-inv $\langle F \rightarrow G, \Gamma \Rightarrow \Delta \rangle$ have L: $\Gamma \Rightarrow F, \Delta G, \Gamma \Rightarrow \Delta$ by blast+

from L(1) have $\Gamma \Rightarrow F, G, \Delta$ using weakenR add-ac(3) by blast

with R have $\Gamma \Rightarrow G, \Delta$ using Imp.IH(1) by simp

with L(2) show $\Gamma \Rightarrow \Delta$ using Imp.IH(2) by clarsimp

qed

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

using cut-cs-cf cut by metis

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: $F, F, \Gamma \Rightarrow \Delta \Longrightarrow (F::'a \text{ formula}), \Gamma \Rightarrow \Delta$

by (rule cut[of F, Γ F Δ , OF extended-Ax]) simp-all

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: $\perp \in \# \Gamma \Longrightarrow \Gamma \Rightarrow \Delta \downarrow \text{Suc } n \mid$
Ax: $\text{Atom } k \in \# \Gamma \Longrightarrow \text{Atom } k \in \# \Delta \Longrightarrow \Gamma \Rightarrow \Delta \downarrow \text{Suc } n \mid$
NotL: $\Gamma \Rightarrow F, \Delta \downarrow n \Longrightarrow \text{Not } F, \Gamma \Rightarrow \Delta \downarrow \text{Suc } n \mid$
NotR: $F, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow \text{Not } F, \Delta \downarrow \text{Suc } n \mid$
AndL: $F, G, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow \text{And } F G, \Gamma \Rightarrow \Delta \downarrow \text{Suc } n \mid$
AndR: $[\Gamma \Rightarrow F, \Delta \downarrow n; \Gamma \Rightarrow G, \Delta \downarrow n] \Longrightarrow \Gamma \Rightarrow \text{And } F G, \Delta \downarrow \text{Suc } n \mid$
OrL: $[F, \Gamma \Rightarrow \Delta \downarrow n; G, \Gamma \Rightarrow \Delta \downarrow n] \Longrightarrow \text{Or } F G, \Gamma \Rightarrow \Delta \downarrow \text{Suc } n \mid$
OrR: $\Gamma \Rightarrow F, G, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow \text{Or } F G, \Delta \downarrow \text{Suc } n \mid$
ImpL: $[\Gamma \Rightarrow F, \Delta \downarrow n; G, \Gamma \Rightarrow \Delta \downarrow n] \Longrightarrow \text{Imp } F G, \Gamma \Rightarrow \Delta \downarrow \text{Suc } n \mid$
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!*]: $\text{Atom } k, \Gamma \Rightarrow \text{Atom } k, \Delta \downarrow \text{Suc } n$
by (*meson SCc.intros union-single-eq-member*)+

lemma *deeper*: $\Gamma \Rightarrow \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow \Delta \downarrow n + m$

by (*induction rule: SCc.induct; insert SCc.intros; auto*)

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

thm *deeper*[*unfolded Suc-eq-plus1[symmetric]*]
by (*drule deeper[where m=1] simp*)

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

assume $?c$ **then guess** n ..

thus $?p$ **by** (*induction rule: SCc.induct; simp add: SCp.intros*)

next

have *deeper-max*: $\Gamma \Rightarrow \Delta \downarrow \text{max } m n \Gamma \Rightarrow \Delta \downarrow \text{max } n m$ **if** $\Gamma \Rightarrow \Delta \downarrow n$ **for** $n m$
and $\Gamma \Delta$:: 'a formula multiset

proof -

have $n \leq m \Longrightarrow \exists k. m = n + k$ **by** *presburger*

with *that[THEN deeper]* **that**

show $\Gamma \Rightarrow \Delta \downarrow \text{max } n m$ **unfolding** *max-def* **by** *clarsimp*

thus $\Gamma \Rightarrow \Delta \downarrow \text{max } m n$ **by** (*simp add: max commute*)

qed

assume $?p$ **thus** $?c$ **by** (*induction rule: SCp.induct*)

(*insert SCc.intros[where 'a='a] deeper-max; metis*)+

qed

lemma *no-0-SC[dest!]*: $\Gamma \Rightarrow \Delta \downarrow 0 \Longrightarrow False$
by(cases rule: *SCc.cases[of $\Gamma \Delta 0$]*) *assumption*

lemma *inR1'*: $\Gamma \Rightarrow G, H, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow H, G, \Delta \downarrow n$ **by**(simp add: *add-mset-commute*)

lemma *inL1'*: $G, H, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow H, G, \Gamma \Rightarrow \Delta \downarrow n$ **by**(simp add: *add-mset-commute*)

lemma *inR2'*: $\Gamma \Rightarrow F, G, H, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow G, H, F, \Delta \downarrow n$ **by**(simp add: *add-mset-commute*)

lemma *inL2'*: $F, G, H, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow G, H, F, \Gamma \Rightarrow \Delta \downarrow n$ **by**(simp add: *add-mset-commute*)

lemma *inR3'*: $\Gamma \Rightarrow F, G, H, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow H, F, G, \Delta \downarrow n$ **by**(simp add: *add-mset-commute*)

lemma *inR4'*: $\Gamma \Rightarrow F, G, H, I, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow H, I, F, G, \Delta \downarrow n$ **by**(simp add: *add-mset-commute*)

lemma *inL3'*: $F, G, H, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow H, F, G, \Gamma \Rightarrow \Delta \downarrow n$ **by**(simp add: *add-mset-commute*)

lemma *inL4'*: $F, G, H, I, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow H, I, F, G, \Gamma \Rightarrow \Delta \downarrow n$ **by**(simp add: *add-mset-commute*)

lemmas *SC-swap-applies[intro,elim!]* = *inL1' inL2' inL3' inL4' inR1' inR2' inR3' inR4'*

lemma *Atom C \rightarrow Atom D \rightarrow Atom E,*
Atom k \rightarrow Atom C \wedge Atom D,
Atom k, {#}
 $\Rightarrow \{ \# \text{ Atom E } \# \} \downarrow \text{Suc (Suc (Suc (Suc (Suc 0))))}$
by(auto intro!: *SCc.intros(3-)* intro: *SCc.intros(1,2)*)

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

proof(induction rule: *SCc.induct*)

case *BotL* **thus** ?case **by**(rule *SCc.BotL*; simp)

next case (*Ax k*) **thus** ?case **by**(intro *SCc.Ax*[of *k*]; simp; metis *diff-single-trivial formula.distinct(1) insert-DiffM lem1*)

next case *NotL* **thus** ?case **using** *SCc.NotL* **by** (metis *add-mset-remove-trivial diff-single-trivial diff-union-swap insert-DiffM*)

next case *NotR* **thus** ?case **using** *SCc.NotR* **by** (metis *diff-union-swap formula.distinct(11)*)

next case *AndR* **thus** ?case **using** *SCc.AndR* **by** (metis *diff-single-trivial diff-union-swap diff-union-swap2 formula.distinct(13)*)

next case *OrR* **thus** ?case **using** *SCc.OrR* **by** (metis *diff-single-trivial diff-union-swap2 formula.distinct(15) insert-iff set-mset-add-mset-insert*)

next case *ImpL* **thus** ?case **using** *SCc.ImpL* **by** (metis *diff-single-trivial diff-union-swap2*)

next case *ImpR* **thus** ?case **using** *SCc.ImpR* **by** (metis *diff-single-trivial diff-union-swap diff-union-swap2 formula.distinct(17)*)

qed (simp-all add: *SCc.intros*)

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

proof(*induction Not F, $\Gamma \Delta n$ arbitrary: Γ rule: SCc.induct*)
case (*NotL $\Gamma' G \Delta n$*) **thus** ?*case* **by**(*cases Not F = Not G*)
(*auto intro!: SCc.intros(3-) dest!: multi-member-split simp: lem1 lem2 deeper-suc*)
qed (*auto intro!: SCc.intros(3-) dest!: multi-member-split simp: SCp.intros lem1 lem2*)

lemma *AndL-inv'*: *And F G, $\Gamma \Rightarrow \Delta \downarrow n \Longrightarrow F, G, \Gamma \Rightarrow \Delta \downarrow n$*
proof(*induction And F G, $\Gamma \Delta n$ arbitrary: Γ rule: SCc.induct*)
case (*AndL F' G' $\Gamma' \Delta$*) **thus** ?*case*
by(*cases And F G = And F' G'*;
auto intro!: SCc.intros(3-) dest!: multi-member-split simp: lem1 lem2 deeper-suc;
metis add-mset-commute)
qed (*auto intro!: SCc.intros(3-) dest!: multi-member-split simp: SCc.intros lem1 lem2 inL2'*)

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(*insert assms, induction Or F G, $\Gamma \Delta n$ arbitrary: Γ rule: SCc.induct*)
case (*OrL F' $\Gamma' \Delta n G'$*) **thus** ?*case*
by(*cases Or F G = Or F' G'*;
auto intro!: SCc.intros(3-) dest!: multi-member-split simp: lem1 lem2 deeper-suc;
blast)
qed (*fastforce intro!: SCc.intros(3-) dest!: multi-member-split simp: SCc.intros lem1 lem2*)+

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(*insert assms, induction Imp F G, $\Gamma \Delta n$ arbitrary: Γ rule: SCc.induct*)
case (*ImpL $\Gamma' F' \Delta n G'$*) **thus** ?*case*
by(*cases Or F G = Or F' G'*;
auto intro!: SCc.intros(3-) dest!: multi-member-split simp: lem1 lem2 deeper-suc;
blast)
qed (*fastforce intro!: SCc.intros(3-) dest!: multi-member-split simp: SCc.intros lem1 lem2*)+

lemma *ImpR-inv'*:
assumes *$\Gamma \Rightarrow Imp F G, \Delta \downarrow n$*
shows *F, $\Gamma \Rightarrow G, \Delta \downarrow n$*
proof(*insert assms, induction $\Gamma Imp F G, \Delta n$ arbitrary: Δ rule: SCc.induct*)
case (*ImpR F' $\Gamma G' \Delta'$*) **thus** ?*case*
by(*cases Or F G = Or F' G'*;
auto intro!: SCc.intros(3-) dest!: multi-member-split simp: lem1 lem2 deeper-suc;
blast)

qed (*fastforce* *intro!*: *SCc.intros*(3-) *dest!*: *multi-member-split simp*: *SCc.intros lem1 lem2*)+

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(*induction* Γ *And* $F \ G, \Delta \ n$ *arbitrary*: Δ *rule*: *SCc.induct*)

case (*AndR* $\Gamma \ F' \ \Delta' \ n \ G'$) **thus** ?*case*

by(*cases* $\text{Or } F \ G = \text{Or } F' \ G'$;

auto *intro!*: *SCc.intros*(3-) *dest!*: *multi-member-split simp*: *lem1 lem2*

deeper-suc;

blast)

qed (*fastforce* *intro!*: *SCc.intros*(3-) *dest!*: *multi-member-split simp*: *SCc.intros lem1 lem2*)+

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

proof(*induction* Γ *Or* $F \ G, \Delta \ n$ *arbitrary*: Δ *rule*: *SCc.induct*)

case (*OrR* $\Gamma \ F' \ G' \ \Delta'$) **thus** ?*case*

by(*cases* $\text{Or } F \ G = \text{Or } F' \ G'$;

auto *intro!*: *SCc.intros*(3-) *dest!*: *multi-member-split simp*: *lem1 lem2*

deeper-suc;

metis *add-mset-commute*)

qed (*fastforce* *intro!*: *SCc.intros*(3-) *dest!*: *multi-member-split simp*: *SCc.intros lem1 lem2*)+

lemma *NotR-inv'*: $\Gamma \Rightarrow \text{Not } F, \Delta \downarrow n \Longrightarrow F, \Gamma \Rightarrow \Delta \downarrow n$

proof(*induction* Γ *Not* $F, \Delta \ n$ *arbitrary*: Δ *rule*: *SCc.induct*)

case (*NotR* $G \ \Gamma \ \Delta'$) **thus** ?*case*

by(*cases* $\text{Not } F = \text{Not } G$;

auto *intro!*: *SCc.intros*(3-) *dest!*: *multi-member-split simp*: *lem1 lem2*

deeper-suc;

metis *add-mset-commute*)

qed (*fastforce* *intro!*: *SCc.intros*(3-) *dest!*: *multi-member-split simp*: *SCc.intros lem1 lem2*)+

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

by(*induction* *rule*: *SCc.induct*)

(*auto* *intro!*: *SCc.intros*(3-) *intro*: *SCc.intros*(1,2))

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

by(*induction* *rule*: *SCc.induct*)

(*auto* *intro!*: *SCc.intros*(3-) *intro*: *SCc.intros*(1,2))

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(*induction* n *arbitrary*: $F \ \Gamma \ \Delta$)

case (*Suc* n)

hence *IH*: $F, F, \Gamma \Rightarrow \Delta \downarrow n \Longrightarrow F, \Gamma \Rightarrow \Delta \downarrow n \ \Gamma \Rightarrow F, F, \Delta \downarrow n \Longrightarrow \Gamma \Rightarrow F,$

$\Delta \downarrow n$ **for** F :: 'a *formula* **and** $\Gamma \ \Delta$ **by** *blast+*

show ?*case* **proof**(*intro* *conjI* *allI* *impI*, *goal-cases*)

```

case 1
let ?ffs = λΓ. Γ - {# F #} - {# F #}
from 1 show ?case proof(insert 1; cases rule: SCc.cases[of F,F,Γ Δ Suc n])
  case (NotL Γ' G)
  show ?thesis
  proof(cases)
    assume e: F = ¬G
    with NotL have Γ': Γ' = ¬G, Γ by simp
    from NotL-inv' NotL(2) have Γ ⇒ G, G, Δ ↓ n unfolding Γ' .
    with IH(2) have Γ ⇒ G, Δ ↓ n .
    with SCc.NotL show ?thesis unfolding e .
  next
  assume e: F ≠ ¬G
  have ?thesis
    using NotL(2) IH(1)[of F ?ffs Γ' G, Δ] SCc.NotL[of F, Γ' - {# F #}
- {# F #} G Δ n]
    using e NotL(1) by (metis (no-types, lifting) insert-DiffM lem2)
    from e NotL(1) have Γ: Γ = ¬ G, ?ffs Γ' by (meson lem1)
    with NotL(1) have Γ': F,F,?ffs Γ' = Γ' by simp
    show ?thesis using NotL(2) IH(1)[of F ?ffs Γ' G, Δ] SCc.NotL[of F, ?ffs
Γ' G Δ n] (F, Γ ⇒ Δ ↓ Suc n) by blast
  qed
next
case (AndL G H Γ') show ?thesis proof cases
  assume e: F = And G H
  with AndL(1) have Γ': Γ' = And G H, Γ by simp
  have G ∧ H, G, H, Γ ⇒ Δ ↓ n using AndL(2) unfolding Γ' by auto
  hence G, H, G, H, Γ ⇒ Δ ↓ n by(rule AndL-inv')
  hence G, H, Γ ⇒ Δ ↓ n using IH(1) by (metis inL1' inL3')
  thus F, Γ ⇒ Δ ↓ Suc n using e SCc.AndL by simp
  next
  assume ne: F ≠ And G H
  with AndL(1) have Γ: Γ = And G H, ?ffs Γ' by (metis (no-types, lifting)
diff-diff-add lem2)
  have F, F, G, H, ?ffs Γ' ⇒ Δ ↓ n using AndL(2) using Γ inL4'
local.AndL(1) by auto
  hence G, H, F, ?ffs Γ' ⇒ Δ ↓ n using IH(1) inL2 by blast
  thus ?thesis using SCc.AndL unfolding Γ using inL1 by blast
  qed
next
case (OrL G Γ' H) show ?thesis proof cases
  assume e: F = Or G H
  with OrL(1) have Γ': Γ' = Or G H, Γ by simp
  have Or G H, G, Γ ⇒ Δ ↓ n Or G H, H, Γ ⇒ Δ ↓ n using OrL(2,3)
unfolding Γ' by simp-all
  hence G, G, Γ ⇒ Δ ↓ n H, H, Γ ⇒ Δ ↓ n using OrL-inv' by blast+
  hence G, Γ ⇒ Δ ↓ n H, Γ ⇒ Δ ↓ n using IH(1) by presburger+
  thus F, Γ ⇒ Δ ↓ Suc n unfolding e using SCc.OrL by blast
  next

```

```

    assume ne: F ≠ Or G H
    with OrL(1) have Γ: Γ = Or G H, ?ffs Γ' by (metis (no-types, lifting)
diff-diff-add lem2)
    have F, F, G, ?ffs Γ' ⇒ Δ ↓ n F, F, H, ?ffs Γ' ⇒ Δ ↓ n using OrL Γ by
auto
    hence G, F, ?ffs Γ' ⇒ Δ ↓ n H, F, ?ffs Γ' ⇒ Δ ↓ n using IH(1) by(metis
add-mset-commute)+
    thus ?thesis using SCc.OrL unfolding Γ by auto
qed
next
case (ImpL Γ' G H) show ?thesis proof cases
  assume e: F = Imp G H
  with ImpL(1) have Γ': Γ' = Imp G H, Γ by simp
  have H, Γ ⇒ Δ ↓ n Γ ⇒ G, Δ ↓ n using IH ImpL-inv' ImpL(2,3) unfolding
Γ'
    by (metis add-mset-commute)+
  thus ?thesis unfolding e using SCc.ImpL[where 'a='a] by simp
next
  assume ne: F ≠ Imp G H
  with ImpL(1) have Γ: Γ = Imp G H, ?ffs Γ' by (metis (no-types, lifting)
diff-diff-add lem2)
  have F, F, ?ffs Γ' ⇒ G, Δ ↓ n F, F, H, ?ffs Γ' ⇒ Δ ↓ n using ImpL Γ
by auto
  thus ?thesis using SCc.ImpL IH unfolding Γ by (metis inL1')
qed
next
case ImpR thus ?thesis by (simp add: IH(1) SCc.intros(10) add-mset-commute)
next
  case (NotR G Δ') thus ?thesis using IH(1) by (simp add: SCc.NotR
add-mset-commute)
  qed (auto intro: IH SCc.intros(1,2) intro!: SCc.intros(3-10))
next
  case 2
  let ?ffs = λΓ. Γ - {# F #} - {# F #}
  have not-principal[dest]:
    ⟦F ≠ f G H; F, F, Δ = f G H, Δ'⟧ ⇒ Δ = f G H, ?ffs Δ' ∧ F, F, ?ffs Δ'
= Δ' for f G H Δ' proof(intro conjI, goal-cases)
    case 2
    from 2 have F ∈# Δ' by(blast dest: lem1[THEN iffD1])
    then obtain Δ'' where Δ': Δ' = F, Δ'' by (metis insert-DiffM)
    with 2(2) have F, Δ = f G H, Δ'' by(simp add: add-mset-commute)
    hence F ∈# Δ'' using 2(1) by(blast dest: lem1[THEN iffD1])
    then obtain Δ''' where Δ'': Δ'' = F, Δ''' by (metis insert-DiffM)
    show ?case unfolding Δ' Δ'' by simp
  case 1 show ?case using 1(2) unfolding Δ' Δ'' by(simp add: add-mset-commute)
  qed
  have principal[dest]: F, F, Δ = f G H, Δ' ⇒ F = f G H ⇒ Δ' = f G H,
Δ for f G H Δ' by simp
  from 2 show ?case proof(cases rule: SCc.cases[of Γ F,F,Δ Suc n])

```

```

case (ImpR G H Δ') thus ?thesis proof cases
  assume e[simp]: F = Imp G H
  with ImpR(1) have Δ'[simp]: Δ' = Imp G H, Δ by simp
  have G, Γ ⇒ Imp G H, H, Δ ↓ n using ImpR(2) by simp
  hence G, G, Γ ⇒ H, H, Δ ↓ n by(rule ImpR-inv')
  hence G, Γ ⇒ H, Δ ↓ n using IH by fast
  thus Γ ⇒ F, Δ ↓ Suc n using SCc.ImpR by simp
next
  assume a: F ≠ Imp G H
  with ImpR(1) have Δ: Δ = Imp G H, ?ffs Δ' by (metis (no-types, lifting)
diff-diff-add lem2)
  have G,Γ ⇒ F, F, H, ?ffs Δ' ↓ n using ImpR Δ by fastforce
  thus ?thesis using SCc.ImpR IH unfolding Δ by (metis inR1')
qed
next
case (AndR G Δ' H) thus ?thesis proof(cases F = And G H)
  case True thus ?thesis using AndR by(auto intro!: SCc.intros(3-) dest!:
AndR-inv' intro: IH)
  next
  case False
  hence e: Δ = And G H, ?ffs Δ' using AndR(1) using diff-diff-add lem2
by blast
  hence G ∧ H, F, F, ?ffs Δ' = G ∧ H, Δ' using AndR(1) by simp
  hence Γ ⇒ F, F, G, ?ffs Δ' ↓ n Γ ⇒ F, F, H, ?ffs Δ' ↓ n using AndR(2,3)
using add-left-imp-eq inR2 by fastforce+
  hence Γ ⇒ G, F, ?ffs Δ' ↓ n Γ ⇒ H, F, ?ffs Δ' ↓ n using IH(2) by
blast+
  thus ?thesis unfolding e by(intro SCc.AndR[THEN inR1'])
qed
next
case (OrR G H Δ') thus ?thesis proof cases
  assume a: F = Or G H
  hence Δ': Δ' = G ∨ H, Δ using OrR(1) by(intro principal)
  hence Γ ⇒ G, H, G, H, Δ ↓ n using inR3'[THEN OrR-inv'] OrR(2) by
auto
  hence Γ ⇒ H, G, Δ ↓ n using IH(2)[of Γ G H,H,Δ] IH(2)[of Γ H G,Δ]
  unfolding add-ac(3)[of {#H#} {#G#}] using inR2 by blast
  hence Γ ⇒ G, H, Δ ↓ n by(elim SC-swap-applies)
  thus ?thesis unfolding a by (simp add: SCc.OrR)
next
  assume a: F ≠ Or G H
  with not-principal have np: Δ = G ∨ H, ?ffs Δ' ∧ F, F, ?ffs Δ' = Δ'
using OrR(1) .
  with OrR(2) have Γ ⇒ G, H, F, ?ffs Δ' ↓ n using IH(2) by (metis inR2'
inR4')
  hence Γ ⇒ F, G ∨ H, ?ffs Δ' ↓ Suc n by(intro SCc.OrR[THEN inR1'])
  thus ?thesis using np by simp
qed
next

```

```

case (NotR G Δ') thus ?thesis proof(cases F = Not G)
  case True
    with principal NotR(1) have Δ' = ¬ G, Δ .
    with NotR-inv' NotR(2) have G, G, Γ ⇒ Δ ↓ n by blast
    with IH(1) have G, Γ ⇒ Δ ↓ n .
    thus Γ ⇒ F, Δ ↓ Suc n unfolding True by(intro SCc.NotR)
  next
    case False
      with not-principal have np: Δ = ¬ G, Δ' - (F, {#F#}) ∧ F, F, Δ' -
(F, {#F#}) = Δ' using NotR(1) by auto
      hence G, Γ ⇒ F, F, ?ffs Δ' ↓ n using NotR(2) by simp
      hence G, Γ ⇒ F, ?ffs Δ' ↓ n by(elim IH(2))
      thus ?thesis using np SCc.NotR inR1 by auto
    qed
  next
    case BotL thus ?thesis by(elim SCc.BotL)
  next
    case (Ax k) thus ?thesis by(intro SCc.Ax[where k=k]) simp-all
  next
    case NotL thus ?thesis by (simp add: SCc.NotL Suc.IH add-mset-commute)
  next
    case AndL thus ?thesis using SCc.AndL Suc.IH by blast
  next
    case OrL thus ?thesis using SCc.OrL Suc.IH by blast
  next
    case ImpL thus ?thesis by (metis SCc.ImpL Suc.IH add-mset-commute)
  qed
qed
qed blast

```

```

lemma Cut-Atom-depth: Atom k,Γ ⇒ Δ ↓ n ⇒ Γ ⇒ Atom k,Δ ↓ m ⇒ Γ ⇒
Δ ↓ n + m
proof(induction Atom k,Γ Δ n arbitrary: Γ m rule: SCc.induct)
  case (BotL Δ)
    hence ⊥ ∈# Γ by simp
    thus ?case using SCc.BotL by auto
  next
    case (Ax l Δ)
      show ?case proof cases
        assume l = k
          with ⟨Atom l ∈# Δ⟩ obtain Δ' where Δ = Atom k, Δ' by (meson multi-member-split)
          with ⟨Γ ⇒ Atom k, Δ ↓ m⟩ have Γ ⇒ Δ ↓ m using contract' by blast
          thus ?thesis by (metis add.commute deeper)
        next
          assume l ≠ k
          with ⟨Atom l ∈# Atom k, Γ⟩ have Atom l ∈# Γ by simp
          with ⟨Atom l ∈# Δ⟩ show ?thesis using SCc.Ax[of l] by simp
        qed

```

```

next
  case (NotL  $\Gamma$  F  $\Delta$ )
  obtain  $\Gamma'$  where  $\Gamma: \Gamma = \text{Not } F, \Gamma'$  by (meson NotL.hyps(3) add-eq-conv-ex
formula.simps(9))
  show ?case unfolding  $\Gamma$ 
    apply(unfold plus-nat.add-Suc)
    apply(intro SCc.NotL)
    apply(intro NotL.hyps )
    subgoal using NotL  $\Gamma$  by (simp add: lem2)
    subgoal using  $\Gamma$  NotL.premis NotL-inv' by blast
  done
next
  case (NotR F  $\Delta$ )
  then show ?case by(auto intro!: SCc.NotR dest!: NotR-inv')
next
  case (AndL F G  $\Gamma$   $\Delta$ )
  obtain  $\Gamma'$  where  $\Gamma: \Gamma = \text{And } F G, \Gamma'$  by (metis AndL.hyps(3) add-eq-conv-diff
formula.distinct(5))
  show ?case unfolding  $\Gamma$ 
    apply(unfold plus-nat.add-Suc)
    apply(intro SCc.AndL)
    apply(intro AndL.hyps )
    subgoal using AndL  $\Gamma$  by (simp add: lem2)
    subgoal using  $\Gamma$  AndL.premis AndL-inv' by blast
  done
next
  case (AndR F  $\Delta$  G)
  then show ?case
    using AndR-inv' SCc.AndR by (metis add-Suc inR1')
next
  case (OrL F  $\Gamma'$   $\Delta$  n G)
  obtain  $\Gamma''$  where  $\Gamma: \Gamma = \text{Or } F G, \Gamma''$  by (meson OrL.hyps(5) add-eq-conv-ex
formula.simps(13))
  have ihm: F,  $\Gamma' = \text{Atom } k, F, \Gamma'' G, \Gamma' = \text{Atom } k, G, \Gamma''$  using OrL  $\Gamma$  by
(simp-all add: lem2)
  show ?case unfolding  $\Gamma$ 
    apply(unfold plus-nat.add-Suc)
    apply(intro SCc.OrL OrL.hyps(2)[OF ihm(1)] OrL.hyps(4)[OF ihm(2)])
    subgoal using  $\Gamma$  OrL.premis OrL-inv' by blast
    subgoal using  $\Gamma$  OrL.premis OrL-inv' by blast
  done
next
  case (OrR F G  $\Delta$ )
  then show ?case by(auto intro!: SCc.intros(3-) dest!: OrR-inv')
next
  case (ImpL  $\Gamma' F \Delta n G$ )
  obtain  $\Gamma''$  where  $\Gamma: \Gamma = \text{Imp } F G, \Gamma''$  by (metis ImpL.hyps(5) add-eq-conv-ex
formula.simps)
  show ?case unfolding  $\Gamma$ 

```



```

apply(unfold plus-nat.add-Suc)
apply(intro SCc.ImpL ImpL.hyps(2) ImpL.hyps(4))
  subgoal using ImpL  $\Gamma$  by (simp add: lem2)
  subgoal using  $\Gamma$  ImpL.prems by(auto dest!: ImpL-inv')
  subgoal using ImpL  $\Gamma$  by (simp add: lem2)
  subgoal using  $\Gamma$  ImpL.prems ImpL-inv' by blast
done
next
  case (ImpR F G  $\Delta$ )
  then show ?case by (auto dest!: ImpR-inv' intro!: SCc.ImpR)
qed
primrec cut-bound :: nat  $\Rightarrow$  nat  $\Rightarrow$  'a formula  $\Rightarrow$  nat where
  cut-bound n m (Atom -) = m + n |
  cut-bound n m Bot = n |
  cut-bound n m (Not F) = cut-bound m n F |
  cut-bound n m (And F G) = cut-bound n (cut-bound n m F) G |
  cut-bound n m (Or F G) = cut-bound (cut-bound n m F) m G |
  cut-bound n m (Imp F G) = cut-bound (cut-bound m n F) m G
theorem cut-bound:  $\Gamma \Rightarrow F, \Delta \downarrow n \Longrightarrow F, \Gamma \Rightarrow \Delta \downarrow m \Longrightarrow \Gamma \Rightarrow \Delta \downarrow \text{cut-bound}$ 
n m F
proof(induction F arbitrary:  $\Gamma \Delta n m$ )
  case (Atom k) thus ?case using Cut-Atom-depth by simp fast
next
  case Bot thus ?case using Bot-delR' by fastforce
next
  case Not from Not.prems show ?case by(auto dest!: NotL-inv' NotR-inv' intro!:
Not.IH elim!: weakenL)
next
  case (And F G) from And.prems show ?case by(auto dest!: AndL-inv' AndR-inv'
intro!: And.IH elim!: weakenR' weakenL')
next
  case (Or F G) from Or.prems show ?case by(auto dest: OrL-inv' OrR-inv'
intro!: Or.IH elim!: weakenR' weakenL')
next
  case (Imp F G)
  from ImpR-inv'  $\langle \Gamma \Rightarrow F \rightarrow G, \Delta \downarrow n \rangle$  have R:  $F, \Gamma \Rightarrow G, \Delta \downarrow n$  by blast
  from ImpL-inv'  $\langle F \rightarrow G, \Gamma \Rightarrow \Delta \downarrow m \rangle$  have L:  $\Gamma \Rightarrow F, \Delta \downarrow m$  G,  $\Gamma \Rightarrow \Delta \downarrow m$ 
by blast+
  from L(1) have  $\Gamma \Rightarrow F, G, \Delta \downarrow m$  using weakenR' by blast
  from Imp.IH(1)[OF this R] have  $\Gamma \Rightarrow G, \Delta \downarrow \text{cut-bound } m n F$  .
  from Imp.IH(2)[OF this L(2)] have  $\Gamma \Rightarrow \Delta \downarrow \text{cut-bound } (\text{cut-bound } m n F) m$ 
G .
  thus  $\Gamma \Rightarrow \Delta \downarrow \text{cut-bound } n m (F \rightarrow G)$  by simp
qed

context begin
private primrec cut-bound' :: nat  $\Rightarrow$  'a formula  $\Rightarrow$  nat where
  cut-bound' n (Atom -) =  $2 * n$  |
  cut-bound' n Bot = n |

```

$cut-bound' n (Not F) = cut-bound' n F \mid$
 $cut-bound' n (And F G) = cut-bound' (cut-bound' n F) G \mid$
 $cut-bound' n (Or F G) = cut-bound' (cut-bound' n F) G \mid$
 $cut-bound' n (Imp F G) = cut-bound' (cut-bound' n F) G$

private lemma *cut-bound'-mono*: $a \leq b \implies cut-bound' a F \leq cut-bound' b F$
by(*induction F arbitrary: a b; simp*)

private lemma *cut-bound-mono*: $a \leq c \implies b \leq d \implies cut-bound a b F \leq cut-bound c d F$
by(*induction F arbitrary: a b c d; simp*)

private lemma *cut-bound-max*: $max n (cut-bound' (max n m) F) = cut-bound' (max n m) F$
by(*induction F arbitrary: n m; simp; metis*)

private lemma *cut-bound-max'*: $max n (cut-bound' n F) = cut-bound' n F$
by(*induction F arbitrary: n; simp; metis max.assoc*)

private lemma *cut-bound-'*: $cut-bound n m F \leq cut-bound' (max n m) F$
proof(*induction F arbitrary: n m*)

case (*Not F*)
then show ?*case by simp (metis max.commute)*
next
case (*And F1 F2*)
from *And.IH(1)* **have** 1: $cut-bound n (cut-bound n m F1) F2 \leq cut-bound n (cut-bound' (max n m) F1) F2$
by(*rule cut-bound-mono[OF order.refl]*)
also from *And.IH(2)* **have** ... $\leq cut-bound' (max n (cut-bound' (max n m) F1)) F2$ **by** *simp*
also have ... $= cut-bound' (cut-bound' (max n m) F1) F2$ **by** (*simp add: cut-bound-max*)
finally show ?*case by simp*
next
case (*Or F1 F2*)
from *Or.IH(1)* **have** 1: $cut-bound (cut-bound n m F1) m F2 \leq cut-bound (cut-bound' (max n m) F1) m F2$
by(*rule cut-bound-mono[OF - order.refl]*)
also from *Or.IH(2)*[*of cut-bound' (max n m) F1*] **have** ... $\leq cut-bound' (max (cut-bound' (max n m) F1) m) F2$ **by** *simp*
also have ... $= cut-bound' (cut-bound' (max n m) F1) F2$ **by** (*simp add: cut-bound-max max.commute*)
finally show ?*case by simp*
next
case (*Imp F1 F2*)
from *Imp.IH(1)* **have** 1: $cut-bound (cut-bound m n F1) m F2 \leq cut-bound (cut-bound' (max m n) F1) m F2$
by(*rule cut-bound-mono[OF - order.refl]*)
also from *Imp.IH(2)*[*of cut-bound' (max m n) F1*] **have** ... $\leq cut-bound' (max (cut-bound' (max m n) F1) m) F2$ **by** *simp*

also have ... = *cut-bound'* (*cut-bound'* (*max n m*) *F1*) *F2* **by** (*simp add:*
cut-bound-max max.commute)
finally show ?*case* **by** *simp*
qed *simp-all*

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

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

private lemma *cbnd-grow*: (*k* :: *nat*) \leq *cbnd* *k* *d*
by(*induction* *d* *arbitrary*: *k*; *simp*) (*insert le-trans, blast*)

private lemma *cbnd-mono*: **assumes** $b \leq d$ **shows** *cbnd* (*a*::*nat*) *b* \leq *cbnd* *a* *d*
proof –
have *cbnd* (*a*::*nat*) *b* \leq *cbnd* *a* (*b* + *d*) **for** *b* *d*
by(*induction* *d* *arbitrary*: *a* *b*; *simp*) (*insert le-trans cbnd-grow, blast*)
thus ?*thesis* **using** *assms* **using** *le-Suc-ex* **by** *blast*
qed

private lemma *cut-bound'-cbnd*: *cut-bound'* *n* *F* \leq *cbnd* *n* (*depth* *F*)
proof(*induction* *F* *arbitrary*: *n*)
next
case (*Not* *F*)
then show ?*case* **using** *cbnd-grow dual-order.trans* **by** *fastforce*
next
case (*And* *F1* *F2*)
let ?*md* = *max* (*depth* *F1*) (*depth* *F2*)
have *cut-bound'* (*cut-bound'* *n* *F1*) *F2* \leq *cut-bound'* (*cbnd* *n* (*depth* *F1*)) *F2* **by**
(*simp add: And.IH(1) cut-bound'-mono*)
also have ... \leq *cut-bound'* (*cbnd* *n* ?*md*) *F2* **by** (*simp add: cbnd-mono cut-bound'-mono*)
also have ... \leq *cbnd* (*cbnd* *n* ?*md*) (*depth* *F2*) **using** *And.IH(2)* **by** *blast*
also have ... \leq *cbnd* (*cbnd* *n* ?*md*) ?*md* **by** (*simp add: cbnd-mono*)
finally show ?*case* **by** *simp*
next
case (*Imp* *F1* *F2*)
case (*Or* *F1* *F2*)

analogous

qed *simp-all*

value *map* (*cbnd* (0::*int*)) [0,1,2,3,4]

```

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

```

```

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

```

```

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

```

```

lemma n + ((n + m) * 2 ^ (size F - Suc 0) +
(n + (n + m + (n + m) * 2 ^ (size F - Suc 0))) * 2 ^ (size G - Suc 0))
≤ (n + (m :: nat)) * 2 ^ (size F + size G)
oops

```

```

lemma cut-bound (n :: nat) m F ≤ (n + m) * (2 ^ (size F - 1) + 1)

```

```

proof(induction F arbitrary: n m)

```

```

next

```

```

  case (Not F)

```

```

  show ?case unfolding cut-bound.simps by(rule le-trans[OF Not]) (simp add:
add commute)

```

```

next

```

```

  have 1 ≤ size F for F :: 'a formula by(cases F; simp)

```

```

  case (And F G)

```

```

  from And(2) have cut-bound n (cut-bound n m F) G ≤ (n + (cut-bound n m
F)) * (2 ^ (size G - 1) + 1) by simp

```

```

  also from And(1) have ... ≤ (n + (n + m) * (2 ^ (size F - 1) + 1)) * (2 ^
(size G - 1) + 1)

```

```

    by (meson add-le-cancel-left mult-le-mono1)

```

```

  also have ... ≤ (n + m) * (2 ^ (size (F ∧ G) - 1) + 1)

```

```

    apply simp

```

```

    oops

```

```

private lemma cbnd-comm: cbnd (l * k::nat) n = l * cbnd (k::nat) n

```

```

by(induction n arbitrary: k; simp)

```

```

private lemma cbnd-closed: cbnd (k::nat) n = k * 2 ^ (2 ^ n)

```

```

by(induction n arbitrary: k; simp add: semiring-normalization-rules(26))

```

```

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

```

```

proof -

```

```

  from cut-bound[OF assms] have c: Γ ⇒ Δ ↓ cut-bound n n F .

```

```

have  $d$ : cut-bound  $n$   $n$   $F \leq \max n n * 2 \wedge (2 \wedge \text{depth } F)$ 
using cut-bound-' cut-bound'-cbnd cbnd-closed by (metis order-trans)
show ?thesis using c d le-Suc-ex deeper unfolding max.idem by metis
qed

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:  $[\mathfrak{D}] \Rightarrow [\mathfrak{D}]$  |
  FalschA:  $[\perp] \Rightarrow []$  |
  VerduennungA:  $\Gamma \Rightarrow \Theta \Longrightarrow \mathfrak{D}\#\Gamma \Rightarrow \Theta$  |
  VerduennungS:  $\Gamma \Rightarrow \Theta \Longrightarrow \Gamma \Rightarrow \mathfrak{D}\#\Theta$  |
  ZusammenziehungA:  $\mathfrak{D}\#\mathfrak{D}\#\Gamma \Rightarrow \Theta \Longrightarrow \mathfrak{D}\#\Gamma \Rightarrow \Theta$  |
  ZusammenziehungS:  $\Gamma \Rightarrow \mathfrak{D}\#\mathfrak{D}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathfrak{D}\#\Theta$  |
  VertauschungA:  $\Delta@\mathfrak{D}\#\mathfrak{E}\#\Gamma \Rightarrow \Theta \Longrightarrow \Delta@\mathfrak{E}\#\mathfrak{D}\#\Gamma \Rightarrow \Theta$  |
  VertauschungS:  $\Gamma \Rightarrow \Theta@\mathfrak{E}\#\mathfrak{D}\#\Lambda \Longrightarrow \Gamma \Rightarrow \Theta@\mathfrak{D}\#\mathfrak{E}\#\Lambda$  |
  Schnitt:  $[\Gamma \Rightarrow \mathfrak{D}\#\Theta; \mathfrak{D}\#\Delta \Rightarrow \Lambda] \Longrightarrow \Gamma@\Delta \Rightarrow \Theta@\Lambda$  |
  UES:  $[\Gamma \Rightarrow \mathfrak{A}\#\Theta; \Gamma \Rightarrow \mathfrak{B}\#\Theta] \Longrightarrow \Gamma \Rightarrow \mathfrak{A}\wedge\mathfrak{B}\#\Theta$  |
  UEA1:  $\mathfrak{A}\#\Gamma \Rightarrow \Theta \Longrightarrow \mathfrak{A}\wedge\mathfrak{B}\#\Gamma \Rightarrow \Theta$  | UEA2:  $\mathfrak{B}\#\Gamma \Rightarrow \Theta \Longrightarrow \mathfrak{A}\wedge\mathfrak{B}\#\Gamma \Rightarrow \Theta$  |
  OEA:  $[\mathfrak{A}\#\Gamma \Rightarrow \Theta; \mathfrak{B}\#\Gamma \Rightarrow \Theta] \Longrightarrow \mathfrak{A}\vee\mathfrak{B}\#\Gamma \Rightarrow \Theta$  |
  OES1:  $\Gamma \Rightarrow \mathfrak{A}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathfrak{A}\vee\mathfrak{B}\#\Theta$  | OES2:  $\Gamma \Rightarrow \mathfrak{B}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathfrak{A}\vee\mathfrak{B}\#\Theta$  |
  FES:  $\mathfrak{A}\#\Gamma \Rightarrow \mathfrak{B}\#\Theta \Longrightarrow \Gamma \Rightarrow \mathfrak{A}\rightarrow\mathfrak{B}\#\Theta$  |
  FEA:  $[\Gamma \Rightarrow \mathfrak{A}\#\Theta; \mathfrak{B}\#\Delta \Rightarrow \Lambda] \Longrightarrow \mathfrak{A}\rightarrow\mathfrak{B}\#\Gamma@\Delta \Rightarrow \Theta@\Lambda$  |
  NES:  $\mathfrak{A}\#\Gamma \Rightarrow \Theta \Longrightarrow \Gamma \Rightarrow \neg\mathfrak{A}\#\Theta$  |
  NEA:  $\Gamma \Rightarrow \mathfrak{A}\#\Theta \Longrightarrow \neg\mathfrak{A}\#\Gamma \Rightarrow \Theta$ 

```

Nota bene: E here stands for “Einführung”, which is introduction and not 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:  $\mathfrak{D}\#\Delta@\Gamma \Rightarrow \Theta \Longrightarrow \Delta@\mathfrak{D}\#\Gamma \Rightarrow \Theta$ 
by(induction  $\Delta$  arbitrary:  $\Gamma$  rule: List.rev-induct) (simp-all add: VertauschungA)
lemma AnfangTauschS:  $\Gamma \Rightarrow \mathfrak{D}\#\Delta@\Theta \Longrightarrow \Gamma \Rightarrow \Delta@\mathfrak{D}\#\Theta$ 

```

by(*induction* Δ *arbitrary*: Θ *rule*: *List.rev-induct*) (*simp-all add*: *VertauschungS*)
lemma *MittenTauschA*: $\Delta @ \mathfrak{D} \# \Gamma \Rightarrow \Theta \Longrightarrow \mathfrak{D} \# \Delta @ \Gamma \Rightarrow \Theta$
 by(*induction* Δ *arbitrary*: Γ *rule*: *List.rev-induct*) (*simp-all add*: *VertauschungA*)
lemma *MittenTauschS*: $\Gamma \Rightarrow \Delta @ \mathfrak{D} \# \Theta \Longrightarrow \Gamma \Rightarrow \mathfrak{D} \# \Delta @ \Theta$
 by(*induction* Δ *arbitrary*: Θ *rule*: *List.rev-induct*) (*simp-all add*: *VertauschungS*)

lemma *BotLe*: $\perp \in \text{set } \Gamma \Longrightarrow \Gamma \Rightarrow \Delta$

proof –

have $A: \perp \# \Gamma \Rightarrow []$ for Γ by(*induction* Γ) (*simp-all add*: *FalschA VerduennungA*
VertauschungA[**where** $\Delta = \text{Nil}$, *simplified*])
 have $*$: $\perp \# \Gamma \Rightarrow \Delta$ for Γ by(*induction* Δ) (*simp-all add*: *A VerduennungS*)
 assume $\perp \in \text{set } \Gamma$ then obtain $\Gamma 1 \ \Gamma 2$ **where** $\Gamma: \Gamma = \Gamma 1 @ \perp \# \Gamma 2$ by (*meson*
split-list)
 show *?thesis* **unfolding** Γ **using** *AnfangTauschA* * **by** *blast*
qed

lemma *Axe*: $A \in \text{set } \Gamma \Longrightarrow A \in \text{set } \Delta \Longrightarrow \Gamma \Rightarrow \Delta$

proof –

have $A: A \# \Gamma \Rightarrow [A]$ for Γ by(*induction* Γ) (*simp-all add*: *Anfang VertauschungA*[**where**
 $\Delta = \text{Nil}$, *simplified*] *VerduennungA*)
 have $S: A \# \Gamma \Rightarrow A \# \Delta$ for $\Gamma \ \Delta$ by(*induction* Δ) (*simp-all add*: *A Anfang*
VertauschungS[**where** $\Theta = \text{Nil}$, *simplified*] *VerduennungS*)
 assume $A \in \text{set } \Gamma \ A \in \text{set } \Delta$ **thus** *?thesis*
 apply(–)
 apply(*drule split-list*)
 apply(*clarify*)
 apply(*intro AnfangTauschA AnfangTauschS*)
 apply(*rule S*)
done
qed

lemma *VerduennungListeA*: $\Gamma \Rightarrow \Theta \Longrightarrow \Gamma @ \Gamma \Rightarrow \Theta$

proof –

have $\Gamma \Rightarrow \Theta \Longrightarrow \exists \Gamma''. \Gamma = \Gamma'' @ \Gamma' \Longrightarrow \Gamma' @ \Gamma \Rightarrow \Theta$ for Γ'
proof(*induction* Γ')
 case (*Cons a as*)
 from $\langle \exists \Gamma''. \Gamma = \Gamma'' @ a \# as \rangle$ **guess** Γ'' ..
 hence $\exists \Gamma''. \Gamma = \Gamma'' @ as$ by(*intro exI*[**where** $x = \Gamma'' @ [a]$] *simp*)
 from *Cons.IH*[*OF Cons.prem*(1) *this*] **have** $as @ \Gamma \Rightarrow \Theta$.
thus *?case* **using** *VerduennungA* **by** *simp*
qed *simp*
thus $\Gamma \Rightarrow \Theta \Longrightarrow \Gamma @ \Gamma \Rightarrow \Theta$ **by** *simp*

qed

lemma *VerduennungListeS*: $\Gamma \Rightarrow \Theta \Longrightarrow \Gamma \Rightarrow \Theta @ \Theta$

proof –

have $\Gamma \Rightarrow \Theta \Longrightarrow \exists \Theta''. \Theta = \Theta'' @ \Theta' \Longrightarrow \Gamma \Rightarrow \Theta' @ \Theta$ for Θ'
proof(*induction* Θ')
 case (*Cons a as*)
 from $\langle \exists \Theta''. \Theta = \Theta'' @ a \# as \rangle$ **guess** Θ'' ..

hence $\exists \Theta'' . \Theta = \Theta'' @ as$ by (intro exI[where $x=\Theta'' @ [a]$]) simp
 from Cons.IH[OF Cons.prem(1) this] have $\Gamma \Rightarrow as @ \Theta$.
 thus ?case using VerduennungS by simp
 qed simp
 thus $\Gamma \Rightarrow \Theta \Longrightarrow \Gamma \Rightarrow \Theta @ \Theta$ by simp
 qed

lemma ZusammenziehungListeA: $\Gamma @ \Gamma \Rightarrow \Theta \Longrightarrow \Gamma \Rightarrow \Theta$

proof –

have $\Gamma' @ \Gamma \Rightarrow \Theta \Longrightarrow \exists \Gamma'' . \Gamma = \Gamma'' @ \Gamma' \Longrightarrow \Gamma \Rightarrow \Theta$ for Γ'

proof(induction Γ')

case (Cons a Γ')

from $\exists \Gamma'' . \Gamma = \Gamma'' @ a \# \Gamma'$ guess Γ'' .. note $\Gamma'' = this$

then obtain $\Gamma 1 \ \Gamma 2$ where $\Gamma : \Gamma = \Gamma 1 @ a \# \Gamma 2$ by blast

from Γ'' have **: $\exists \Gamma'' . \Gamma = \Gamma'' @ \Gamma'$ by (intro exI[where $x=\Gamma'' @ [a]$]) simp

from Cons.prem(1) have $a \# (a \# \Gamma') @ \Gamma 1 @ \Gamma 2 \Rightarrow \Theta$ unfolding Γ using MittenTauschA by (metis append-assoc)

hence $(a \# \Gamma') @ \Gamma 1 @ \Gamma 2 \Rightarrow \Theta$ using ZusammenziehungA by auto

hence $\Gamma' @ \Gamma \Rightarrow \Theta$ unfolding Γ using AnfangTauschA by (metis append-Cons append-assoc)

from Cons.IH[OF this **] show $\Gamma \Rightarrow \Theta$.

qed simp

thus $\Gamma @ \Gamma \Rightarrow \Theta \Longrightarrow \Gamma \Rightarrow \Theta$ by simp

qed

lemma ZusammenziehungListeS: $\Gamma \Rightarrow \Theta @ \Theta \Longrightarrow \Gamma \Rightarrow \Theta$

proof –

have $\Gamma \Rightarrow \Theta' @ \Theta \Longrightarrow \exists \Theta'' . \Theta = \Theta'' @ \Theta' \Longrightarrow \Gamma \Rightarrow \Theta$ for Θ'

proof(induction Θ')

case (Cons a Θ')

from $\exists \Theta'' . \Theta = \Theta'' @ a \# \Theta'$ guess Θ'' .. note $\Theta'' = this$

then obtain $\Theta 1 \ \Theta 2$ where $\Theta : \Theta = \Theta 1 @ a \# \Theta 2$ by blast

from Θ'' have **: $\exists \Theta'' . \Theta = \Theta'' @ \Theta'$ by (intro exI[where $x=\Theta'' @ [a]$]) simp

from Cons.prem(1) have $\Gamma \Rightarrow a \# (a \# \Theta') @ \Theta 1 @ \Theta 2$ unfolding Θ using MittenTauschS by (metis append-assoc)

hence $\Gamma \Rightarrow (a \# \Theta') @ \Theta 1 @ \Theta 2$ using ZusammenziehungS by auto

hence $\Gamma \Rightarrow \Theta' @ \Theta$ unfolding Θ using AnfangTauschS by (metis append-Cons append-assoc)

from Cons.IH[OF this **] show $\Gamma \Rightarrow \Theta$.

qed simp

thus $\Gamma \Rightarrow \Theta @ \Theta \Longrightarrow \Gamma \Rightarrow \Theta$ by simp

qed

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

assume $mset \ \Gamma \Rightarrow mset \ \Delta$

then obtain n where $mset \ \Gamma \Rightarrow mset \ \Delta \downarrow n$ unfolding SC-SCp-eq[symmetric]

..

thus $\Gamma \Rightarrow \Delta$

proof(induction n arbitrary: $\Gamma \ \Delta$ rule: nat.induct)

```

    case (Suc n)
    have sr:  $\exists \Gamma 1 \ \Gamma 2. \ \Gamma = \Gamma 1 \ @ \ F \ # \ \Gamma 2 \ \wedge \ \Gamma' = \text{mset} \ (\Gamma 1 @ \Gamma 2)$  (is ?s) if mset
 $\Gamma = F, \ \Gamma'$  for  $\Gamma \ \Gamma' \ F$  proof –
      from that obtain  $\Gamma 1 \ \Gamma 2$  where  $\Gamma: \Gamma = \Gamma 1 \ @ \ F \ # \ \Gamma 2$  by (metis split-list
add.commute ex-mset list.set-intros(1) mset.simps(2) set-mset-mset)
      hence  $\Gamma': \Gamma' = \text{mset} \ (\Gamma 1 @ \Gamma 2)$  using that by auto
      show ?s using  $\Gamma \ \Gamma'$  by blast
    qed
    from Suc.premis show ?case proof (cases rule: SCc.cases)
      case BotL thus ?thesis using BotLe by simp
    next
      case Ax thus ?thesis using Axe by simp
    next
      case (NotL  $\Gamma' \ F$ )
      from (mset  $\Gamma = \neg \ F, \ \Gamma'$ ) obtain  $\Gamma 1 \ \Gamma 2$  where  $\Gamma: \Gamma = \Gamma 1 \ @ \ \neg \ F \ # \ \Gamma 2$ 
      by (metis split-list add.commute ex-mset list.set-intros(1) mset.simps(2)
set-mset-mset)
      hence  $\Gamma': \Gamma' = \text{mset} \ (\Gamma 1 @ \Gamma 2)$  using NotL(1) by simp
      from ( $\Gamma' \Rightarrow F, \ \text{mset} \ \Delta \downarrow n$ ) have  $\text{mset} \ (\Gamma 1 @ \Gamma 2) \Rightarrow \text{mset} \ (F \ # \ \Delta) \downarrow n$ 
    unfolding  $\Gamma'$  by (simp add: add.commute)
      from Suc.IH[OF this] show ?thesis unfolding  $\Gamma$  using AnfangTauschA
NEA by blast
    next
      case (NotR  $F \ \Delta'$ )
      from sr[OF NotR(1)] guess  $\Delta 1$  .. then guess  $\Delta 2$  .. note  $\Delta = \text{this}$ 
      with NotR have  $\text{mset} \ (F \ # \ \Gamma) \Rightarrow \text{mset} \ (\Delta 1 @ \Delta 2) \downarrow n$  by (simp add:
add.commute)
      from Suc.IH[OF this] show ?thesis using  $\Delta$  using AnfangTauschS NES by
blast
    next
      case (AndR  $F \ \Delta' \ G$ )
      from sr[OF AndR(1)] guess  $\Delta 1$  .. then guess  $\Delta 2$  .. note  $\Delta = \text{this}$ 
      with AndR have  $\text{mset} \ \Gamma \Rightarrow \text{mset} \ (F \ # \ \Delta 1 @ \Delta 2) \downarrow n$   $\text{mset} \ \Gamma \Rightarrow \text{mset} \ (G \ # \
\Delta 1 @ \Delta 2) \downarrow n$  by (simp add: add.commute)+
      from this[THEN Suc.IH] show ?thesis using  $\Delta$  using AnfangTauschS UES
by blast
    next
      case (OrR  $F \ G \ \Delta'$ )
      from sr[OF OrR(1)] guess  $\Delta 1$  .. then guess  $\Delta 2$  .. note  $\Delta = \text{this}$ 
      with OrR have  $\text{mset} \ \Gamma \Rightarrow \text{mset} \ (G \ # \ F \ # \ \Delta 1 @ \Delta 2) \downarrow n$  by (simp add:
add.commute add.left-commute add-mset-commute)
      from this[THEN Suc.IH] have  $\Gamma \Rightarrow G \ # \ F \ # \ \Delta 1 \ @ \ \Delta 2$  .
      with OES2 have  $\Gamma \Rightarrow F \vee G \ # \ F \ # \ \Delta 1 \ @ \ \Delta 2$  .
      with VertauschungS[where  $\Theta = \text{Nil}$ , simplified] have  $\Gamma \Rightarrow F \ # \ F \vee G \ # \ \Delta 1
@ \ \Delta 2$  .
      with OES1 have  $\Gamma \Rightarrow F \vee G \ # \ F \vee G \ # \ \Delta 1 \ @ \ \Delta 2$  .
      hence  $\Gamma \Rightarrow F \vee G \ # \ \Delta 1 \ @ \ \Delta 2$  using ZusammenziehungS by fast
      thus ?thesis unfolding  $\Delta$ [THEN conjunct1] using AnfangTauschS by blast
    next

```



```

    case (ImpR F G Δ')
    from sr[OF ImpR(1)] guess Δ1 .. then guess Δ2 .. note Δ = this
    with ImpR have mset (F#Γ) ⇒ mset (G # Δ1@Δ2) ↓ n by (simp add:
add commute)
    from this[THEN Suc.IH] show ?thesis using Δ using AnfangTauschS FES
by blast
  next
    case (AndL F G Γ')
    from sr[OF this(1)] guess Γ1 .. then guess Γ2 .. note Γ = this
    with AndL have mset (G # F # Γ1@Γ2) ⇒ mset Δ ↓ n by (simp add:
add commute add left-commute add-mset-commute)
    from this[THEN Suc.IH] have G # F # Γ1 @ Γ2 ⇒ Δ .
    with UEA2 have F ∧ G # F # Γ1 @ Γ2 ⇒ Δ .
    with VertauschungA[where Δ=Nil, simplified] have F # F ∧ G # Γ1 @
Γ2 ⇒ Δ .
    with UEA1 have F ∧ G # F ∧ G # Γ1 @ Γ2 ⇒ Δ .
    hence F ∧ G # Γ1 @ Γ2 ⇒ Δ using ZusammenziehungA by fast
    thus ?thesis unfolding Γ[THEN conjunct1] using AnfangTauschA by blast
  next
    case (OrL F Δ' G)
    from sr[OF this(1)] guess Γ1 .. then guess Γ2 .. note Γ = this
    with OrL have mset (F # Γ1@Γ2) ⇒ mset Δ ↓ n mset (G # Γ1@Γ2) ⇒
mset Δ ↓ n by (simp add: add commute)+
    from this[THEN Suc.IH] show ?thesis using Γ using AnfangTauschA OEA
by blast
  next
    case (ImpL Γ' F G)
    from sr[OF this(1)] guess Γ1 .. then guess Γ2 .. note Γ = this
    with ImpL have mset (Γ1@Γ2) ⇒ mset (F#Δ) ↓ n mset (G # Γ1@Γ2) ⇒
mset Δ ↓ n by (simp add: add commute)+
    from this[THEN Suc.IH] have Γ1 @ Γ2 ⇒ F # Δ G # Γ1 @ Γ2 ⇒ Δ .
    from FEA[OF this] have F → G # (Γ1 @ Γ2) @ (Γ1 @ Γ2) ⇒ Δ @ Δ .
    hence F → G # (Γ1 @ Γ2) @ (F → G # Γ1 @ Γ2) ⇒ Δ @ Δ using
AnfangTauschA VerduennungA by blast
    hence F → G # (Γ1 @ Γ2) ⇒ Δ @ Δ using ZusammenziehungListeA[where
Γ=F → G # (Γ1 @ Γ2)] by simp
    thus ?thesis unfolding Γ[THEN conjunct1] by(intro AnfangTauschA; elim
ZusammenziehungListeS)
  qed
qed blast
next
  have mset-Cons[simp]: mset (A # S) = A, mset S for A::'a formula and S by
(simp add: add commute)
  note mset.simps(2)[simp del]
  show Γ ⇒ Δ ⇒ mset Γ ⇒ mset Δ proof(induction rule: SCg.induct)

    case (Anfang D) thus ?case using extended-Ax SC-SCp-eq by force
  next
    case (FalschA) thus ?case using SCp.BotL by force

```

```

next
  case (VerduennungA  $\Gamma \Theta \mathfrak{D}$ ) thus ?case by (simp add: SC.weakenL)
next
  case (VerduennungS  $\Gamma \Theta \mathfrak{D}$ ) thus ?case by (simp add: SC.weakenR)
next
  case (ZusammenziehungA  $\mathfrak{D} \Gamma \Theta$ ) thus ?case using contractL by force
next
  case (ZusammenziehungS  $\Gamma \mathfrak{D} \Theta$ ) thus ?case using contract by force
next
  case (VertauschungA  $\Delta \mathfrak{D} \mathfrak{E} \Gamma \Theta$ ) thus ?case by fastforce
next
  case (VertauschungS  $\Gamma \Theta \mathfrak{E} \mathfrak{D} \Lambda$ ) thus ?case by fastforce
next
  case (Schnitt  $\Gamma \mathfrak{D} \Theta \Delta \Lambda$ )
  hence  $mset \Gamma \Rightarrow \mathfrak{D}, mset \Theta \mathfrak{D}, mset \Delta \Rightarrow mset \Lambda$  using SC-SCp-eq by auto
  from cut-cf[OF this] show ?case unfolding SC-SCp-eq by simp
next
  case (UES  $\Gamma \mathfrak{A} \Theta \mathfrak{B}$ ) thus ?case using SCp.AndR by (simp add: SC-SCp-eq)
next
  case (UEA1  $\mathfrak{A} \Gamma \Theta \mathfrak{B}$ )
  from  $\langle mset (\mathfrak{A} \# \Gamma) \Rightarrow mset \Theta \rangle$  have  $\mathfrak{A}, \mathfrak{B}, mset \Gamma \Rightarrow mset \Theta$  using SC.weakenL
  by auto
  thus ?case using SCp.AndL by force
next
  case (UEA2  $\mathfrak{B} \Gamma \Theta \mathfrak{A}$ )
  from  $\langle mset (\mathfrak{B} \# \Gamma) \Rightarrow mset \Theta \rangle$  have  $\mathfrak{A}, \mathfrak{B}, mset \Gamma \Rightarrow mset \Theta$  using SC.weakenL
  by auto
  thus ?case using SCp.AndL by force
next
  case (OEA  $\mathfrak{A} \Gamma \Theta \mathfrak{B}$ ) thus ?case unfolding SC-SCp-eq by (simp add:
  SCp.OrL)
next
  case (OES1  $\Gamma \mathfrak{A} \Theta \mathfrak{B}$ ) thus ?case using SC.weakenR[where 'a='a] by(auto
  intro!: SCp.intros( $\beta$ -))
next
  case (OES2  $\Gamma \mathfrak{B} \Theta \mathfrak{A}$ ) thus ?case by (simp add: SC.weakenR SCp.OrR)
next
  case (FES  $\mathfrak{A} \Gamma \mathfrak{B} \Theta$ ) thus ?case using weakenR unfolding SC-SCp-eq by
  (simp add: SCp.ImpR)
next
  case (FEA  $\Gamma \mathfrak{A} \Theta \mathfrak{B} \Delta \Lambda$ )
  from  $\langle mset \Gamma \Rightarrow mset (\mathfrak{A} \# \Theta) \rangle$  [THEN weakenL-set, THEN weakenR-set, of
   $mset \Delta mset \Lambda$ ]
  have  $S: mset (\Gamma @ \Delta) \Rightarrow \mathfrak{A}, mset (\Theta @ \Lambda)$  unfolding mset-append mset-Cons by
  (simp add: add-ac)
  from FEA obtain  $m$  where  $mset (\mathfrak{B} \# \Delta) \Rightarrow mset \Lambda$  by blast
  hence  $mset \Gamma + mset (\mathfrak{B} \# \Delta) \Rightarrow mset \Theta + mset \Lambda$  using weakenL-set
  weakenR-set by fast
  hence  $A: \mathfrak{B}, mset (\Gamma @ \Delta) \Rightarrow mset (\Theta @ \Lambda)$  by (simp add: add.left-commute)

```

```

  show ?case using S A SC-SCp-eq SCp.ImpL unfolding mset-Cons by blast
next
  case (NES  $\mathfrak{A}$   $\Gamma$   $\Theta$ ) thus ?case using SCp.NotR by (simp add: SC-SCp-eq)
next
  case (NEA  $\Gamma$   $\mathfrak{A}$   $\Theta$ ) thus ?case using SCp.NotL by (simp add: SC-SCp-eq)
qed
qed

```

end

2.1.4 Soundness, Completeness

```

theory SC-Sema
imports SC Sema
begin

```

definition *sequent-antics* :: '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*: $\models \Gamma \Rightarrow \{\#\delta\# \} \longleftrightarrow \text{set-mset } \Gamma \models \delta$
unfolding *sequent-semantics-def entailment-def* **by** *simp*

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

by(*induction rule: SCp.induct*) (*auto simp add: sequent-semantics-def*)

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

function(*sequential*)

sc :: 'a formula list \Rightarrow 'a list \Rightarrow 'a formula list \Rightarrow 'a list \Rightarrow ('a list \times 'a list) set

where

sc (\perp # Γ) $A \Delta B = \{\}$ |

sc [] A [] $B = (\text{if set } A \cap \text{set } B = \{\} \text{ then } \{\text{(remdups } A, \text{remdups } B)\} \text{ else } \{\})$ |

sc (*Atom* k # Γ) $A \Delta B = \text{sc } \Gamma (k\#A) \Delta B$ |

sc (*Not* F # Γ) $A \Delta B = \text{sc } \Gamma A (F\#\Delta) B$ |

sc (*And* $F G$ # Γ) $A \Delta B = \text{sc } (F\#G\#\Gamma) A \Delta B$ |

sc (*Or* $F G$ # Γ) $A \Delta B = \text{sc } (F\#\Gamma) A \Delta B \cup \text{sc } (G\#\Gamma) A \Delta B$ |

sc (*Imp* $F G$ # Γ) $A \Delta B = \text{sc } \Gamma A (F\#\Delta) B \cup \text{sc } (G\#\Gamma) A \Delta B$ |

sc $\Gamma A (\perp\#\Delta) B = \text{sc } \Gamma A \Delta B$ |

sc $\Gamma A (\text{Atom } k \# \Delta) B = \text{sc } \Gamma A \Delta (k\#B)$ |

```

sc  $\Gamma$  A (Not F #  $\Delta$ ) B = sc (F# $\Gamma$ ) A  $\Delta$  B |
sc  $\Gamma$  A (And F G #  $\Delta$ ) B = sc  $\Gamma$  A (F# $\Delta$ ) B  $\cup$  sc  $\Gamma$  A (G# $\Delta$ ) B |
sc  $\Gamma$  A (Or F G #  $\Delta$ ) B = sc  $\Gamma$  A (F#G# $\Delta$ ) B |
sc  $\Gamma$  A (Imp F G #  $\Delta$ ) B = sc (F# $\Gamma$ ) A (G# $\Delta$ ) B
  by pat-completeness auto

```

definition *list-sequent-cost* $\Gamma \Delta = 2 * \text{sum-list} (\text{map size } (\Gamma @ \Delta)) + \text{length } (\Gamma @ \Delta)$
termination *sc* **by** (relation measure $(\lambda(\Gamma, A, \Delta, B). \text{list-sequent-cost } \Gamma \Delta)$) (*simp-all*
add: list-sequent-cost-def)

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

by *code-simp*

lemma *sc-sim*:

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

assumes *sc* Γ A Δ B = {}

shows *image-mset Atom* (mset A) + mset $\Gamma \Rightarrow$ *image-mset Atom* (mset B) + mset Δ

proof –

have *[*simp*]: *image-mset Atom* (mset A) \Rightarrow *image-mset Atom* (mset B) (**is** ?k)

if $k \in \text{set A}$ $k \in \text{set B}$ **for** A B :: 'a list **and** k

proof –

from *that* **obtain** a **where** $a \in \text{set A}$ $a \in \text{set B}$ **by** *blast*

thus ?k **by** (*force simp: in-image-mset intro: SCp.Ax*[**where** $k=a$])

qed

from *assms* **show** ?thesis

by (*induction rule: sc.induct*[**where** 'a='a]) (*auto*

simp add: list-sequent-cost-def add.assoc Bot-delR-simp

split: if-splits option.splits

intro: SCp.intros(3-))

qed

lemma *scc-ce-distinct*:

(C, E) \in *sc* Γ G Δ D \Longrightarrow *set C* \cap *set E* = {}

by (*induction* Γ G Δ D *arbitrary: C E rule: sc.induct*)

(*fastforce split: if-splits*)+

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

theorem *SC-counterexample*:

(C, D) \in *sc* Γ A Δ B \Longrightarrow

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

by (*induction rule: sc.induct*[**where** 'a='a];

simp add: sequent-semantics-def split: if-splits;

blast)

corollary *SC-counterexample'*:

assumes $(C,D) \in sc \Gamma \square \Delta \square$
shows $(\lambda k. k \in set C) \neg \models mset \Gamma \Rightarrow mset \Delta$
using *SC-counterexample[OF assms]* **by** *simp*

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

proof

assume $\Gamma \Rightarrow \Delta$ **thus** $\models \Gamma \Rightarrow \Delta$ **using** *SC-soundness* **by** *blast*

next

obtain $\Gamma' \Delta'$ **where** [*simp*]: $\Gamma = mset \Gamma' \Delta = mset \Delta'$ **by** (*metis ex-mset*)

assume $\models \Gamma \Rightarrow \Delta$

hence $sc \Gamma' \square \Delta' \square = \{\}$

proof(*rule contrapos-pp*)

assume $sc \Gamma' \square \Delta' \square \neq \{\}$

then obtain $C E$ **where** $(C,E) \in sc \Gamma' \square \Delta' \square$ **by** *fast*

thus $\neg \models \Gamma \Rightarrow \Delta$ **using** *SC-counterexample'* **by** *fastforce*

qed

from *sc-sim[OF this]* **show** $\Gamma \Rightarrow \Delta$ **by** *auto*

qed

theorem $\models \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta$

proof –

assume $s: \models \Gamma \Rightarrow \Delta$

obtain $\Gamma' \Delta'$ **where** $p: \Gamma = mset \Gamma' \Delta = mset \Delta'$ **by** (*metis ex-mset*)

have $mset \Gamma' \Rightarrow mset \Delta'$

proof *cases* — just to show that we didn't need to show the lemma above by contraposition. It's just quicker to do so.

assume $sc \Gamma' \square \Delta' \square = \{\}$

from *sc-sim[OF this]* **show** $mset \Gamma' \Rightarrow mset \Delta'$ **by** *auto*

next

assume $sc \Gamma' \square \Delta' \square \neq \{\}$

with *SC-counterexample* **have** $\neg \models mset \Gamma' \Rightarrow mset \Delta'$ **by** *fastforce*

moreover note $s[\text{unfolded } p]$

ultimately have *False* ..

thus $mset \Gamma' \Rightarrow mset \Delta'$..

qed

thus *?thesis unfolding p* .

qed

end

theory *SC-Depth-Limit*

imports *SC-Sema SC-Depth*

begin

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

proof(*induction sequent-cost* $\Gamma \Delta$ *arbitrary:* $\Gamma \Delta$)

```

  case 0 hence False by (simp add: sequent-cost-def) thus ?case by clarify
next
  case (Suc n)
  from Suc(3) show ?case
    using SCc.cases[OF Suc.hyps(1)]
oops

```

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 -
  have [simp]: image-mset Atom (mset A)  $\Rightarrow$  image-mset Atom (mset B)  $\downarrow$  Suc 0
  (is ?k) if set A  $\cap$  set B  $\neq \{\}$  for A B
  proof -
    from that obtain a where a  $\in$  set A a  $\in$  set B by blast
    thus ?k by (force simp: in-image-mset intro: SCc.Ax[where k=a])
  qed
  note SCc.intros(3-)[intro]
  have [elim!]:  $\Gamma \Rightarrow \Delta \downarrow n \Longrightarrow n \leq m \Longrightarrow \Gamma \Rightarrow \Delta \downarrow m$  for  $\Gamma \Delta n m$  using
  dec-induct by (fastforce elim!: deeper-suc)
  from assms show ?thesis
  by (induction  $\Gamma A \Delta B$  rule: sc.induct)
  (auto
  simp add: list-sequent-cost-def add.assoc deeper-suc weakenR'
  split: if-splits option.splits)
qed

```

corollary *sc-depth-complete*:

```

  assumes s:  $\models \Gamma \Rightarrow \Delta$ 
  shows  $\Gamma \Rightarrow \Delta \downarrow$  sum-mset (image-mset size ( $\Gamma + \Delta$ ))
proof -
  obtain  $\Gamma' \Delta'$  where  $p: \Gamma =$  mset  $\Gamma' \Delta =$  mset  $\Delta'$  by (metis ex-mset)
  with s have sl:  $\models$  mset  $\Gamma' \Rightarrow$  mset  $\Delta'$  by simp
  let ?d = sum-mset (image-mset size ( $\Gamma + \Delta$ ))
  have d: ?d = sum-list (map size ( $\Gamma' @ \Delta'$ ))
  unfolding p by (metis mset-append mset-map sum-mset-sum-list)
  have mset  $\Gamma' \Rightarrow$  mset  $\Delta' \downarrow$  ?d
  proof cases
    assume sc  $\Gamma' [] \Delta' [] = \{\}$ 
    from sc-sim-depth[OF this] show mset  $\Gamma' \Rightarrow$  mset  $\Delta' \downarrow$  ?d unfolding d by
  auto
  next
    assume sc  $\Gamma' [] \Delta' [] \neq \{\}$ 
    with SC-counterexample have  $\neg \models$  mset  $\Gamma' \Rightarrow$  mset  $\Delta'$  by fastforce
    moreover note s[unfolded p]

```

```

    ultimately have False ..
    thus mset  $\Gamma' \Rightarrow \text{mset } \Delta' \downarrow ?d$  ..
  qed
  thus ?thesis unfolding p .
qed

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

context begin
private lemma reasonable:
   $\forall \Gamma'. F \triangleright \text{set-mset } \Gamma = \text{set-mset } \Gamma' \longrightarrow P \Gamma' \Longrightarrow P (F, \Gamma)$ 
   $\forall \Gamma'. F \triangleright G \triangleright \text{set-mset } \Gamma = \text{set-mset } \Gamma' \longrightarrow P \Gamma' \Longrightarrow P (F, G, \Gamma)$  by simp-all

lemma SC-consistent: pcp {set-mset  $\Gamma \mid \Gamma. \neg(\Gamma \Rightarrow \{\#\})$ }
  unfolding pcp-def
  apply(intro ballI conjI; erule contrapos-pp; clarsimp; ((drule reasonable)+)?)
  apply(auto dest!: NotL-inv AndL-inv OrL-inv ImpL-inv NotR-inv AndR-inv
OrR-inv ImpR-inv multi-member-split contractL contractR intro!: SCp.intros(3-)
intro: contractR contractL)
  apply (metis add-mset-commute contract)

done

end

lemma
  fixes  $\Gamma \Delta :: 'a :: \text{countable formula multiset}$ 
  shows  $\models \Gamma \Rightarrow \Delta \Longrightarrow \Gamma \Rightarrow \Delta$ 
proof(erule contrapos-pp)
  have NotInv:  $\Gamma + \text{image-mset Not } \Delta \Rightarrow \{\#\} \Longrightarrow \Gamma \Rightarrow \Delta$ 
  by (induction  $\Delta$  arbitrary:  $\Gamma$ ; simp add: NotL-inv)
  assume  $\langle \neg \Gamma \Rightarrow \Delta \rangle$ 
  hence  $\langle \neg \Gamma + \text{image-mset Not } \Delta \Rightarrow \{\#\} \rangle$  using NotInv by blast
  with pcp-sat[OF SC-consistent]
  have sat (set-mset ( $\Gamma + \text{image-mset } \neg \Delta$ )) by blast
  thus  $\langle \neg (\models \Gamma \Rightarrow \Delta) \rangle$  unfolding sat-def sequent-semantics-def not-all by (force
elim!: ex-forward)
qed

end

```

2.2 Natural Deduction

```

theory ND
imports Formulas
begin

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

```

```

lemma Weaken:  $[\Gamma \vdash F; \Gamma \subseteq \Gamma'] \Rightarrow \Gamma' \vdash F$ 
proof(induct arbitrary:  $\Gamma'$  rule: ND.induct)
  case (NotI F  $\Gamma$ ) thus ?case using ND.NotI by auto
next
  case Ax thus ?case by(blast intro: ND.Ax)
next
  case NotE thus ?case by(blast intro: ND.NotE)
next
  case CC thus ?case using ND.CC by blast
next
  case AndE1 thus ?case using ND.AndE1 by metis
next
  case AndE2 thus ?case using ND.AndE2 by metis
next
  case AndI thus ?case by (simp add: ND.AndI)
next
  case OrI1 thus ?case using ND.OrI1 by blast
next
  case OrI2 thus ?case using ND.OrI2 by blast
next
  case (OrE  $\Gamma F G H$ ) show ?case apply(insert OrE.prem)
    apply(rule ND.OrE[of  $\Gamma' F G$ ])
    apply(rule OrE.hyps(2)[OF OrE.prem])
    apply(rule OrE.hyps(4); blast)
    apply(rule OrE.hyps(6); blast)
  done
next
  case ImpI thus ?case using ND.ImpI by blast

```


next
 case *ImpE* **thus** *?case* **using** *ND.ImpE* **by** *metis*
qed

lemma *BotE* : $\Gamma \vdash \perp \Longrightarrow \Gamma \vdash F$
by (*meson CC subset-insertI Weaken*)

lemma *Not2E*: $\text{Not}(\text{Not } F) \triangleright \Gamma \vdash F$
by (*metis CC ND.Ax NotE insertI1 insert-commute*)

lemma *Not2I*: $F \triangleright \Gamma \vdash \text{Not}(\text{Not } F)$
by (*metis CC ND.Ax NotE insertI1 insert-commute*)

lemma *Not2IE*: $F \triangleright \Gamma \vdash G \Longrightarrow \text{Not}(\text{Not } F) \triangleright \Gamma \vdash G$
by (*meson ImpE ImpI Not2E Weaken subset-insertI*)

lemma *NDtrans*: $\Gamma \vdash F \Longrightarrow F \triangleright \Gamma \vdash G \Longrightarrow \Gamma \vdash G$
using *ImpE ImpI* **by** *blast*

lemma *AndL-sim*: $F \triangleright G \triangleright \Gamma \vdash H \Longrightarrow \text{And } F G \triangleright \Gamma \vdash H$
apply (*drule Weaken*[**where** $\Gamma' = \text{And } F G \triangleright F \triangleright G \triangleright \Gamma$])
apply *blast*
by (*metis AndE1 AndE2 ND.Ax NDtrans insertI1 insert-commute*)

lemma *NotSwap*: $\text{Not } F \triangleright \Gamma \vdash G \Longrightarrow \text{Not } G \triangleright \Gamma \vdash F$
using *CC NotE insert-commute subset-insertI Weaken* **by** (*metis Ax insertI1*)

lemma *AndR-sim*: $\llbracket \text{Not } F \triangleright \Gamma \vdash H; \text{Not } G \triangleright \Gamma \vdash H \rrbracket \Longrightarrow \text{Not}(\text{And } F G) \triangleright \Gamma \vdash H$
using *AndI NotSwap* **by** *blast*

lemma *OrL-sim*: $\llbracket F \triangleright \Gamma \vdash H; G \triangleright \Gamma \vdash H \rrbracket \Longrightarrow F \vee G \triangleright \Gamma \vdash H$
using *Weaken*[**where** $\Gamma' = F \triangleright \text{Or } F G \triangleright \Gamma$] *Weaken*[**where** $\Gamma' = G \triangleright \text{Or } F G \triangleright \Gamma$]
by (*meson ND.Ax OrE insertI1 insert-mono subset-insertI*)

lemma *OrR-sim*: $\llbracket \neg F \triangleright \neg G \triangleright \Gamma \vdash \perp \rrbracket \Longrightarrow \neg(G \vee F) \triangleright \Gamma \vdash \perp$
proof –
assume $\neg F \triangleright \neg G \triangleright \Gamma \vdash \perp$
then have $\bigwedge f. f \triangleright \neg F \triangleright \neg G \triangleright \Gamma \vdash \perp$ **by** (*meson Weaken subset-insertI*)
then have $\bigwedge f. \neg G \triangleright \neg(f \vee F) \triangleright \Gamma \vdash \perp$ **by** (*metis NDtrans Not2E NotSwap OrI2 insert-commute*)
then show *?thesis* **by** (*meson NDtrans Not2I NotSwap OrI1*)
qed

lemma *ImpL-sim*: $\llbracket \neg F \triangleright \Gamma \vdash \perp; G \triangleright \Gamma \vdash \perp \rrbracket \Longrightarrow F \rightarrow G \triangleright \Gamma \vdash \perp$
by (*meson CC ImpE ImpI ND.Ax Weaken insertI1 subset-insertI*)

lemma *ImpR-sim*: $\llbracket \neg G \triangleright F \triangleright \Gamma \vdash \perp \rrbracket \Longrightarrow \neg(F \rightarrow G) \triangleright \Gamma \vdash \perp$
by (*metis (full-types) ImpI NotSwap insert-commute*)

```

lemma ND-lem: {} ⊢ Not F ∨ F
  apply(rule CC)
  apply(rule OrE[of - Not F F])
    apply(rule OrI1)
    apply(rule NotI)
  apply(rule NotE[of - (¬ F ∨ F)]); blast intro: OrI1 OrI2 Ax)+
done

lemma ND-caseDistinction: [ F ▷ Γ ⊢ H; Not F ▷ Γ ⊢ H ] ⇒ Γ ⊢ H
  by (meson ND-lem OrE empty-subsetI Weaken)

lemma [¬ F ▷ Γ ⊢ H; G ▷ Γ ⊢ H] ⇒ F → G ▷ Γ ⊢ H
  apply(rule ND-caseDistinction[of F])
  apply (meson ImpE ImpI ND.intros(1) Weaken insertI1 subset-insertI)
  apply (metis Weaken insert-commute subset-insertI)
done

lemma ND-deMorganAnd: {¬ (F ∧ G)} ⊢ ¬ F ∨ ¬ G
  apply(rule CC)
  apply(rule NotE[of - F ∧ G])
    apply(simp add: Ax; fail)
  apply(rule AndI)
  apply(rule CC)
  apply(rule NotE[of - ¬ F ∨ ¬ G])
    apply(simp add: Ax; fail)
  apply(rule OrI1)
  apply(simp add: Ax; fail)
  apply(rule CC)
  apply(rule NotE[of - ¬ F ∨ ¬ G])
    apply(simp add: Ax; fail)
  apply(rule OrI2)
  apply(simp add: Ax; fail)
done

lemma ND-deMorganOr: {¬ (F ∨ G)} ⊢ ¬ F ∧ ¬ G
  apply(rule ND-caseDistinction[of F]; rule ND-caseDistinction[of G])
  apply(rule CC; rule NotE[of - F ∨ G]; simp add: Ax OrI2 OrI1; fail)+
  apply(rule AndI; simp add: Ax; fail)
done

lemma sim-sim: F ▷ Γ ⊢ H ⇒ G ▷ Γ ⊢ F ⇒ G ▷ Γ ⊢ H
  by (meson ImpE ImpI Weaken subset-insertI)
thm sim-sim[where Γ={}, rotated, no-vars]

lemma Top-provable[simp,intro!]: Γ ⊢ ⊤ unfolding Top-def by (intro ND.ImpI
ND.Ax) simp

lemma NotBot-provable[simp,intro!]: Γ ⊢ ¬ ⊥ using NotI BotE Ax by blast

```

```

lemma Top-useless:  $\Gamma \vdash F \implies \Gamma - \{\top\} \vdash F$ 
  by (metis NDtrans Top-provable Weaken insert-Diff-single subset-insertI)

lemma AssmBigAnd:  $set\ G \vdash F \iff \{\} \vdash (\bigwedge G \rightarrow F)$ 
proof(induction G arbitrary: F)
  case Nil thus ?case by(fastforce intro: ND.ImpI elim: Weaken ImpE[OF - NotBot-provable])
  next
    case (Cons G GS) show ?case proof
      assume  $set\ (G \# GS) \vdash F$ 
      hence  $set\ GS \vdash G \rightarrow F$  by(intro ND.ImpI simp)
      with Cons.IH have *:  $\{\} \vdash \bigwedge GS \rightarrow G \rightarrow F$  ..
      hence  $\{G, \bigwedge GS\} \vdash F$  proof -
        have *:  $\{\bigwedge GS\} \vdash G \rightarrow F$ 
          using Weaken[OF * empty-subsetI] ImpE[where  $\Gamma = \{\bigwedge GS\}$  and  $F = \bigwedge GS$ ] by (simp add: ND.Ax)
          show  $\{G, \bigwedge GS\} \vdash F$  using Weaken[OF *] ImpE[where  $\Gamma = \{G, \bigwedge GS\}$  and  $F = G$ ] ND.Ax by (simp add: ND.Ax)
        qed
      thus  $\{\} \vdash \bigwedge (G \# GS) \rightarrow F$  by(intro ND.ImpI; simp add: AndL-sim)
    next
      assume  $\{\} \vdash \bigwedge (G \# GS) \rightarrow F$ 
      hence  $\{G \wedge \bigwedge GS\} \vdash F$  using ImpE[OF - Ax[OF singletonI]] Weaken by
fastforce
      hence  $\{G, \bigwedge GS\} \vdash F$  by (meson AndI ImpE ImpI ND.intros(1) Weaken insertI1 subset-insertI)
      hence  $\{\bigwedge GS\} \vdash G \rightarrow F$  using ImpI by blast
      hence  $\{\} \vdash \bigwedge GS \rightarrow G \rightarrow F$  using ImpI by blast
      with Cons.IH have  $set\ GS \vdash G \rightarrow F$  ..
      thus  $set\ (G \# GS) \vdash F$  using ImpE Weaken by (metis Ax list.set-intros(1) set-subset-Cons)
    qed
  qed

end
theory ND-Sound
imports ND Sema
begin

lemma BigAndImp:  $A \models (\bigwedge P \rightarrow G) \iff ((\forall F \in set\ P. A \models F) \longrightarrow A \models G)$ 
  by(induction P; simp add: entailment-def)

lemma ND-sound:  $\Gamma \vdash F \implies \Gamma \models F$ 
  by(induction rule: ND.induct; simp add: entailment-def; blast)

end
theory ND-Compl-Truthable
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!*]: $\mathcal{A} \models \text{turn-true } \mathcal{A} F$ **unfolding** *turn-true-def* **by** *simp*

lemma *turn-true-simps*[*simp*]:

$\mathcal{A} \models F \implies \text{turn-true } \mathcal{A} F = F$

$\neg \mathcal{A} \models F \implies \text{turn-true } \mathcal{A} F = \neg F$

unfolding *turn-true-def* **by** *simp-all*

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*]:

line-suitable $\mathcal{A} (\text{Not } F) = \text{line-suitable } \mathcal{A} F$

line-suitable $\mathcal{A} (\text{And } F G) = (\text{line-suitable } \mathcal{A} F \wedge \text{line-suitable } \mathcal{A} G)$

line-suitable $\mathcal{A} (\text{Or } F G) = (\text{line-suitable } \mathcal{A} F \wedge \text{line-suitable } \mathcal{A} G)$

line-suitable $\mathcal{A} (\text{Imp } F G) = (\text{line-suitable } \mathcal{A} F \wedge \text{line-suitable } \mathcal{A} G)$

unfolding *line-suitable-def* **by**(*clarsimp; linarith*)**+**

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

unfolding *line-assm-def* **by** *simp*

lemma *NotD*: $\Gamma \vdash \neg F \implies F \triangleright \Gamma \vdash \perp$ **by** (*meson Not2I NotE Weaken subset-insertI*)

lemma *truthline-ND-proof*:

fixes F :: 'a formula

assumes *line-suitable* $Z F$

shows *line-assm* $\mathcal{A} Z \vdash \text{turn-true } \mathcal{A} F$

using *assms* **proof**(*induction F*)

case (*Atom k*) **thus** ?*case* **using** Ax [**where** 'a='a] **by** (*simp add: line-suitable-def line-assm-def*)

next

case *Bot*

have *turn-true* $\mathcal{A} \perp = \text{Not } \text{Bot}$ **unfolding** *turn-true-def* **by** *simp*

thus ?*case* **by** (*simp add: Ax NotI*)

next

have [*simp*]: $\Gamma \vdash \neg (\neg F) \longleftrightarrow \Gamma \vdash F$ **for** F :: 'a formula **and** Γ **by** (*metis NDtrans Not2E Not2I*)

case (*Not F*)

hence *line-assm* $\mathcal{A} Z \vdash \text{turn-true } \mathcal{A} F$ **by** *simp*

thus ?*case* **by**(*cases* $\mathcal{A} \models F$; *simp*)

next

have [*simp*]: $\llbracket \text{line-assm } \mathcal{A} Z \vdash \neg F; \neg \mathcal{A} \models F \rrbracket \implies F \wedge G \triangleright \text{line-assm } \mathcal{A} Z \vdash$

```

⊥ for  $F G$  by(blast intro!: NotE[where  $F=F$ ] intro: AndE1[OF Ax] Weaken[OF
- subset-insertI])
  have [simp]: [ $\text{line-asm } \mathcal{A} Z \vdash \neg G; \neg \mathcal{A} \models G$ ]  $\implies F \wedge G \triangleright \text{line-asm } \mathcal{A} Z \vdash$ 
⊥ for  $F G$  by(blast intro!: NotE[where  $F=G$ ] intro: AndE2[OF Ax] Weaken[OF
- subset-insertI])

  case (And F G)
  thus ?case by(cases  $\mathcal{A} \models F$ ; cases  $\mathcal{A} \models G$ ; simp; intro ND.NotI AndI; simp)
next
  case (Or F G)
  thus ?case by(cases  $\mathcal{A} \models F$ ; cases  $\mathcal{A} \models G$ ; simp; (elim ND.OrI1 ND.OrI2)?)
(force intro!: NotI dest!: NotD dest: OrL-sim)
next
  case (Imp F G)
  hence mIH:  $\text{line-asm } \mathcal{A} Z \vdash \text{turn-true } \mathcal{A} F \text{ line-asm } \mathcal{A} Z \vdash \text{turn-true } \mathcal{A} G$ 
by simp+
  thus ?case by(cases  $\mathcal{A} \models F$ ; cases  $\mathcal{A} \models G$ ; simp; intro ImpI NotI ImpL-sim;
simp add: Weaken[OF - subset-insertI] NotSwap NotD NotD[THEN BotE])
qed
thm NotD[THEN BotE]

```

```

lemma deconstruct-assm-set:
  assumes IH:  $\bigwedge \mathcal{A}. \text{line-asm } \mathcal{A} (k \triangleright Z) \vdash F$ 
  shows  $\bigwedge \mathcal{A}. \text{line-asm } \mathcal{A} Z \vdash F$ 
proof cases
  assume  $k \in Z$  with IH show ?thesis  $\mathcal{A}$  for  $\mathcal{A}$  by (simp add: insert-absorb)
next
  assume  $k \notin Z$ 
  fix  $\mathcal{A}$ 

```

Since we require the IH for arbitrary \mathcal{A} , we use a modified \mathcal{A} from the conclusion like this:

```

from IH have av:  $\text{line-asm } (\mathcal{A}(k := v)) (k \triangleright Z) \vdash F$  for  $v$  by blast

```

However, that modification is only relevant for $k \triangleright Z$, nothing from Z gets touched.

```

from  $\langle k \notin Z \rangle$  have  $\text{line-asm } (\mathcal{A}(k := v)) Z = \text{line-asm } \mathcal{A} Z$  for  $v$  unfolding
line-asm-def turn-true-def by force

```

That means we can rewrite the modified *line-asm* like this:

```

hence  $\text{line-asm } (\mathcal{A}(k := v)) (k \triangleright Z) =$ 
  (if  $v$  then Atom k else Not (Atom k))  $\triangleright \text{line-asm } \mathcal{A} Z$  for  $v$  by simp

```

Inserting *True* and *False* for v yields the two alternatives.

```

with av have Atom k  $\triangleright \text{line-asm } \mathcal{A} Z \vdash F$  Not (Atom k)  $\triangleright \text{line-asm } \mathcal{A} Z \vdash F$ 
  by(metis (full-types))+
with ND-caseDistinction show  $\text{line-asm } \mathcal{A} Z \vdash F$  .
qed

```

```

theorem ND-complete:
  assumes taut:  $\models F$ 
  shows  $\{\} \vdash F$ 
proof -
  have [simp]: turn-true  $Z F = F$  for  $Z$  using taut by simp

  have line-assm  $\mathcal{A} \{\} \vdash F$  for  $\mathcal{A}$ 
proof(induction arbitrary: A rule: finite-empty-induct)
  show fat: finite (atoms  $F$ ) by (fact atoms-finite)
next
  have su: line-suitable (atoms  $F$ )  $F$  unfolding line-suitable-def by simp
  with truthline-ND-proof[OF su] show base: line-assm  $\mathcal{A}$  (atoms  $F$ )  $\vdash F$  for
 $\mathcal{A}$  by simp
next
  case ( $\exists k Z$ )
  from  $\langle k \in Z \rangle$  have *:  $\langle k \triangleright Z - \{k\} = Z \rangle$  by blast
  from  $\langle \bigwedge \mathcal{A}. \textit{line-assm } \mathcal{A} Z \vdash F \rangle$ 
  show  $\langle \textit{line-assm } \mathcal{A} (Z - \{k\}) \vdash F \rangle$ 
  using deconstruct-assm-set[of k Z - {k} F A]
  unfolding * by argo
qed

  thus ?thesis unfolding line-assm-def by simp
qed

corollary ND-sound-complete:  $\{\} \vdash F \longleftrightarrow \models F$ 
  using ND-sound[of { } F] ND-complete[of F] unfolding entailment-def by blast

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

theorem
  fixes  $\Gamma :: 'a :: \textit{countable formula set}$ 
  shows  $\Gamma \models F \implies \Gamma \vdash F$ 
proof -
  assume  $\langle \Gamma \models F \rangle$ 
  with compact-to-formula guess  $G$  .
  from ND-complete  $\langle \models \bigwedge G \rightarrow F \rangle$  have  $\langle \{\} \vdash \bigwedge G \rightarrow F \rangle$  .
  with AssmBigAnd have  $\langle \textit{set } G \vdash F \rangle$  ..
  with Weaken show ?thesis using  $\langle \textit{set } G \subseteq \Gamma \rangle$  .
qed

end

```

2.3 Hilbert Calculus

```
theory HC
imports Formulas
begin
```

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

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

```
context begin
```

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

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

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

```
inductive-set AXH where
 $(F \rightarrow (G \rightarrow F)) \in AXH$  |
 $(F \rightarrow (F \rightarrow G)) \rightarrow (F \rightarrow G) \in AXH$  |
 $(F \rightarrow (G \rightarrow H)) \rightarrow (G \rightarrow (F \rightarrow H)) \in AXH$  |
 $(G \rightarrow H) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H)) \in AXH$ 
```

```
lemma HC-mono:  $S \vdash_H F \Longrightarrow S \subseteq T \Longrightarrow T \vdash_H F$ 
by(induction rule: HC.induct) (auto intro: HC.intros)
```

```

lemma AX010: AX0  $\subseteq$  AX10
  apply(rule)
  apply(cases rule: AX0.cases, assumption)
  apply(auto intro: AX10.intros)
done
lemma AX100[simp]: AX0  $\cup$  AX10 = AX10 using AX010 by blast

```

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

```

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

```

proof –

have 0:

AX0 $\vdash_H (F \rightarrow (F \rightarrow G)) \rightarrow (F \rightarrow G)$

AX0 $\vdash_H (F \rightarrow (G \rightarrow H)) \rightarrow (G \rightarrow (F \rightarrow H))$

AX0 $\vdash_H (G \rightarrow H) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H))$

for F G H **using** HC-intros(1,2) MP **by** metis+

have H:

AXH $\vdash_H (F \rightarrow (G \rightarrow H)) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H))$

for F G H

proof –

note AXH.intros[THEN HC.Ax]

thus ?thesis **using** MP **by** metis

qed

note * = H 0

note * = *[THEN HC-mono, OF Un-upper1]

show ?thesis (is ?Z \longleftrightarrow ?H)

proof

assume ?Z **thus** ?H **proof** induction

case MP **thus** ?case **using** HC.MP **by** blast

next

case (Ax F) **thus** ?case **proof**

assume F \in AX0 **thus** ?thesis **by** induction (simp-all add: AXH.intros(1)

HC.Ax *)

next

assume F \in Γ **thus** ?case **using** HC.Ax[of F] **by** simp

qed

qed

next

assume ?H **thus** ?Z **proof** induction

case MP **thus** ?case **using** HC.MP **by** blast

next

case (Ax F) **thus** ?case **proof**

assume F \in AXH **thus** ?thesis **by** induction (simp-all add: AX0.intros(1)

HC.Ax *)

next

assume F \in Γ **thus** ?case **using** HC.Ax[of F] **by** simp

qed

qed

qed
qed

lemma $AX0 \vdash_H F \rightarrow F$ **by** (*meson HC-intros(1,2) HC.MP*)

lemma *imp-self*: $AX0 \vdash_H F \rightarrow F$ **proof** –

let $?d = \lambda f. AX0 \vdash_H f$

note *MP*

moreover have $?d (F \rightarrow (G \rightarrow F))$ for G using *HC-intros(1)*[where $G=G$ and $F=F$].

moreover {

note *MP*

moreover have $?d (F \rightarrow ((G \rightarrow F) \rightarrow F))$ for G using *HC-intros(1)*[where $G=G \rightarrow F$ and $F=F$].

moreover have $?d ((F \rightarrow ((G \rightarrow F) \rightarrow F)) \rightarrow ((F \rightarrow (G \rightarrow F)) \rightarrow (F \rightarrow F)))$ for G using *HC-intros(2)*[where $G=G \rightarrow F$ and $F=F$ and $H=F$].

ultimately have $?d ((F \rightarrow (G \rightarrow F)) \rightarrow (F \rightarrow F))$ for G . }

ultimately show $?d (F \rightarrow F)$.

qed

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

proof(*induction rule: HC.induct*)

case (*Ax G*)

show $?case$ **proof cases**

assume $F = G$

from *imp-self* have $AX0 \vdash_H G \rightarrow G$.

with *HC-mono* show $?case$ **unfolding** $\langle F = G \rangle$ using *sup-ge1*.

next

assume $F \neq G$

note *HC.MP*

moreover {

from $\langle F \neq G \rangle \langle G \in AX0 \cup \text{insert } F \Gamma \rangle$ have $G \in AX0 \cup \Gamma$ by *simp*

with *HC.Ax* have $AX0 \cup \Gamma \vdash_H G$.

}

moreover from *HC-mono*[*OF HC-intros(1) sup-ge1*] have $AX0 \cup \Gamma \vdash_H G \rightarrow (F \rightarrow G)$.

ultimately show $?case$.

qed

next

case (*MP G H*)

have $AX0 \cup \Gamma \vdash_H (F \rightarrow (G \rightarrow H)) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H))$ using *HC-mono* by *blast*

with *HC.MP* $\langle AX0 \cup \Gamma \vdash_H F \rightarrow (G \rightarrow H) \rangle$ have $AX0 \cup \Gamma \vdash_H (F \rightarrow G) \rightarrow (F \rightarrow H)$.

with *HC.MP* $\langle AX0 \cup \Gamma \vdash_H F \rightarrow G \rangle$ show $AX0 \cup \Gamma \vdash_H F \rightarrow H$.

qed

end

end

theory *HC-Compl-Consistency*

imports *Consistency HC*

begin

context begin

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

by (*metis AX100 Deduction-theorem Un-insert-right sup-left-commute*)

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

using *MP dt by blast*

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

using *MP dt by (metis Un-insert-left)*

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

proof *goal-cases*

case 1

have 2: $\Gamma \cup AX10 \vdash_H F \rightarrow H$ **by** (*simp add: 1 dt*)

have 3: $\Gamma \cup AX10 \vdash_H G \rightarrow H$ **by** (*simp add: 1 dt*)

have 4: $\Gamma \cup AX10 \vdash_H (F \vee G) \rightarrow H$ **by** (*meson 2 3 HC.simps HC-intros(7) HC-mono sup-ge2*)

thus *?case* **using** 1(3) *MP by blast*

qed

private lemma *someax*: $\Gamma \cup AX10 \vdash_H F \rightarrow \neg F \rightarrow \perp$

proof –

have $F \triangleright \Gamma \cup AX10 \vdash_H \neg F \rightarrow F \rightarrow \perp$

by (*meson HC-intros(12) HC-mono subset-insertI sup-ge2*)

then have $\neg F \triangleright F \triangleright \Gamma \cup AX10 \vdash_H \perp$

by (*meson HC.simps HC-mono insertI1 subset-insertI*)

then show *?thesis*

by (*metis (no-types) Un-insert-left dt*)

qed

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

proof –

thm *HC-intros(7)[of F ⊥ Not F]*

have $F \triangleright \Gamma \cup AX10 \vdash_H (\neg F \vee F)$

by (*metis AX10.intros(3) Ax HC-mono MP Un-commute Un-insert-left insertI1 sup-ge1*)

hence $F \triangleright \Gamma \cup AX10 \vdash_H \neg(\neg F \vee F) \rightarrow \perp$ **using** *someax* **by** (*metis HC.simps Un-insert-left*)

hence $\neg(\neg F \vee F) \triangleright F \triangleright \Gamma \cup AX10 \vdash_H \perp$ **by** (*meson Ax HC-mono MP insertI1 subset-insertI*)

hence $\neg(\neg F \vee F) \triangleright \Gamma \cup AX10 \vdash_H F \rightarrow \perp$

by (*metis Un-insert-left dt insert-commute*)

have $\neg F \triangleright \Gamma \cup AX10 \vdash_H (\neg F \vee F)$
by (*metis HC.simps HC-intros(5) HC-mono inf-sup-ord(4) insertI1 insert-is-Un*)
hence $\neg F \triangleright \Gamma \cup AX10 \vdash_H \neg (\neg F \vee F) \rightarrow \perp$ **using** *someax* **by** (*metis HC.simps Un-insert-left*)
hence $\neg (\neg F \vee F) \triangleright \neg F \triangleright \Gamma \cup AX10 \vdash_H \perp$ **by** (*meson Ax HC-mono MP insertI1 subset-insertI*)
hence $\neg (\neg F \vee F) \triangleright \Gamma \cup AX10 \vdash_H \neg F \rightarrow \perp$
by (*metis Un-insert-left dt insert-commute*)

hence $\Gamma \cup AX10 \vdash_H \neg (\neg F \vee F) \rightarrow \perp$
by (*meson HC-intros(13) HC-mono MP* $\langle \neg (\neg F \vee F) \triangleright \Gamma \cup AX10 \vdash_H F \rightarrow \perp \rangle$ *dt subset-insertI sup-ge2*)
thus *?thesis* **by** (*meson HC.simps HC-intros(13) HC-mono sup-ge2*)

qed

lemma *exchg*: $\Gamma \cup AX10 \vdash_H F \vee G \implies \Gamma \cup AX10 \vdash_H G \vee F$
by (*meson AX10.intros(3) HC.simps HC-intros(5) HC-intros(7) HC-mono sup-ge2*)

lemma *lem2*: $\Gamma \cup AX10 \vdash_H F \vee \neg F$ **by** (*simp add: exchg lem*)

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

proof *goal-cases case 1*

have $\Gamma \cup AX10 \vdash_H F \rightarrow \neg F \vee G$

proof $-$

have $f1: \forall F f Fa. \neg (F \vdash_H f) \vee \neg F \subseteq Fa \vee Fa \vdash_H f$

using *HC-mono* **by** *blast*

then have $f2: F \triangleright \Gamma \cup AX10 \vdash_H F \rightarrow G$

by (*metis 1 subset-insertI*)

have $\Gamma \cup AX10 \vdash_H G \rightarrow \neg F \vee G$

using $f1$ **by** *blast*

then show *?thesis*

using $f2 f1$ **by** (*metis (no-types) HC.simps dt insertI1 subset-insertI*)

qed

moreover have $\Gamma \cup AX10 \vdash_H \neg F \rightarrow \neg F \vee G$ **by** (*simp add: AX10.intros(2) Ax*)

ultimately show *?case*

proof $-$

have $\bigwedge F f fa fb. \neg (F \vdash_H f \rightarrow fa) \vee \neg (fb \triangleright F \vdash_H f) \vee fb \triangleright F \vdash_H fa$

by (*meson HC-mono MP subset-insertI*)

then show *?thesis*

by (*metis Ax* $\langle \Gamma \cup AX10 \vdash_H F \rightarrow \neg F \vee G \rangle$ $\langle \Gamma \cup AX10 \vdash_H \neg F \rightarrow \neg F \vee G \rangle$ *insertI1 lem sim-disj*)

qed

qed

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

by (*meson HC-intros(13) HC-mono MP dt subset-insertI sup-ge2*)

lemma *HC-case-distinction*: $\Gamma \cup AX10 \vdash_H F \rightarrow G \implies \Gamma \cup AX10 \vdash_H \neg F \rightarrow G \implies \Gamma \cup AX10 \vdash_H G$
using *HC-intros(7)*[of *F G Not F*] *lem2*
by (*metis (no-types, hide-lams) HC.simps HC-mono insertI1 sim-disj subset-insertI*)

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

proof *goal-cases case 1*

have $\Gamma \cup AX10 \vdash_H F \rightarrow G \rightarrow F \wedge G$ **by** (*simp add: AX10.intros(7) Ax*)

hence $2: F \triangleright G \triangleright \Gamma \cup AX10 \vdash_H F \wedge G$

by (*meson Ax HC-mono MP insertI1 subset-insertI*)

hence $3: F \triangleright G \triangleright \Gamma \cup AX10 \vdash_H \perp$ **using** *1* **by** (*meson HC-intros(12) HC-mono MP subset-insertI sup-ge2*)

from *2* **have** $\Gamma \cup AX10 \vdash_H G \rightarrow F \rightarrow F \wedge G$ **by** (*metis Un-insert-left dt*)

have $4: \Gamma \cup AX10 \vdash_H \neg F \rightarrow \neg F \vee \neg G$ **by** (*simp add: AX10.intros(2) Ax*)

have $5: \Gamma \cup AX10 \vdash_H \neg G \rightarrow F \rightarrow \neg F \vee \neg G$

by (*metis (full-types) AX10.intros(3) AX100 Ax HC-mono MP Un-assoc Un-insert-left dt inf-sup-ord(4) insertI1 subset-insertI sup-ge2*)

have $6: \Gamma \cup AX10 \vdash_H G \rightarrow F \rightarrow \neg F \vee \neg G$ **using** *3 inpcp* **by** (*metis Un-insert-left dt*)

have $7: \Gamma \cup AX10 \vdash_H F \rightarrow \neg F \vee \neg G$ **using** *5 6 HC-case-distinction* **by** *blast*

show *?case* **using** *4 7 HC-case-distinction* **by** *blast*

qed

lemma *HC-contrapos-nn*:

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

proof *goal-cases case 1*

from *1(1)* **have** $\Gamma \cup AX10 \vdash_H F \rightarrow \perp$ **using** *HC-intros(12)* **using** *HC-mono MP* **by** *blast*

hence $\Gamma \cup AX10 \vdash_H G \rightarrow \perp$ **by** (*meson 1(2) HC.intros(1) HC-mono MP dt insertI1 subset-insertI*)

thus *?case* **by**(*meson HC-intros(11) HC-intros(3) HC-mono MP sup-ge2*)

qed

lemma *nor-sim*:

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

shows $\Gamma \cup AX10 \vdash_H \neg F \quad \Gamma \cup AX10 \vdash_H \neg G$

using *HC-contrapos-nn assms* **by** (*metis HC-intros(5,6) HC-mono sup-ge2*)+

lemma *HC-contrapos-np*:

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

by (*meson HC-intros(12) HC-intros(13) HC-mono MP sup-ge2 HC-contrapos-nn*[of $\Gamma F \text{Not } G$])

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

proof *goal-cases case 1*

have $\Gamma \cup AX10 \vdash_H \neg F \rightarrow F \rightarrow \perp$ **by** (*simp add: AX10.intros(9) Ax*)

hence $\neg F \triangleright F \triangleright \Gamma \cup AX10 \vdash_H \perp$ **by** (*meson HC.simps HC-mono insertI1*)

subset-insertI)
hence $\neg F \triangleright F \triangleright \Gamma \cup AX10 \vdash_H G$ **by** (*metis (no-types, hide-lams) Un-commute Un-insert-right inpcp*)
thus *?case* **by** (*metis Un-insert-left dt insert-commute*)
qed

lemma *HC-consistent*: *pcp* $\{\Gamma \mid \Gamma. \neg(\Gamma \cup AX10 \vdash_H \perp)\}$
unfolding *pcp-def*
apply(*intro ballI conjI; unfold mem-Collect-eq; elim exE conjE; erule contrapos-np; clarsimp*)
subgoal by (*simp add: HC.Ax*)
subgoal by (*meson Ax HC-intros(12) HC-mono MP Un-upper1 sup-ge2*)
subgoal using *sim-conj* **by** (*metis (no-types, lifting) Ax HC-intros(8) HC-intros(9) HC-mono MP sup-ge1 sup-ge2*)
subgoal using *sim-disj* **using** *Ax* **by** *blast*
subgoal by (*erule (1) sim-disj*) (*simp add: Ax imp-sim*)
subgoal by (*metis Ax HC-contrapos-nn MP Un-iff Un-insert-left dt inpcp someax*)
subgoal by(*erule (1) sim-disj*) (*simp add: Ax nand-sim*)
subgoal by(*erule sim-conj*) (*meson Ax Un-iff nor-sim*)+
subgoal for $\Gamma F G$ **apply**(*erule sim-conj*)
subgoal by (*meson Ax HC-Compl-Consistency.not-imp HC-contrapos-np Un-iff*)
subgoal by (*metis Ax HC-contrapos-nn HC-intros(3) HC-mono sup-ge1 sup-ge2*)
done
done

end

corollary *HC-complete*:
fixes $F :: 'a :: \text{countable formula}$
shows $\models F \implies AX10 \vdash_H F$
proof(*erule contrapos-pp*)
let $?W = \{\Gamma \mid \Gamma. \neg(\Gamma :: ('a :: \text{countable}) \text{formula set}) \cup AX10 \vdash_H \perp\}$
note *[[show-types]]*
assume $\langle \neg (AX10 \vdash_H F) \rangle$
hence $\neg (\neg F \triangleright AX10 \vdash_H \perp)$
by (*metis AX100 Deduction-theorem HC-intros(13) MP Un-insert-right*)
hence $\{\neg F\} \in ?W$ **by** *simp*
with *pcp-sat HC-consistent* **have** $\text{sat } \{\neg F\}$.
thus $\neg \models F$ **by** (*simp add: sat-def*)
qed

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 (-  $\vdash$  - [30] 28)
where
  Ass:  $C \in S \Longrightarrow S \vdash C$  |
  R:  $S \vdash C \Longrightarrow S \vdash D \Longrightarrow k^+ \in C \Longrightarrow k^{-1} \in D \Longrightarrow S \vdash (C - \{k^+\}) \cup (D - \{k^{-1}\})$ 

```

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

```

lemma Resolution-weaken:  $S \vdash D \Longrightarrow T \cup S \vdash D$ 
  by(induction rule: Resolution.induct; auto intro: Resolution.intros)

```

```

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
  assume ?l
  from <?l> sd show ?r
  proof(induction T  $\cup$  S D rule: Resolution.induct)
    case (Ass D)
    show ?case proof cases
      assume  $D \in S$  with Resolution.Ass show ?thesis .
    next
      assume  $D \notin S$ 
      with Ass.hyps have  $D \in T$  by simp
      with Ass.prem show ?thesis by blast
    qed
  next
    case (R D H k) thus ?case by (simp add: Resolution.R)
  qed
next
  assume ?r with Resolution-weaken show ?l by blast
qed

```

```

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

```

```

proof(induction S  $\cup$  T C rule: Resolution.induct)
  case (Ass C)
  show ?case proof(cases C  $\in$  S)

```

```

case True
hence  $D \cup C \in op \cup D \text{ ' } S \cup T$  by simp
with Resolution.Ass have  $(op \cup D \text{ ' } S) \cup T \vdash D \cup C$  .
thus ?thesis by blast
next
case False
with Ass have  $C \in T$  by simp
hence  $(op \cup D \text{ ' } S) \cup T \vdash C$  by(simp add: Resolution.Ass)
thus ?thesis by(intro exI[where x={}]) simp
qed
next
case  $(R \ C1 \ C2 \ k)$ 
let  $?m = (op \cup D \text{ ' } S) \cup T$ 
from  $R$  obtain  $R1$  where  $IH1: R1 \subseteq D \ ?m \vdash R1 \cup C1$  by blast
from  $R$  obtain  $R2$  where  $IH2: R2 \subseteq D \ ?m \vdash R2 \cup C2$  by blast
from  $R$  have  $k^+ \in R1 \cup C1 \ k^{-1} \in R2 \cup C2$  by simp-all
note Resolution.R[OF IH1(2) IH2(2) this]
hence  $?m \vdash (R1 - \{k^+\}) \cup (R2 - \{k^{-1}\}) \cup (C1 - \{k^+\}) \cup (C2 - \{k^{-1}\})$ 
by (simp add: Un-Diff Un-left-commute sup.assoc)
moreover have  $(R1 - \{k^+\}) \cup (R2 - \{k^{-1}\}) \subseteq D$ 
using  $IH1(1) \ IH2(1)$  by blast
ultimately show ?case by blast
qed

```

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

using *assms* **proof**(*induction rule: Resolution.induct*)

case $(\text{Ass } C \ S)$ **thus** *?case* **using** *Resolution.Ass* **by**(*intro exI[where x={C}]*)

auto

next

case $(R \ S \ C \ D \ k)$

from R .*prems* **have** *finite C finite D* **by** *simp-all*

with R .*IH* **obtain** $SC \ SD$ **where** IH :

$SC \subseteq S \ (\forall C \in SC. \text{finite } C) \ \text{finite } SC \ SC \vdash C$

$SD \subseteq S \ (\forall D \in SD. \text{finite } D) \ \text{finite } SD \ SD \vdash D$

by *blast*

```

hence IHw:  $SC \cup SD \vdash C$   $SC \cup SD \vdash D$  using Resolution-weaken
  by(simp-all add: sup-commute Resolution-weaken )
with IH(1-3,5-7) show ?case
  by(blast intro!: exI[where  $x=SC \cup SD$ ] Resolution.R[OF - -  $\langle k^+ \in C \rangle \langle k^{-1} \in D \rangle$ ])
qed

```

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 =
  ( $\bigcup C1 \in S. \bigcup C2 \in S. \bigcup L1 \in C1. \bigcup L2 \in C2.$ 
   (case (L1,L2) of (Pos i,Neg j)  $\Rightarrow$  if  $i=j$  then  $\{(C1 - \{Pos\ i\}) \cup (C2 - \{Neg\ j\})\}$  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 ( $\lambda S. \square \notin S \wedge \neg res\ S \subseteq S$ ) ( $\lambda S. res\ S \cup S$ )

```

```

value [code] Rwhile ex1
lemma  $\square \in the$  (Rwhile ex1) by eval

```

```

lemma Rwhile-sound: assumes Rwhile S = Some S'
  shows  $\forall C \in S'. Resolution\ S\ C$ 
apply(rule while-option-rule[OF - assms[unfolded Rwhile-def]])
apply (auto simp: Ass R res-def split: if-splits literal.splits)
done

```

```

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

```

```

lemma s-sub-all-clauses:  $S \subseteq all-clauses\ S$ 
  unfolding all-clauses-def
  apply(rule)
  apply(simp)
  apply(rule)
  apply(simp add: atoms-of-cnf-alt lit-atoms-cases[abs-def])
  by (metis imageI literal.exhaust literal.simps(5) literal.simps(6))
lemma atoms-res:  $atoms-of-cnf\ (res\ S) \subseteq atoms-of-cnf\ S$ 
  unfolding res-def atoms-of-cnf-alt
  by(clarsimp simp: lit-atoms-cases[abs-def] image-def split: literal.splits if-splits)
  blast

```



```

lemma exlitE: ( $\bigwedge x. xe = Pos\ x \implies P\ x$ )  $\implies$  ( $\bigwedge x. xe = Neg\ x \implies False$ )  $\implies$ 
 $\exists x. xe = Pos\ x \wedge P\ x$ 
  by(cases xe) auto
lemma res-in-all-clauses:  $res\ S \subseteq all-clauses\ S$ 

  apply(clarsimp simp: res-def all-clauses-def atoms-of-cnf-alt image-def lit-atoms-cases
    split: literal.splits if-splits)
  apply(rule exlitE)
  apply (blast+)
done

lemma Res-in-all-clauses:  $res\ S \cup S \subseteq all-clauses\ S$ 
  by (simp add: res-in-all-clauses s-sub-all-clauses)
lemma all-clauses-Res-inv:  $all-clauses\ (res\ S \cup S) = all-clauses\ S$ 
  unfolding all-clauses-def atoms-of-cnf-Un
  using atoms-res by fast
lemma all-clauses-finite:  $finite\ S \wedge (\forall C \in S. finite\ C) \implies finite\ (all-clauses\ S)$ 
  unfolding all-clauses-def atoms-of-cnf-def by simp
lemma finite-res:  $\forall C \in S. finite\ C \implies \forall C \in res\ S. finite\ C$ 
  unfolding res-def by(clarsimp split: literal.splits)

lemma finite T  $\implies S \subseteq T \implies card\ S < Suc\ (card\ T)$ 
  by (simp add: card-mono le-imp-less-Suc)

lemma finite S  $\wedge (\forall C \in S. finite\ C) \implies \exists T. Rwhile\ S = Some\ T$ 
  apply(unfold Rwhile-def)
  apply(rule measure-while-option-Some[rotated, where f= $\lambda T. Suc\ (card\ (all-clauses\ S)) - card\ T$ 
    and P= $\lambda T. finite\ T \wedge (\forall C \in T. finite\ C) \wedge all-clauses\ T = all-clauses\ S$ 
  apply(simp;fail)
  apply(intro conjI)
  subgoal by (meson all-clauses-finite finite-UnI finite-subset res-in-all-clauses)
  subgoal using finite-res by blast
  subgoal using all-clauses-Res-inv by blast
  subgoal
    apply(rule diff-less-mono2)
    subgoal by (metis Res-in-all-clauses all-clauses-finite card-seteq finite-subset
not-le sup-commute sup-ge2)
    subgoal apply(intro card-mono le-imp-less-Suc)
    subgoal using all-clauses-finite by blast
    subgoal using s-sub-all-clauses by blast
  done
done
done

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

```

```

value [code] Res ex1
lemma  $\square \in \text{the } (Res \text{ ex1})$  by code-simp

lemma res:  $C \in \text{res } S \implies S \vdash C$ 
  unfolding res-def by(auto split: literal.splits if-splits) (metis Resolution.simps
  literal.exhaust)

lemma Res-sound:  $Res \ S = \text{Some } S' \implies (\forall C \in S'. S \vdash C)$ 
proof (induction arbitrary: S S' rule: Res.fixp-induct)
  fix X S S'
  assume IH:  $\bigwedge S S'. X \ S = \text{Some } S' \implies (\forall C \in S'. S \vdash C)$ 
  assume prem: (let R = res S  $\cup$  S in if R = S then Some S else X R) = Some S'
  thus  $(\forall C \in S'. S \vdash C)$ 
  proof cases
    assume res S  $\cup$  S = S
    with prem show ?thesis by (simp add: Resolution.Ass)
  next
    assume 1: res S  $\cup$  S  $\neq$  S
    with prem have X (res S  $\cup$  S) = Some S' by simp
    with IH have  $\forall C \in S'. \text{res } S \cup S \vdash C$  by blast
    thus ?thesis using Resolution-unnecessary res by blast
  qed
qed (fast intro!: option-admissible)+

lemma Res-terminates: finite S  $\implies \forall C \in S. \text{finite } C \implies \exists T. Res \ S = \text{Some } T$ 
proof(induction card (all-clauses S) – card S arbitrary: S rule: less-induct)
  case less
    let ?r = res S  $\cup$  S
    show ?case proof(cases ?r = S)
      case False
        have b: finite (res S  $\cup$  S) by (meson less Res-in-all-clauses all-clauses-finite
        infinite-super)
        have c: Ball (res S  $\cup$  S) finite using less.prem(2) finite-res by auto
        have card S < card ?r by (metis False b psubsetI psubset-card-mono sup-ge2)
        moreover have card ?r  $\leq$  (card (all-clauses S))
          by (meson less Res-in-all-clauses all-clauses-finite card-mono le-imp-less-Suc)
        ultimately have a: (card (all-clauses ?r)) – card ?r < (card (all-clauses S))
        – card S
          using all-clauses-Res-inv[of S] by simp
        from less(1)[OF a b c] show ?thesis by (subst Res.simps) (simp add: Let-def)
      qed (simp add: Res.simps)
    qed

```

code-pred Resolution .
print-theorems

end

```

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

lemma Resolution-insert:  $S \vdash R \implies \text{cnf-semantic} \mathcal{A} S \implies \text{cnf-semantic} \mathcal{A} \{R\}$ 
by(induction rule: Resolution.induct;
    clarsimp simp add: cnf-semantic-def clause-semantic-def lit-semantic-cases
    split: literal.splits;
    blast)

lemma  $S \vdash R \implies \text{cnf-semantic} \mathcal{A} S \longleftrightarrow \text{cnf-semantic} \mathcal{A} (R \triangleright S)$ 
    using Resolution-insert cnf-semantic-def by (metis insert-iff)

corollary Resolution-cnf-sound: assumes  $S \vdash \square$  shows  $\neg \text{cnf-semantic} \mathcal{A} S$ 
proof(rule notI)
    assume cnf-semantic  $\mathcal{A} S$ 
    with Resolution-insert assms have cnf-semantic  $\mathcal{A} \{\square\}$  .
    thus False by(simp add: cnf-semantic-def clause-semantic-def)
qed

corollary Resolution-sound:
    assumes rp: cnf (nnf F)  $\vdash \square$ 
    shows  $\neg \mathcal{A} \models F$ 
proof –
    from Resolution-cnf-sound rp have  $\neg \text{cnf-semantic} \mathcal{A} (\text{cnf} (\text{nnf} F))$  .
    hence  $\neg \mathcal{A} \models \text{nnf} F$  unfolding cnf-semantic[OF is-nnf-nnf] .
    thus ?thesis unfolding nnf-semantic .
qed

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: atoms-of-cnf (restrict-cnf-atom a v c) \subseteq*

atoms-of-cnf $c - \{a\}$
unfolding *restrict-cnf-atom-def atoms-of-cnf-alt lit-atoms-cases make-lit-def*
by (*force split: literal.splits bool.splits*)

lemma *cnf-substitution-lemma:*

cnf-semantic A (*restrict-cnf-atom* a v S) = *cnf-semantic* ($A(a := v)$) S

unfolding *restrict-cnf-atom-def cnf-semantic-def clause-semantic-def lit-semantic-cases make-lit-def*

apply (*clarsimp split: bool.splits literal.splits*)

apply *safe*

subgoal for s **by** (*fastforce elim!: allE[of - $s - \{Neg\} a]$]*)

subgoal by (*metis DiffI singletonD*)

subgoal for s **by** (*fastforce elim!: allE[of - $s - \{Pos\} a]$]*)

subgoal by (*metis DiffI singletonD*)

done

lemma *finite-restrict: finite* $S \implies$ *finite* (*restrict-cnf-atom* a v S)

unfolding *restrict-cnf-atom-def* **by** (*simp add: image-iff*)

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:*

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

proof –

have $\llbracket xa \in \text{restrict-cnf-atom } a \text{ } v \text{ } S; \{make\text{-lit } (\neg v) a\} \cup xa \notin S; x = xa \rrbracket \implies xa \in S$ **for** x xa

unfolding *restrict-cnf-atom-def* **using** *insert-Diff* **by** *fastforce*

hence $x \in (\lambda c. \text{if } \{make\text{-lit } (\neg v) a\} \cup c \in S \text{ then } \{make\text{-lit } (\neg v) a\} \cup c \text{ else } c)$ ‘*restrict-cnf-atom* a v $S \implies x \in S$ **for** x

unfolding *image-iff* **by** (*elim bexE*) *simp*

thus *?thesis ..*

qed

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

assumes *pr: S* \vdash \square

assumes *su: S* \subseteq T

shows $\exists R \subseteq \{make\text{-lit } v a\}. (\lambda c. \text{if } c \in n \text{ then } \{make\text{-lit } v a\} \cup c \text{ else } c)$ ‘ T $\vdash R$

proof –

from *Resolution-taint-assumptions* [**where** $D = \{make\text{-lit } v a\}$]

have *taint: $\Gamma \cup \Lambda \vdash \square \implies \exists R \subseteq \{make\text{-lit } v a\}. \text{insert } (make\text{-lit } v a)$* ‘ $\Gamma \cup \Lambda \vdash R$

for Γ Λ **by** (*metis image-cong insert-def sup-bot.right-neutral*)

have $S: S = \{c \in S. c \in n\} \cup \{c \in S. c \notin n\}$ **by** *blast*

hence $SI: (\lambda c. \text{if } c \in n \text{ then } \{make\text{-lit } v a\} \cup c \text{ else } c)$ ‘ $S = (\text{insert } (make\text{-lit } v a)$ ‘ $\{c \in S. c \in n\}) \cup \{c \in S. c \notin n\}$

by *auto*

```

from pr have  $\exists R \subseteq \{make\text{-}lit\ v\ a\}$ .
( $\lambda c. \text{if } c \in n \text{ then } \{make\text{-}lit\ v\ a\} \cup c \text{ else } c$ ) ‘  $S \vdash R$ 
  apply(subst SI)
  apply(subst(asm) S)
  apply(elim taint)
done
thus ?thesis using Resolution-weaken su by (metis (no-types, lifting) image-Un
sup.order-iff)
qed

```

```

lemma unrestrict':
  fixes R :: 'a clause
  assumes rp: restrict-cnf-atom a v S  $\vdash \square$ 
  shows  $\exists R \subseteq \{make\text{-}lit\ (\neg v)\ a\}. S \vdash R$ 
proof –
  fix C :: 'a clause fix k :: 'a
  from unrestrict-effects[of v a S]

```

The idea is that the restricted set lost some clauses, and that some others were crippled. So, there must be a set of clauses to heal and a set of clauses to reinsert to get the original. (Mind you, this is not exactly what is happening, because e.g. both C and $\{k^{-1}\} \cup C$ might be in there and get reduced to one C . You then heal that C to $\{k^{-1}\} \cup C$ and insert the shadowed C ... Details.)

```

obtain n where S:
  ( $\lambda c. \text{if } c \in n \text{ then } \{make\text{-}lit\ (\neg v)\ a\} \cup c \text{ else } c$ ) ‘ restrict-cnf-atom a v S  $\subseteq S$ 
  using exI[where x={c. {make-lit (¬v) a} ∪ c ∈ S}] by force
note finite-restrict S
show ?thesis using can-cope-with-unrestrict-effects[OF rp]
  by (metis (no-types) S Resolution-weaken subset-refl sup.order-iff)
qed

```

```

lemma Resolution-complete-standalone-finite:
  assumes ns:  $\forall \mathcal{A}. \neg cnf\text{-}semantics\ \mathcal{A}\ S$ 
  assumes fin: finite (atoms-of-cnf S)
  shows  $S \vdash \square$ 
using fin ns
proof(induction atoms-of-cnf S arbitrary: S rule: finite-psubset-induct)
  case psubset
  show ?case proof(cases)
    assume e: atoms-of-cnf S =  $\{\}$ 
    from  $\langle \forall \mathcal{A}. \neg cnf\text{-}semantics\ \mathcal{A}\ S \rangle$  have  $S \neq \{\}$  unfolding cnf-semantics-def
by blast
    with e have  $S = \{\square\}$  unfolding atoms-of-cnf-def by simp fast
    thus ?case using Resolution.Ass by blast
  next
  have unsat-restrict:  $\forall \mathcal{A}. \neg cnf\text{-}semantics\ \mathcal{A}\ (restrict\text{-}cnf\text{-}atom\ a\ v\ S)$  for a v
    using  $\langle \forall \mathcal{A}. \neg cnf\text{-}semantics\ \mathcal{A}\ S \rangle$  by(simp add: cnf-substitution-lemma)
    assume ne: atoms-of-cnf S  $\neq \{\}$ 

```

```

then obtain  $a$  where  $a \in \text{atoms-of-cnf } S$  by blast
hence  $\text{atoms-of-cnf } (\text{restrict-cnf-atom } a \ v \ S) \subset \text{atoms-of-cnf } S$  for  $v$ 
using restrict-cnf-remove [where  $'a='a$ ] by blast
from psubset(2) [OF this unsat-restrict]
have  $IH: \text{restrict-cnf-atom } a \ v \ S \vdash \square$  for  $v$  .
from unrestrict' [OF IH, of  $\neg$  -] have  $\text{unr-IH}: \exists R \subseteq \{\text{make-lit } v \ a\}. S \vdash R$ 
for  $v$  by simp
from this [of False] this [of True] show ?case using Resolution.R [OF - - singletonI singletonI]
by (simp add: make-lit-def) (fast dest: subset-singletonD)
qed
qed

```

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':  $(\neg P \implies Q) \implies P \vee Q$  by auto

```

```

lemma atomD:  $\text{Atom } k \in S \implies \{\text{Pos } k\} \in \text{UNION } S \ \text{cnf } \text{Not } (\text{Atom } k) \in S \implies \{\text{Neg } k\} \in \text{UNION } S \ \text{cnf}$  by force+

```

```

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 goal-cases

```

```

case 1

```

```

from 1(1,4) have  $\text{is-cnf } (F \vee G)$  by blast

```

```

hence db: is-disj F is-lit-plus F is-disj G by (cases F; simp)+

```

```

hence  $\text{is-cnf } F \wedge \text{is-cnf } G$  by (cases F; cases G; simp)

```

```

with 1 have  $IH: (\bigcup (\text{cnf } ' (F \triangleright \Gamma))) \vdash \square \ (\bigcup (\text{cnf } ' (G \triangleright \Gamma))) \vdash \square$  by simp-all

```

```

let  $? \Gamma = (\bigcup (\text{cnf } ' \Gamma))$ 

```

```

from  $IH$  have  $IH\text{-readable}: \text{cnf } F \cup ? \Gamma \vdash \square \ \text{cnf } G \cup ? \Gamma \vdash \square$  by auto

```

```

show ?case proof (cases cnf F = {})  $\vee$  (cases cnf G = {})

```

```

case True

```

```

hence  $\text{cnf } (F \vee G) = \{\}$  by auto

```

```

thus ?thesis using True IH by auto

```

```

next
  case False
  then obtain S T where ST: cnf F = {S} cnf G = {T}
    using cnf-disj-ex db(1,3) by metis

  hence R: cnf (F ∨ G) = { S ∪ T } by simp
  have  $\llbracket S \triangleright ?\Gamma \vdash \square; T \triangleright ?\Gamma \vdash \square \rrbracket \implies S \cup T \triangleright ?\Gamma \vdash \square$  proof –
    assume s: S  $\triangleright$  ?Γ  $\vdash$   $\square$  and t: T  $\triangleright$  ?Γ  $\vdash$   $\square$ 
    hence s-w: S  $\triangleright$  S  $\cup$  T  $\triangleright$  ?Γ  $\vdash$   $\square$  using Resolution-weaken by (metis
insert-commute insert-is-Un)
    note Resolution-taint-assumptions[of {T} ?Γ □ S] t
    then obtain R where R: S  $\cup$  T  $\triangleright$  ?Γ  $\vdash$  R R  $\subseteq$  S by (auto simp: Un-commute)
    have literal-subset-sandwich: R =  $\square \vee R = S$  if is-lit-plus F cnf F = {S} R
 $\subseteq S$ 
    using that by (cases F rule: is-lit-plus.cases; simp) blast+
    show ?thesis using literal-subset-sandwich[OF db(2) ST(1) R(2)] proof
      assume R =  $\square$  thus ?thesis using R(1) by blast
    next
      from Resolution-unnecessary[where T={-}, simplified] R(1)
      have  $(R \triangleright S \cup T \triangleright ?\Gamma \vdash \square) = (S \cup T \triangleright ?\Gamma \vdash \square)$  .
      moreover assume R = S
      ultimately show ?thesis using s-w by simp
    qed
  qed
  thus ?thesis using IH ST R 1(1) by (metis UN-insert insert-absorb insert-is-Un)
  qed
  qed

```

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

```

  unfolding pcp-def
  unfolding Ball-def
  unfolding mem-Collect-eq
  apply (intro allI impI)
  apply (erule contrapos-pp)
  apply (unfold not-ex de-Morgan-conj de-Morgan-disj not-not not-all not-imp disj-not1)
  apply (intro impI allI)
  apply (elim disjE exE conjE; intro OrI2')
  apply (unfold not-ex de-Morgan-conj de-Morgan-disj not-not not-all not-imp
disj-not1 Ball-def[symmetric])
  apply safe
  apply (metis Ass Pow-bottom Pow-empty UN-I cnf.simps(3))
  apply (metis Diff-insert-absorb Resolution.simps insert-absorb singletonI
sup-bot.right-neutral atomD)
  apply (simp; metis (no-types, hide-lams) UN-insert cnf.simps(5) insert-absorb
is-cnf.simps(1) sup-assoc)
  apply (auto intro: pcp-disj)
  done

```

theorem *Resolution-complete*:

```

fixes  $F :: 'a :: \text{countable formula}$ 
shows  $\models F \implies \text{cnf } (\text{nnf } (\neg F)) \vdash \square$ 
proof(erule contrapos-pp)
  assume  $c: \neg (\text{cnf } (\text{nnf } (\neg F)) \vdash \square)$ 
  have  $\{\text{cnf-form-of } (\text{nnf } (\neg F))\} \in \{\Gamma \mid \Gamma. \neg ((\forall \gamma \in \Gamma. \text{is-cnf } \gamma) \longrightarrow \text{UNION } \Gamma \text{ cnf } \vdash \square)\}$ 
  by(simp add: cnf-cnf[OF is-nnf-nnf] c cnf-form-of-is[OF is-nnf-nnf])
  from pcp-sat[OF R-consistent this] have  $\text{sat } \{\text{cnf-form-of } (\text{nnf } (\neg F))\}$  .
  thus  $\neg \models F$  by(simp add: sat-def cnf-form-antics[OF is-nnf-nnf] nnf-semantics)
qed

```

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!]:  $F, \Gamma \Rightarrow F, \Delta$  by (intro extended-Ax) (simp add: add.commute inter-add-right2)

```

```

theorem HCSC:  $AX10 \cup \text{set-mset } \Gamma \vdash_H F \implies \Gamma \Rightarrow \{\#F\#$ 
proof(induction rule: HC.induct)
  case (Ax F) thus ?case proof
    note SCp.intros(3-)[intro!]

```

Essentially, we need to prove all the axioms of Hilbert Calculus in Sequent Calculus.

```

  have  $A: \Gamma \Rightarrow \{\#F \rightarrow (F \vee G)\#\}$  for  $F G$  by blast
  have  $B: \Gamma \Rightarrow \{\#G \rightarrow (F \vee G)\#\}$  for  $G F$  by blast
  have  $C: \Gamma \Rightarrow \{\#(F \rightarrow H) \rightarrow ((G \rightarrow H) \rightarrow ((F \vee G) \rightarrow H))\#\}$  for  $F H G$ 
by blast
  have  $D: \Gamma \Rightarrow \{\#(F \wedge G) \rightarrow F\#\}$  for  $F G$  by blast
  have  $E: \Gamma \Rightarrow \{\#(F \wedge G) \rightarrow G\#\}$  for  $F G$  by blast
  have  $F: \Gamma \Rightarrow \{\#F \rightarrow (G \rightarrow (F \wedge G))\#\}$  for  $F G$  by blast
  have  $G: \Gamma \Rightarrow \{\#(F \rightarrow \perp) \rightarrow \neg F\#\}$  for  $F$  by blast
  have  $H: \Gamma \Rightarrow \{\#\neg F \rightarrow (F \rightarrow \perp)\#\}$  for  $F$  by blast
  have  $I: \Gamma \Rightarrow \{\#(\neg F \rightarrow \perp) \rightarrow F\#\}$  for  $F$  by blast
  have  $K: \Gamma \Rightarrow \{\#F \rightarrow (G \rightarrow F)\#\}$  for  $F G$  by blast
  have  $L: \Gamma \Rightarrow \{\#(F \rightarrow (G \rightarrow H)) \rightarrow ((F \rightarrow G) \rightarrow (F \rightarrow H))\#\}$  for  $F G H$ 
by blast

```



```

    have J:  $F \in AX0 \implies \Gamma \Rightarrow \{\#F\# \}$  for F by (induction rule: AX0.induct;
intro K L)
    assume  $F \in AX10$  thus ?thesis by (induction rule: AX10.induct; intro A B
C D E F G H I J)
  next
    assume  $F \in \text{set-mset } \Gamma$  thus ?thesis by (intro extended-Ax) simp
  qed
next
  case (MP F G)
  from MP have IH:  $\Gamma \Rightarrow \{\#F\# \} \Gamma \Rightarrow \{\#F \rightarrow G\# \}$  by blast+
  with ImpR-inv[where  $\Delta = \text{Mempty}$ , simplified] have  $F, \Gamma \Rightarrow \{\#G\# \}$  by auto
  moreover from IH(1) weakenR have  $\Gamma \Rightarrow F, \{\#G\# \}$  by blast
  ultimately show  $\Gamma \Rightarrow \{\#G\# \}$  using cut[where  $F=F$ ] by simp
qed

end

```

3.2 SC to ND

```

theory SCND
imports SC ND
begin

lemma SCND:  $\Gamma \Rightarrow \Delta \implies (\text{set-mset } \Gamma) \cup \text{Not } \text{' } (\text{set-mset } \Delta) \vdash \perp$ 
proof (induction  $\Gamma \Delta$  rule: SCp.induct)
  case BotL thus ?case by (simp add: ND.Ax)
next
  case Ax thus ?case by (meson ND.Ax NotE UnCI image-iff)
next
  case NotL thus ?case by (simp add: NotI)
next
  case (NotR F  $\Gamma \Delta$ ) thus ?case by (simp add: Not2IE)
next
  case (AndL F G  $\Gamma \Delta$ ) thus ?case by (simp add: AndL-sim)
next
  case (AndR  $\Gamma F \Delta G$ ) thus ?case by (simp add: AndR-sim)
next
  case OrL thus ?case by (simp add: OrL-sim)
next
  case OrR thus ?case using OrR-sim[where 'a='a] by (simp add: insert-commute)
next
  case (ImpL  $\Gamma F \Delta G$ ) from ImpL.IH show ?case by (simp add: ImpL-sim)
next
  case ImpR from ImpR.IH show ?case by (simp add: ImpR-sim)
qed

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(induction rule: ND.induct)
  case Ax thus ?case by(auto intro: HC.Ax)
next
  case NotE thus ?case by (meson AX10.intros(9) HC.simps subsetCE sup-ge1)
next
  case (NotI F  $\Gamma$ )
  from HC-intros(11) have HC-NotI:  $AX10 \cup \Gamma \vdash_H A \rightarrow \perp \implies AX10 \cup \Gamma \vdash_H \neg A$  for A
  using MP HC-mono by (metis sup-ge1)
  from NotI show ?case using Deduction-theorem[where  $\Gamma=AX10 \cup \Gamma$ ] HC-NotI
by (metis AX100 Un-assoc Un-insert-right)
next
  case (CC F  $\Gamma$ )
  hence  $AX10 \cup \Gamma \vdash_H \neg F \rightarrow \perp$  using Deduction-theorem[where  $\Gamma=AX10 \cup \Gamma$ ] by (metis AX100 Un-assoc Un-insert-right)
  thus  $AX10 \cup \Gamma \vdash_H F$  using AX10.intros(10) by (metis HC.simps UnCI)
next
  case (AndE1  $\Gamma F G$ ) thus ?case by (meson AX10.intros(5) HC.simps UnCI)
next
  case (AndE2  $\Gamma F G$ ) thus ?case by (meson AX10.intros(6) HC.simps UnCI)
next
  case (AndI  $\Gamma F G$ ) thus ?case by (meson HC-intros(10) HC-mono HC.simps sup-ge1)
next
  case (OrE  $\Gamma F G H$ )
  from  $\langle AX10 \cup (F \triangleright \Gamma) \vdash_H H \rangle \langle AX10 \cup (G \triangleright \Gamma) \vdash_H H \rangle$  have
     $AX10 \cup \Gamma \vdash_H F \rightarrow H$   $AX10 \cup \Gamma \vdash_H G \rightarrow H$ 
  using Deduction-theorem[where  $\Gamma=AX10 \cup \Gamma$ ] by (metis AX100 Un-assoc Un-insert-right)+
  with HC-intros(7)[THEN HC-mono[OF - sup-ge1]] MP
  have  $AX10 \cup \Gamma \vdash_H (F \vee G) \rightarrow H$  by metis
  with MP  $\langle AX10 \cup \Gamma \vdash_H F \vee G \rangle$  show ?case .
next
  case (OrI1  $\Gamma F G$ ) thus ?case by (meson AX10.intros(2) HC.simps UnCI)
next
  case (OrI2  $\Gamma F G$ ) thus ?case by (meson AX10.intros(3) HC.simps UnCI)
next
  case (ImpE  $\Gamma F G$ )

```

```

from MP  $\langle AX10 \cup \Gamma \vdash_H F \rangle \langle AX10 \cup \Gamma \vdash_H F \rightarrow G \rangle$  show ?case .
next
  case (ImpI  $F \Gamma G$ ) thus ?case using Deduction-theorem[where  $\Gamma = AX10 \cup \Gamma$ ]
by (metis AX100 Un-assoc Un-insert-right)
qed

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  $sc F \longleftrightarrow hc F$ 
using HCSC[where  $F = F$  and  $\Gamma = Mempty$ , simplified]
  SCND[where  $\Gamma = Mempty$  and  $\Delta = \{\#F\#\}$ ] ND.ND.CC[where  $F = F$  and
 $\Gamma = \{\}$ ]
  NDHC[where  $\Gamma = \{\}$  and  $F = F$ ]
by (simp-all add: assms) blast+

```

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: $F, G, \Gamma \Rightarrow_n \Longrightarrow F \wedge G, \Gamma \Rightarrow_n \mid$

OrL: $F, \Gamma \Rightarrow_n \Longrightarrow G, \Gamma \Rightarrow_n \Longrightarrow F \vee G, \Gamma \Rightarrow_n \mid$

— nnf rules:

NotOrNNF: $\neg F, \neg G, \Gamma \Rightarrow_n \Longrightarrow \neg(F \vee G), \Gamma \Rightarrow_n \mid$

NotAndNNF: $\neg F, \Gamma \Rightarrow_n \Longrightarrow \neg G, \Gamma \Rightarrow_n \Longrightarrow \neg(F \wedge G), \Gamma \Rightarrow_n \mid$

ImpNNF: $\neg F, \Gamma \Rightarrow_n \Longrightarrow G, \Gamma \Rightarrow_n \Longrightarrow F \rightarrow G, \Gamma \Rightarrow_n \mid$

NotImpNNF: $F, \neg G, \Gamma \Rightarrow_n \Longrightarrow \neg(F \rightarrow G), \Gamma \Rightarrow_n \mid$

NotNotNNF: $F, \Gamma \Rightarrow_n \Longrightarrow \neg(\neg F), \Gamma \Rightarrow_n$

lemmas *LSC.intros*[*intro!*]

You can prove that derivability in *SCp* is invariant to *nnf*, and then transform *SCp* to *LSC* while assuming NNF. However, the transformation introduces

the trouble of handling the right side of *SCp*. The idea behind this is that handling the transformation is easier when not requiring NNF.

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

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

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

proof(*induction* $\Gamma \{\#\}::'a \text{ formula multiset rule: } SCp.induct$)

case (*BotL* Γ)

then obtain Γ' **where** $\Gamma = \perp, \Gamma'$ **by**(*meson multi-member-split*)

thus *?case* **by** *auto*

next

case (*Ax A* Γ) **hence** *False* **by** *simp* **thus** *?case ..*

next

case (*AndL* $\Gamma F G$)

hence *IH*: $\Gamma, F, G \Rightarrow_n$ **by** *force*

thus *?case* **by** *auto*

next

case (*NotL*) **thus** *?case*

oops

lemma *LSC-to-SC*:

shows $\Gamma \Rightarrow_n \Longrightarrow \Gamma \Rightarrow \{\#\}$

proof(*induction rule: LSC.induct*)

qed (*auto dest!: NotL-inv intro!: SCp.intros(3-) intro: extended-Ax*)

lemma *SC-to-LSC*:

assumes $\Gamma \Rightarrow \Delta$

shows $\Gamma + (\text{image-mset } \text{Not } \Delta) \Rightarrow_n$

proof –

have *GO[simp]*:

NO-MATCH $\{\# B \#\} F \Longrightarrow (F, S) + T = F, (S+T)$

NO-MATCH $\{\# B \#\} S \Longrightarrow S + (F, T) = F, (S+T)$

NO-MATCH $(\neg H) F \Longrightarrow F, \neg G, S = \neg G, F, S$

for *B S F G H T* **by** *simp-all*

from *assms* **show** *?thesis*

proof(*induction rule: SCp.induct*)

case (*BotL* Γ)

then obtain Γ' **where** $\Gamma = \perp, \Gamma'$ **by**(*meson multi-member-split*)

thus *?case* **by** *auto*

next

case (*Ax k* $\Gamma \Delta$)

then obtain $\Gamma' \Delta'$ **where** $\Gamma = \text{Atom } k, \Gamma' \Delta = \text{Atom } k, \Delta'$ **by**(*meson multi-member-split*)

thus *?case* **using** *LSC.Ax* **by** *simp*

qed *auto*

qed

corollary *SC-LSC*: $\Gamma \Rightarrow \{\#\} \longleftrightarrow \Gamma \Rightarrow_n$ **using** *SC-to-LSC* *LSC-to-SC* **by** *fastforce*

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(*induction rule: LSC.induct*)

```
  case (NotOrNNF F G  $\Gamma$ )
  from NotOrNNF.IH have nnf ( $\neg$  F), nnf ( $\neg$  G), image-mset nnf  $\Gamma \Rightarrow_n$  by
simp
  with LSC.AndL have nnf ( $\neg$  F)  $\wedge$  nnf ( $\neg$  G), image-mset nnf  $\Gamma \Rightarrow_n$  .
  thus ?case by simp
next
  case (AndL F G  $\Gamma$ )
  from AndL.IH have nnf F, nnf G, image-mset nnf  $\Gamma \Rightarrow_n$  by simp
  with LSC.AndL[where 'a='a] have nnf F  $\wedge$  nnf G, image-mset nnf  $\Gamma \Rightarrow_n$  by
simp
  thus ?case by simp
next
qed (auto, metis add-mset-commute)
```

lemma *LSC-NNF-back*: $\text{image-mset nnf } \Gamma \Rightarrow_n \Longrightarrow \Gamma \Rightarrow_n$

proof(*induction image-mset nnf Γ rule: LSC.induct*)

oops

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 $\Gamma \Rightarrow_n$

using *assms*

unfolding *SC-LSC[symmetric]*

using *NotL-inv cut* **by** *blast*

lemma

shows *LSC-AndL-inv*: $F \wedge G, \Gamma \Rightarrow_n \Longrightarrow F, G, \Gamma \Rightarrow_n$

and *LSC-OrL-inv*: $F \vee G, \Gamma \Rightarrow_n \Longrightarrow F, \Gamma \Rightarrow_n \wedge G, \Gamma \Rightarrow_n$

using *SC-LSC AndL-inv OrL-inv* **by** *blast+*

lemmas *LSC-invs* = *LSC-AndL-inv* *LSC-OrL-inv*

lemma *LSC-weaken-set*: $\Gamma \Rightarrow_n \Longrightarrow \Gamma + \Theta \Rightarrow_n$
by(*induction rule*: *LSC.induct*) (*auto simp*: *add.assoc*)
lemma *LSC-weaken*: $\Gamma \Rightarrow_n \Longrightarrow F, \Gamma \Rightarrow_n$
using *LSC-weaken-set* **by** (*metis add-mset-add-single*)

lemma *LSC-Contract*:
assumes *sfp*: $F, F, \Gamma \Rightarrow_n$
shows $F, \Gamma \Rightarrow_n$
using *SC-LSC contractL sfp* **by** *blast*

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 –
have *GO[simp]*:
 $F, G, S \Rightarrow_n \Longrightarrow G, F, S \Rightarrow_n$
for $F, G :: 'a$ *formula* **and** S **by** (*simp add*: *add-mset-commute*)
have *?r1* **if** *?l1* **proof** –
from $\langle ?l1 \rangle$ [*THEN LSC-invs(2)*] **have** $f: F, \Gamma \Rightarrow_n G \wedge H, \Gamma \Rightarrow_n$ **by** *simp-all*
from $\langle ?l1 \rangle$ [*THEN LSC-invs(1)*] **have** $gh: G, H, \Gamma \Rightarrow_n$ **by** *simp*
show *?r1* **using** f, gh **by**(*auto dest!*: *LSC-invs simp*: *LSC-weaken*)
qed
moreover **have** *?r2* **if** *?l2* **proof** –
from $\langle ?l2 \rangle$ **have** $f: F, \Gamma \Rightarrow_n G, H, \Gamma \Rightarrow_n$ **by**(*auto dest!*: *LSC-invs*)
thus *?r2* **by**(*auto dest!*: *LSC-invs simp*: *LSC-weaken*)
qed
moreover **have** *?l1* **if** *?r1* **proof** –
from $\langle ?r1 \rangle$ [*THEN LSC-invs(1)*] **have** $*$: $F \vee G, F \vee H, \Gamma \Rightarrow_n$ **by** *simp*
hence $F, \Gamma \Rightarrow_n G, H, \Gamma \Rightarrow_n$ **by** (*auto elim!*: *LSC-Contract dest!*: *LSC-invs*)
thus *?l1* **by**(*intro LSC.intros*)
qed
moreover **have** *?l2* **if** *?r2* **proof** –
from $\langle ?r2 \rangle$ [*THEN LSC-invs(1)*] **have** $*$: $G \vee F, H \vee F, \Gamma \Rightarrow_n$ **by** *simp*
hence $F, \Gamma \Rightarrow_n G, H, \Gamma \Rightarrow_n$ **by**(*auto elim!*: *LSC-Contract dest!*: *LSC-invs*)
thus *?l2* **by**(*intro LSC.intros*)
qed
ultimately **show** $?l1 \longleftrightarrow ?r1$ $?l2 \longleftrightarrow ?r2$ **by** *argo+*
qed

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 –
have *GO[simp]*:

```

     $F, G, S \Rightarrow_n \implies G, F, S \Rightarrow_n$ 
    for  $F G H I J S$  by(simp-all add: add-mset-commute)
    show  $?t1 ?t2$  by(auto dest!: LSC-invs)
qed

```

lemma *LSC-sim-Resolution1*:

```

    assumes  $R: S \vee T, \Gamma \Rightarrow_n$ 
    shows  $Atom\ k \vee S, (\neg(Atom\ k)) \vee T, \Gamma \Rightarrow_n$ 
proof -
    from  $R$  have  $r: T, \Gamma \Rightarrow_n S, \Gamma \Rightarrow_n$  by(auto dest: LSC-invs)
    show  $?thesis$  proof(rule LHCut[where F=Atom k])
      have  $2: T, Atom\ k, Atom\ k \vee S, \Gamma \Rightarrow_n$  using LSC-weaken r(1) by auto
      hence  $\neg(Atom\ k) \vee T, Atom\ k, Atom\ k \vee S, \Gamma \Rightarrow_n$  by auto
      thus  $Atom\ k, Atom\ k \vee S, \neg(Atom\ k) \vee T, \Gamma \Rightarrow_n$ 
        by(auto dest!: LSC-invs) (metis add-mset-commute)
    next

```

analogously

```

    show  $\neg(Atom\ k), Atom\ k \vee S, \neg(Atom\ k) \vee T, \Gamma \Rightarrow_n$  by simp
qed
qed

```

lemma

```

    LSC-need-it-once-have-many:
    assumes  $el: A \in set\ F$ 
    assumes once: form-of-lit A  $\vee$  disj-of-clause (removeAll A F),  $\Gamma \Rightarrow_n$ 
    shows disj-of-clause F,  $\Gamma \Rightarrow_n$ 
using assms
proof(induction F)
  case Nil hence False by simp thus  $?case ..$ 
next
  case (Cons f F)
  thus  $?case$  proof(cases A = f)
    case True
    with Cons.prems have ihm: form-of-lit A  $\vee$  disj-of-clause (removeAll A F),  $\Gamma \Rightarrow_n$  by simp
    with True have split: form-of-lit f,  $\Gamma \Rightarrow_n$  disj-of-clause (removeAll A F),  $\Gamma \Rightarrow_n$ 
      by (auto dest!: LSC-invs(2))
    from Cons.IH[OF - ihm] have  $A \in set\ F \implies disj-of-clause\ F, \Gamma \Rightarrow_n .$ 
    with split(2) have disj-of-clause F,  $\Gamma \Rightarrow_n$  by(cases A  $\in$  set F) simp-all
    with split(1) show  $?thesis$  by auto
  next
  case False
  with Cons.prems(2) have prem: form-of-lit A,  $\Gamma \Rightarrow_n$  form-of-lit f,  $\Gamma \Rightarrow_n$ 
disj-of-clause (removeAll A F),  $\Gamma \Rightarrow_n$ 
    by (auto dest!: LSC-invs(2))
  hence d: form-of-lit A  $\vee$  disj-of-clause (removeAll A F),  $\Gamma \Rightarrow_n$  by blast
  from False Cons.prems have  $el: A \in set\ F$  by simp

```

from *Cons.IH*[*OF el d*] **have** *disj-of-clause* $F, \Gamma \Rightarrow_n$.
with *prem*(2) **show** *?thesis* **by** *auto*
qed
qed

lemma *LSC-Sim-resolution-la*:

fixes $k :: 'a$
assumes R : *disj-of-clause* (*removeAll* (k^+) $F @$ *removeAll* (k^{-1}) G), $\Gamma \Rightarrow_n$
assumes el : $k^+ \in \text{set } F$ $k^{-1} \in \text{set } G$
shows *disj-of-clause* $F, \text{disj-of-clause } G, \Gamma \Rightarrow_n$
proof –
have *LSC-or-assoc*: ($F \vee G$) $\vee H, \Gamma \Rightarrow_n \longleftrightarrow F \vee (G \vee H), \Gamma \Rightarrow_n$ **if** *is-nnf* F
is-nnf G *is-nnf* H **for** $F G H$
using *that* **by**(*auto dest!*: *LSC-invs*(2))
have dd : *disj-of-clause* ($F @ G$), $\Gamma \Rightarrow_n \implies \text{disj-of-clause } F \vee \text{disj-of-clause } G,$
 $\Gamma \Rightarrow_n$ **for** $F G$
by(*induction* F) (*auto dest!*: *LSC-invs*(2)) *simp add*: *LSC-or-assoc*
from *LSC-sim-Resolution1*[*OF dd*[*OF R*]]
have $unord$: *Atom* $k \vee \text{disj-of-clause} (*removeAll* (k^+) F), \neg (*Atom* k) $\vee \text{disj-of-clause}$
(*removeAll* (k^{-1}) G), $\Gamma \Rightarrow_n$.
show *?thesis*
using *LSC-need-it-once-have-many*[*OF el*(1)] *LSC-need-it-once-have-many*[*OF*
 el (2)] $unord$
by(*simp add*: *add-mset-commute del*: *sc-insertion-ordering*)
qed$

lemma *two-list-induct*[*case-names Nil Cons*]: $P [] [] \implies (\bigwedge a S T. P S T \implies P$
 $(a \# S) T \ \&\&\& \ P S (a \# T)) \implies P S T$
apply(*induction* S)
apply(*induction* T)
apply(*simp-all*)
done

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

proof(*induction* G *arbitrary*: Γ)

have GO [*simp*]:
NO-MATCH ($\{\#IVJ\# \}$) $H \implies H + \{\#F \vee G\# \} + S = F \vee G, H + S$
for $F G H S I J$ **by**(*simp-all add*: *add-mset-commute*)
case (*Cons* $g G$)
from $\langle F, \Gamma \Rightarrow_n \rangle$ **have** 1: $F, \text{disj-of-clause } g, \Gamma \Rightarrow_n$ **by** (*metis* *LSC-weaken*
add-mset-commute)
from $\langle \text{image-mset } \text{disj-of-clause} (mset (g \# G)) + \Gamma \Rightarrow_n \rangle$
have 2: *image-mset* *disj-of-clause* ($mset G$) + (*disj-of-clause* $g, \Gamma \Rightarrow_n$) **by**(*simp*
add: *add-mset-commute*)
from *Cons.IH*[*OF 1 2*] **have** IH : *disj-of-clause* $g, mset (\text{map } (\lambda d. F \vee \text{disj-of-clause}$
 $d) G) + \Gamma \Rightarrow_n$


```

  by(simp add: add-mset-commute)
  from  $\langle F, \Gamma \Rightarrow_n \rangle$  have  $\exists: F, \text{mset}(\text{map}(\lambda d. F \vee \text{disj-of-clause } d) G) + \Gamma \Rightarrow_n$ 
    using LSC-weaken-set by (metis add.assoc add.commute add-mset-add-single)
  from IH  $\exists$  show  $?case$  by auto
qed simp

```

lemma *mset-concat-map-cons*:

```

   $\text{mset}(\text{concat}(\text{map}(\lambda c. F \# G \ c) S)) = \text{mset}(\text{map } F \ S) + \text{mset}(\text{concat}(\text{map}
  G \ S))$ 

```

by(*induction S; simp add: add-mset-commute*)

lemma *distrib*:

```

   $\text{image-mset } \text{disj-of-clause}(\text{mset } F) + \Gamma \Rightarrow_n \implies$ 

```

```

   $\text{image-mset } \text{disj-of-clause}(\text{mset } G) + \Gamma \Rightarrow_n \implies$ 

```

```

   $\text{mset}[\text{disj-of-clause } c \vee \text{disj-of-clause } d. c \leftarrow F, d \leftarrow G] + \Gamma \Rightarrow_n$ 

```

proof(*induction F G arbitrary: Γ rule: two-list-induct*)

case (*Cons a F G*)

case 1

```

  from  $\langle \text{image-mset } \text{disj-of-clause}(\text{mset}(a \# F)) + \Gamma \Rightarrow_n \rangle$ 

```

have $a: \text{disj-of-clause } a, \text{image-mset } \text{disj-of-clause}(\text{mset } F) + \Gamma \Rightarrow_n$ **by**(*simp add: add-mset-commute*)

```

  from  $\langle \text{image-mset } \text{disj-of-clause}(\text{mset } G) + \Gamma \Rightarrow_n \rangle$ 

```

have $b: \text{image-mset } \text{disj-of-clause}(\text{mset } G) + (\text{image-mset } \text{disj-of-clause}(\text{mset } F) + \Gamma) \Rightarrow_n$

and $c: \text{image-mset } \text{disj-of-clause}(\text{mset } G) + (\text{mset}(\text{map}(\lambda d. \text{disj-of-clause } a \vee \text{disj-of-clause } d) G) + \Gamma) \Rightarrow_n$

using *LSC-weaken-set by (metis add.commute union-assoc)+*

from *distrib1[OF a b]*

have $\text{image-mset } \text{disj-of-clause}(\text{mset } F) + (\text{mset}(\text{map}(\lambda d. \text{disj-of-clause } a \vee \text{disj-of-clause } d) G) + \Gamma) \Rightarrow_n$

by (*simp add: union-lcomm*)

from *Cons[OF this c]*

have $\text{mset}(\text{concat}(\text{map}(\lambda c. \text{map}(\lambda d. \text{disj-of-clause } c \vee \text{disj-of-clause } d) G) F)) +$

```

   $(\text{mset}(\text{map}(\lambda d. \text{disj-of-clause } a \vee \text{disj-of-clause } d) G) + \Gamma) \Rightarrow_n .$ 

```

thus $?case$ **by**(*simp add: add.commute union-assoc*)

next

case (*Cons a F G*) **case** 2

Just the whole thing again, with slightly more mset magic and swapping things around.

```

  from  $\langle \text{image-mset } \text{disj-of-clause}(\text{mset}(a \# G)) + \Gamma \Rightarrow_n \rangle$ 

```

have $a: \text{disj-of-clause } a, \text{image-mset } \text{disj-of-clause}(\text{mset } G) + \Gamma \Rightarrow_n$ **by**(*simp add: add-mset-commute*)

```

  from  $\langle \text{image-mset } \text{disj-of-clause}(\text{mset } F) + \Gamma \Rightarrow_n \rangle$ 

```

have $b: \text{image-mset } \text{disj-of-clause}(\text{mset } F) + (\text{image-mset } \text{disj-of-clause}(\text{mset } G) + \Gamma) \Rightarrow_n$

and $c: \text{image-mset } \text{disj-of-clause}(\text{mset } F) + (\text{mset}(\text{map}(\lambda d. \text{disj-of-clause } a \vee \text{disj-of-clause } d) F) + \Gamma) \Rightarrow_n$

```

using LSC-weaken-set by (metis add commute union-assoc)+
have list-commute: (mset (map (λd. disj-of-clause a ∨ disj-of-clause d) F) + Γ)
⇒n =
(mset (map (λd. disj-of-clause d ∨ disj-of-clause a) F) + Γ) ⇒n for Γ
proof(induction F arbitrary: Γ)
case (Cons f F)
have mset (map (λd. disj-of-clause a ∨ disj-of-clause d) (f # F)) + Γ ⇒n =
disj-of-clause a ∨ disj-of-clause f, mset (map (λd. disj-of-clause a ∨ disj-of-clause
d) F) + Γ ⇒n by(simp add: add-mset-commute)
also have ... = disj-of-clause f ∨ disj-of-clause a, mset (map (λd. disj-of-clause
a ∨ disj-of-clause d) F) + Γ ⇒n
by(auto dest!: LSC-invs)
also have ... = mset (map (λd. disj-of-clause a ∨ disj-of-clause d) F) +
(disj-of-clause f ∨ disj-of-clause a, Γ) ⇒n by (simp add: add-mset-commute)
also have ... = mset (map (λd. disj-of-clause d ∨ disj-of-clause a) F) +
(disj-of-clause f ∨ disj-of-clause a, Γ) ⇒n using Cons.IH
by (metis disj-of-clause-is-nnf insert-iff is-nnf.simps(3) set-mset-add-mset-insert)

finally show ?case by simp
qed simp
from distrib1[OF a b]
have image-mset disj-of-clause (mset G) + (mset (map (λd. disj-of-clause a ∨
disj-of-clause d) F) + Γ) ⇒n
by(auto simp add: add.left-commute)
from Cons[OF c this ]
have mset (concat (map (λc. map (λd. disj-of-clause c ∨ disj-of-clause d) G)
F)) +
(mset (map (λd. disj-of-clause a ∨ disj-of-clause d) F) + Γ) ⇒n .
thus ?case using list-commute by (simp add: mset-concat-map-cons add.assoc
add.left-commute)
qed simp

lemma LSC-BigAndL: mset F + Γ ⇒n ⇒ ∙ F, Γ ⇒n
by(induction F arbitrary: Γ; simp add: LSC-weaken) (metis LSC.AndL add-mset-commute
union-mset-add-mset-right)
lemma LSC-Top-unused: [Γ ⇒n; is-nnf-mset Γ] ⇒ ∙ Γ - {#¬ ⊥#} ⇒n
proof(induction rule: LSC.induct)
case Ax thus ?case by (metis LSC.Ax add commute diff-union-swap formula.distinct(1,3)
formula.inject(2))
next
case BotL thus ?case by (metis LSC.BotL add commute diff-union-swap for-
mula.distinct(11))
next
case (AndL F G Γ)
hence (F, G, Γ) - {#¬ ⊥#} ⇒n by simp-all
hence F ∧ G, Γ - {#¬ ⊥#} ⇒n
by (metis AndL.hyps LSC.AndL diff-single-trivial diff-union-swap2)
thus ?case by (metis add commute diff-union-swap formula.distinct(19))
next

```

case (*OrL F Γ G*)
hence $(F, \Gamma) - \{\#\neg \perp\#\} \Rightarrow_n (G, \Gamma) - \{\#\neg \perp\#\} \Rightarrow_n$ **by** *simp-all*
hence $F \vee G, \Gamma - \{\#\neg \perp\#\} \Rightarrow_n$ **by** (*metis LSC.OrL OrL.hyps(1) OrL.hyps(2)*
diff-single-trivial diff-union-swap2)
thus *?case* **by** (*metis diff-union-swap formula.distinct(21)*)
qed *auto*

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

proof(*induction F arbitrary: Γ*)

case *Nil*

then show *?case* **using** *LSC-Top-unused* **by** *fastforce*

next

case (*Cons a F*)

hence $\bigwedge F, a, \Gamma \Rightarrow_n$ **by**(*auto dest: LSC-invs simp: add-mset-commute*)

with *Cons* **have** $\text{mset } F + (a, \Gamma) \Rightarrow_n$ **by** *fastforce*

then show *?case* **by** *simp*

qed

lemma *LSC-reassociate-And*: $\{\#\text{disj-of-clause } c \vee \text{disj-of-clause } d. (c, d) \in\#$
 $C\#\} + \Gamma \Rightarrow_n \implies$

is-nnf-mset $\Gamma \implies$

$\{\#\text{disj-of-clause } (c @ d). (c, d) \in\# C\#\} + \Gamma \Rightarrow_n$

proof(*induction C arbitrary: Γ*)

case (*add x C*)

obtain *a b* **where** [*simp*]: $x = (a, b)$ **by**(*cases x*)

from *add.prem*s **have** *a*: (*disj-of-clause a \vee disj-of-clause b*), $\{\#\text{disj-of-clause } c$
 $\vee \text{disj-of-clause } d. (c, d) \in\# C\#\} + \Gamma \Rightarrow_n$ **by**(*simp add: add-mset-commute*)

hence (*disj-of-clause (a@b)*), $\{\#\text{disj-of-clause } c \vee \text{disj-of-clause } d. (c, d) \in\#$
 $C\#\} + \Gamma \Rightarrow_n$ **proof** -

have *pn*: *is-nnf-mset* ($\{\#\text{disj-of-clause } c \vee \text{disj-of-clause } d. (c, d) \in\# C\#\} +$
 Γ)

using (*is-nnf-mset Γ*) **by** *auto*

have *disj-of-clause a \vee disj-of-clause b*, $\Gamma \Rightarrow_n \implies \text{is-nnf-mset } \Gamma \implies \text{disj-of-clause}$
 $(a @ b), \Gamma \Rightarrow_n$ **for** Γ

by(*induction a*) (*auto dest!: LSC-invs*)

from *this[OF - pn]* **a show** *?thesis* .

qed

hence $\{\#\text{disj-of-clause } c \vee \text{disj-of-clause } d. (c, d) \in\# C\#\} + ((\text{disj-of-clause}$
 $(a@b)), \Gamma) \Rightarrow_n$ **by**(*simp add: add-mset-commute*)

with *add.IH* **have** $\{\#\text{disj-of-clause } (c @ d). (c, d) \in\# C\#\} + (\text{disj-of-clause}$
 $(a @ b), \Gamma) \Rightarrow_n$

using (*is-nnf-mset Γ*) **by** *fastforce*

thus *?case* **by**(*simp add: add-mset-commute*)

qed *simp*

lemma *LSC-cnf*: $\Gamma \Rightarrow_n \implies \text{is-nnf-mset } \Gamma \implies \text{image-mset cnf-form-of } \Gamma \Rightarrow_n$

proof(*induction Γ rule: LSC.induct*)

have [*simp*]: *NO-MATCH (And I J) F \implies NO-MATCH ($\neg \perp$) F \implies F, $\neg \perp, \Gamma$*

```

=  $\neg\perp, F, \Gamma$  for  $F I J \Gamma$  by simp
  have [intro!]:  $\Gamma \Rightarrow_n \implies \neg\perp, \Gamma \Rightarrow_n$  for  $\Gamma$  by (simp add: LSC-weaken)
  case  $Ax$  thus ?case by(auto simp: cnf-form-of-defs)
next
  case BotL show ?case by(auto simp: cnf-form-of-defs)
next
  have GO[simp]:
    NO-MATCH ( $\{\#\wedge I\#\}$ )  $F \implies F + (\wedge G, S) = \wedge G, (F + S)$ 
    for  $F G H S I J a b$  by(simp-all add: add-mset-commute)
  case (AndL  $F G \Gamma$ ) thus ?case
  by(auto dest!: LSC-BigAndL-inv intro!: LSC-BigAndL simp add: cnf-form-of-defs)
  (simp add: add-ac)
next
  case (OrL  $F \Gamma G$ )
  have 2: image-mset disj-of-clause (mset (concat (map ( $\lambda f. \text{map } (op @ f)$ ) (cnf-lists G)) (cnf-lists F)))) +  $\Gamma \Rightarrow_n$ 
    if pig: is-nnf-mset  $\Gamma$  and a:
      mset (concat (map ( $\lambda c. \text{map } (\lambda d. \text{disj-of-clause } c \vee \text{disj-of-clause } d)$ ) (cnf-lists G)) (cnf-lists F)))) +  $\Gamma \Rightarrow_n$ 
    for  $\Gamma$ 
  proof -
    note cms[simp] = mset-map[symmetric] map-concat comp-def
    from a have image-mset ( $\lambda(c,d). \text{disj-of-clause } c \vee \text{disj-of-clause } d$ ) (
      mset (concat (map ( $\lambda c. \text{map } (\lambda d. (c,d))$ ) (cnf-lists G)) (cnf-lists F)))) +  $\Gamma$ 
     $\Rightarrow_n$  by simp
    hence image-mset ( $\lambda(c,d). \text{disj-of-clause } (c@d)$ ) (
      mset (concat (map ( $\lambda c. \text{map } (\lambda d. (c,d))$ ) (cnf-lists G)) (cnf-lists F)))) +  $\Gamma$ 
     $\Rightarrow_n$ 
    using LSC-reassociate-Ands pig by blast
  thus ?thesis by simp
qed
  have 1:  $\llbracket \wedge (\text{map } \text{disj-of-clause } (\text{cnf-lists } F)), \Gamma \Rightarrow_n; \wedge (\text{map } \text{disj-of-clause } (\text{cnf-lists } G)), \Gamma \Rightarrow_n \rrbracket$ 
     $\implies \text{is-nnf-mset } \Gamma$ 
     $\implies \wedge (\text{map } \text{disj-of-clause } (\text{concat } (\text{map } (\lambda f. \text{map } (op @ f)) (\text{cnf-lists } G)) (\text{cnf-lists } F))))), \Gamma \Rightarrow_n$ 

    for  $\Gamma$  using distrib[where 'a='a] 2 by(auto dest!: LSC-BigAndL-inv intro!: LSC-BigAndL)
  from OrL show ?case
  by(auto elim!: 1 simp add: cnf-form-of-def form-of-cnf-def)
qed auto
end

```

3.6 Converting between Resolution and SC proofs

```

theory LSC-Resolution
imports LSC Resolution

```

begin

lemma *literal-subset-sandwich*:

assumes *is-lit-plus* $F \text{ cnf } F = \{C\} \ R \subseteq C$

shows $R = \square \vee R = C$

using *assms* **by**(*cases F rule: is-lit-plus.cases; simp*) *blast+*

Proof following Gallier [3].

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

proof(*induction rule: LSC.induct*)

case (*Ax k* Γ)

let $?s = \bigcup (\text{cnf } \text{set-mset } (\neg (\text{Atom } k), \text{Atom } k, \Gamma))$

have $?s \vdash \{k^+\} \ ?s \vdash \{k^{-1}\}$ **using** *Resolution.Ass*[**where** $'a='a$] **by** *simp-all*

from *Resolution.R*[*OF this, of k*]

have $?s \vdash \square$ **by** *simp*

thus *?case* **by** *simp*

next

case (*BotL* Γ) **thus** *?case* **by**(*simp add: Ass*)

next

case (*AndL F G* Γ)

hence $\bigcup (\text{cnf } \text{set-mset } (F, G, \Gamma)) \vdash \square$ **by** *simp*

thus *?case* **by**(*simp add: Un-left-commute sup.assoc*)

next

case (*OrL F* Γ G)

hence *is-cnf* $(F \vee G)$ **by** *simp*

hence *d: is-disj* $(F \vee G)$ **by** *simp*

hence *db: is-disj F is-lit-plus F is-disj G* **by** $(-, \text{cases } F)$ *simp-all*

hence *is-cnf F* \wedge *is-cnf G* **by**(*cases F; cases G; simp*)

with *OrL* **have** *IH*: $(\bigcup (\text{cnf } \text{set-mset } (F, \Gamma))) \vdash \square \ (\bigcup (\text{cnf } \text{set-mset } (G, \Gamma))) \vdash \square$ **by** *simp-all*

let $?\Gamma = (\bigcup (\text{cnf } \text{set-mset } \Gamma))$

from *IH* **have** *IH-readable: cnf F* \cup $?\Gamma \vdash \square$ *cnf G* \cup $?\Gamma \vdash \square$ **by** *auto*

show *?case* **proof**(*cases cnf F = {}* \vee *cnf G = {}*)

case *True*

hence *cnf* $(F \vee G) = \{\}$ **by** *auto*

thus *?thesis* **using** *True IH* **by** *auto*

next

case *False*

then obtain $S \ T$ **where** *ST: cnf F = {S} cnf G = {T}*

using *cnf-disj-ex db(1,3)* **by** *metis*

hence *R: cnf* $(F \vee G) = \{S \cup T\}$ **by** *simp*

have $[S \triangleright ?\Gamma \vdash \square; T \triangleright ?\Gamma \vdash \square] \Longrightarrow S \cup T \triangleright ?\Gamma \vdash \square$ **proof** $-$

assume $s: S \triangleright ?\Gamma \vdash \square$ **and** $t: T \triangleright ?\Gamma \vdash \square$

hence *s-w: S* \triangleright $S \cup T \triangleright ?\Gamma \vdash \square$ **using** *Resolution-weaken* **by** (*metis insert-commute insert-is-Un*)

note *Resolution-taint-assumptions*[*of {T} ?\Gamma* \square S] t

then obtain R where $R: S \cup T \triangleright \bigcup(\text{cnf } ' \text{ set-mset } \Gamma) \vdash R R \subseteq S$ by (auto simp: Un-commute)
show ?thesis using literal-subset-sandwich[OF db(2) ST(1) R(2)] proof
assume $R = \square$ thus ?thesis using R(1) by blast
next
from Resolution-unnecessary[where $T=\{-\}$, simplified] R(1)
have $(R \triangleright S \cup T \triangleright ?\Gamma \vdash \square) = (S \cup T \triangleright ?\Gamma \vdash \square)$.
moreover assume $R = S$
ultimately show ?thesis using s-w by simp
qed
qed
thus ?thesis using IH ST R by simp
qed
hence case-readable: $\text{cnf } (F \vee G) \cup ?\Gamma \vdash \square$ by auto
qed auto

corollary LSC-Resolution:
assumes $\Gamma \Rightarrow_n$
shows $(\bigcup(\text{cnf } ' \text{ nnf } ' \text{ set-mset } \Gamma)) \vdash \square$
proof –
from assms
have image-mset nnf $\Gamma \Rightarrow_n$ by (simp add: LSC-NNF)
from LSC-cnf[OF this]
have image-mset (cnf-form-of \circ nnf) $\Gamma \Rightarrow_n$ by (simp add: image-mset.compositionality is-nnf-nnf)
moreover have $\forall \gamma \in \text{set-mset } (\text{image-mset } (\text{cnf-form-of } \circ \text{ nnf}) \Gamma). \text{is-cnf } \gamma$
using cnf-form-of-is[where 'a='a, OF is-nnf-nnf] by simp
moreover note CSC-Resolution-pre
ultimately have $\bigcup(\text{cnf } ' \text{ set-mset } (\text{image-mset } (\text{cnf-form-of } \circ \text{ nnf}) \Gamma)) \vdash \square$
by blast
hence $\bigcup((\lambda F. \text{cnf } (\text{cnf-form-of } (\text{nnf } F))) ' \text{ set-mset } \Gamma) \vdash \square$ by simp
thus ?thesis unfolding cnf-cnf[OF is-nnf-nnf] by simp
qed

corollary SC-Resolution:
assumes $\Gamma \Rightarrow \{\#\}$
shows $(\bigcup(\text{cnf } ' \text{ nnf } ' \text{ set-mset } \Gamma)) \vdash \square$
proof –
from assms have image-mset nnf $\Gamma \Rightarrow_n$ by (simp add: LSC-NNF SC-LSC)
hence $\bigcup(\text{cnf } ' \text{ nnf } ' \text{ set-mset } (\text{image-mset } \text{nnf } \Gamma)) \vdash \square$ using LSC-Resolution
by blast
thus ?thesis using is-nnf-nnf-id[where 'a='a] is-nnf-nnf[where 'a='a] by auto
qed

lemma Resolution-LSC-pre:
assumes $S \vdash R$
assumes finite R

```

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

using assms proof(induction S R rule: Resolution.induct)
case (Ass F S)

def  $Sm \equiv S - \{F\}$ 
hence  $Sm: S = F \triangleright Sm \ F \notin Sm$  using Ass by fast+
with Ass have  fsm: finite Sm Ball Sm finite by auto
then obtain  $Sm'$  where  $Sm = \text{set } (\text{map set } Sm')$  by (metis (full-types) ex-map-conv
finite-list)
moreover obtain  $R'$  where  $[simp]: F = \text{set } R'$  using Ass finite-list by blast
ultimately have  $S: S = \text{set } (\text{map set } (R'\#Sm'))$  unfolding  $Sm$  by simp
show ?case
  using LSC-Contract[where 'a='a]
  by(intro exI[where x=R'\#Sm'] exI[where x=R'] (simp add: S add-ac))

next
case (R S F G k)
from R.prems have fin: finite F finite G by simp-all
from R.IH(1)[OF fin(1) R.prems(2,3)] obtain  $FR FS$  where IHF:
 $\text{set } FR = F \text{ set } (\text{map set } FS) = S$ 
 $\bigwedge \Gamma \ GS. (\text{disj-of-clause } FR, \text{image-mset disj-of-clause } (\text{mset } (FS@GS))) + \Gamma$ 
 $\Rightarrow_n$ 
 $\implies \text{image-mset disj-of-clause } (\text{mset } (FS@GS)) + \Gamma \Rightarrow_n)$ 
by simp (metis add.assoc)
from R.IH(2)[OF fin(2) R.prems(2,3)] obtain  $GR GS$  where IHG:
 $\text{set } GR = G \text{ set } (\text{map set } GS) = S$ 
 $\bigwedge \Gamma \ HS. (\text{disj-of-clause } GR, \text{image-mset disj-of-clause } (\text{mset } (GS@HS))) + \Gamma$ 
 $\Rightarrow_n$ 
 $\implies \text{image-mset disj-of-clause } (\text{mset } (GS@HS)) + \Gamma \Rightarrow_n)$ 
by simp (metis add.assoc)
have IH: image-mset disj-of-clause (mset (FS @ GS)) + \Gamma \Rightarrow_n
if disj-of-clause FR, disj-of-clause GR, image-mset disj-of-clause (mset (FS @
GS)) + \Gamma \Rightarrow_n
for  $\Gamma$  using IHF(3)[of GS \Gamma] IHG(3)[of FS disj-of-clause FR, \Gamma] that
by(simp add: add-mset-commute add-ac)
show ?case
  apply(intro exI[where x=FS @ GS] exI[where x=removeAll (k+) FR @
removeAll (k-1) GR] allI impI conjI)
  apply(simp add: IHF IHG;fail)
  apply(insert IHF IHG; simp;fail)
  apply(intro IH)
  apply(auto dest!: LSC-Sim-resolution-la simp add: IHF IHG R.hyps)
done
qed

```

lemma *Resolution-LSC-pre-nodisj*:

assumes $S \vdash R$
assumes *finite R*
assumes *finite S Ball S finite*
shows $\exists S' R'. \forall \Gamma. \text{is-nnf-mset } \Gamma \longrightarrow \text{is-disj } R' \wedge \text{is-nnf } S' \wedge \text{cnf } R' = \{R\} \wedge$
 $\text{cnf } S' \subseteq S \wedge$
 $(R', S', \Gamma \Rightarrow_n \longrightarrow S', \Gamma \Rightarrow_n)$
proof –
have *mehorder*: $F, \bigwedge G, \Gamma = \bigwedge G, F, \Gamma$ **for** $F G \Gamma$ **by** (*simp add: add-ac*)
from *Resolution-LSC-pre* [**where** $'a = 'a, OF \text{ assms}$]
obtain $S' R'$ **where** $o: \bigwedge \Gamma. \text{is-nnf-mset } \Gamma \Longrightarrow \text{set } R' = R \wedge \text{set } (\text{map set } S') =$
 $S \wedge$
 $(\text{disj-of-clause } R', \text{image-mset disj-of-clause } (\text{mset } S') + \Gamma \Rightarrow_n \longrightarrow \text{image-mset}$
 $\text{disj-of-clause } (\text{mset } S') + \Gamma \Rightarrow_n)$
by *blast*
hence $p: \text{is-nnf-mset } \Gamma \Longrightarrow (\text{disj-of-clause } R', \text{image-mset disj-of-clause } (\text{mset}$
 $S') + \Gamma \Rightarrow_n \Longrightarrow \text{image-mset disj-of-clause } (\text{mset } S') + \Gamma \Rightarrow_n)$
for Γ **by** *blast*
show *?thesis*
apply (*rule exI* [**where** $x = \bigwedge \text{map disj-of-clause } S'$])
apply (*rule exI* [**where** $x = \text{disj-of-clause } R'$])
apply *safe*
apply (*intro disj-of-clause-is; fail*)
apply (*simp add: cnf-disj o; fail*) +
subgoal using o **by** (*fastforce simp add: cnf-BigAnd cnf-disj*)
subgoal for Γ
apply (*frule p*)
apply (*unfold mehorder*)
apply (*drule LSC-BigAndL-inv*)
apply (*simp; fail*) +
by (*simp add: LSC-BigAndL*)
done
qed

corollary *Resolution-LSC1*:

assumes $S \vdash \square$
shows $\exists F. \text{is-nnf } F \wedge \text{cnf } F \subseteq S \wedge \{\#F\# \} \Rightarrow_n$
proof –
have $*$: $\{f \cup g \mid f g. f \in F \wedge g \in G\} = \{\square\} \Longrightarrow F = \{\square\}$ **for** $F G$
proof (*rule ccontr*)
assume $m: \{f \cup g \mid f g. f \in F \wedge g \in G\} = \{\square\}$
assume $F \neq \{\square\}$
hence $F = \{\}$ $\vee (\exists E. E \in F \wedge E \neq \square)$ **by** *blast*
thus *False* **proof**
assume $F = \{\}$
with m **show** *False* **by** *simp*
next
assume $\exists E. E \in F \wedge E \neq \square$ **then** *guess E .. note E = this*
show *False* **proof** *cases*


```

    assume  $G = \{\}$  with  $m$  show False by simp
next
  assume  $G \neq \{\}$ 
  then obtain  $D$  where  $D \in G$  by blast
  with  $E$  have  $E \cup D \in \{f \cup g \mid f \in F \wedge g \in G\}$  by blast
  with  $m E$  show False by simp
qed
qed
qed
have *:  $F = \{\square\} \wedge G = \{\square\}$  if  $\{f \cup g \mid f \in F \wedge g \in G\} = \{\square\}$  for  $F G$ 
proof (intro conjI)
  show  $G = \{\square\}$ 
  apply (rule *[of G F])
  apply (subst that[symmetric])
  by blast
qed (rule *[OF that])
have *:  $is\text{-nnf } F \implies is\text{-nnf}\text{-mset } \Gamma \implies cnf F = \{\square\} \implies F, \Gamma \Rightarrow_n$  for  $F \Gamma$ 
  apply (induction F rule: cnf.induct; simp)
  apply blast
  apply (metis LSC.LSC.AndL LSC-weaken add-mset-commute singleton-Un-iff)
  apply (drule *; simp add: LSC.LSC.OrL)
done
from Resolution-useless-infinite[OF assms]
obtain  $S'$  where  $su: S' \subseteq S$  and  $fin: finite\ S' \wedge Ball\ S' \text{ finite}$  and  $pr: (S' \vdash \square)$ 
by blast
from Resolution-LSC-pre-nodisj[OF pr finite.emptyI fin]
obtain  $S''$  where  $is\text{-nnf } S'' \wedge cnf\ S'' \subseteq S' \wedge \{\# S'' \#\} \Rightarrow_n$ 
  using *[OF disj-is-nnf, of - Mempty]
  by (metis LSC-weaken add-mset-commute empty-iff set-mset-empty)
with  $su$  show ?thesis by blast
qed

corollary Resolution-SC1:
  assumes  $S \vdash \square$ 
  shows  $\exists F. cnf (nnf F) \subseteq S \wedge \{\# F \#\} \Rightarrow \{\#\}$ 
  apply (insert Resolution-LSC1[OF assms])
  apply (elim ex-forward)
  apply (elim conjE; intro conjI)
  subgoal by (subst is-nnf-nnf-id; assumption)
  apply (unfold SC-LSC)
  subgoal by (simp; fail)
done

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(induction rule: ND.induct)
  case (Ax F  $\Gamma$ ) thus ?case by(intro exI[of - {F}]) (simp add: ND.Ax)
next
  case (AndI  $\Gamma$   $F$   $G$ )
  from AndI.IH(1) guess  $\Gamma 1$  .. moreover
  from AndI.IH(2) guess  $\Gamma 2$  .. ultimately
  show ?case by(intro exI[where x= $\Gamma 1 \cup \Gamma 2$ ]) (force elim: Weaken intro!: ND.AndI)
next
  case (CC F  $\Gamma$ )
  from CC.IH guess  $\Gamma'$  .. note  $\Gamma' = \text{this}$ 
  thus ?case proof(cases Not F  $\in \Gamma'$ )

```

case distinction: Did we actually use $\neg F$?

```

  case False hence  $\Gamma' \subseteq \Gamma$  using  $\Gamma'$  by blast
  with  $\Gamma'$  show ?thesis using BotE by(intro exI[where x= $\Gamma'$ ]) fast
next
  case True
  then obtain  $\Gamma''$  where  $\Gamma' = \neg F \triangleright \Gamma'' \neg F \notin \Gamma''$  by (meson Set.set-insert)
  hence  $\Gamma'' \subseteq \Gamma$  finite  $\Gamma'' \neg F \triangleright \Gamma'' \vdash \perp$  using  $\Gamma'$  by auto
  thus ?thesis using ND.CC by auto
qed
next
  case AndE1 thus ?case by(blast dest: ND.AndE1) next
  case AndE2 thus ?case by(blast dest: ND.AndE2)
next
  case OrI1 thus ?case by(blast dest: ND.OrI1) next
  case OrI2 thus ?case by(blast dest: ND.OrI2)
next
  case (OrE  $\Gamma$   $F$   $G$   $H$ )
  from OrE.IH(1) guess  $\Gamma 1$  .. moreover
  from OrE.IH(2) guess  $\Gamma 2$  .. moreover
  from OrE.IH(3) guess  $\Gamma 3$  ..
  note IH = calculation this
  let ?w =  $\Gamma 1 \cup (\Gamma 2 - \{F\}) \cup (\Gamma 3 - \{G\})$ 
  from IH have ?w  $\vdash F \vee G$  using Weaken[OF - sup-ge1] by metis moreover
  from IH have  $F \triangleright ?w \vdash H$   $G \triangleright ?w \vdash H$  using Weaken by (metis Un-commute
  Un-insert-right Un-upper1 Weaken insert-Diff-single) ultimately
  have ?w  $\vdash H$  using ND.OrE by blast
  thus ?case using IH by(intro exI[where x=?w]) auto

```

Clever evasion of the case distinction made for CC.

```

next
  case (ImpI F  $\Gamma$   $G$ )
  from ImpI.IH guess  $\Gamma'$  ..
  thus ?case by (intro exI[where x= $\Gamma' - \{F\}$ ]) (force elim: Weaken intro!:
  ND.ImpI)
next

```

```

    case (ImpE  $\Gamma$  F G)
    from ImpE.IH(1) guess  $\Gamma 1$  .. moreover
    from ImpE.IH(2) guess  $\Gamma 2$  .. ultimately
    show ?case by(intro exI[where  $x=\Gamma 1 \cup \Gamma 2$ ]) (force elim: Weaken intro: ND.ImpE[where
F=F])
next
    case (NotE  $\Gamma$  F)
    from NotE.IH(1) guess  $\Gamma 1$  .. moreover
    from NotE.IH(2) guess  $\Gamma 2$  .. ultimately
    show ?case by(intro exI[where  $x=\Gamma 1 \cup \Gamma 2$ ]) (force elim: Weaken intro: ND.NotE[where
F=F])
next
    case (NotI F  $\Gamma$ )
    from NotI.IH guess  $\Gamma'$  ..
    thus ?case by(intro exI[where  $x=\Gamma' - \{F\}$ ]) (force elim: Weaken intro: ND.NotI[where
F=F])
qed

```

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

corollary *ND-no-assms*:

```

    assumes  $\Gamma \vdash F$ 
    obtains  $\Gamma'$  where set  $\Gamma' \subseteq \Gamma \wedge (\{\} \vdash \bigwedge \Gamma' \rightarrow F)$ 
proof(goal-cases)
    case 1
    from ND-finite-assms[OF assms] obtain  $\Gamma'$  where  $\Gamma' \subseteq \Gamma$  finite  $\Gamma' \vdash F$  by
blast
    from ⟨finite  $\Gamma'$ ⟩ obtain G where  $\Gamma'$ [simp]:  $\Gamma' = \text{set } G$  using finite-list by blast
    with ⟨ $\Gamma' \subseteq \Gamma$ ⟩ have set  $G \subseteq \Gamma$  by clarify
    moreover from ⟨ $\Gamma' \vdash F$ ⟩ have  $\{\} \vdash \bigwedge G \rightarrow F$  unfolding  $\Gamma'$  AssmBigAnd .
    ultimately show ?case by(intro 1[where  $\Gamma'=G$ ] conjI)
qed
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  $\Gamma \vdash F \longleftrightarrow \Gamma \Vdash F$  (is ?n  $\longleftrightarrow$  ?s)

```

```

proof
  assume ?n thus ?s by (fact ND-sound)
next
  assume s: ?s
  with compact-entailment obtain  $\Gamma'$  where 0: finite  $\Gamma' \Gamma' \models F \Gamma' \subseteq \Gamma$ 
    unfolding entailment-def by metis
  then obtain  $\Gamma''$  where  $\Gamma'': \Gamma' = \text{set-mset } \Gamma''$  using finite-set-mset-mset-set by
    blast
  have su: set-mset  $\Gamma'' \subseteq \Gamma$  using 0  $\Gamma''$  by fast
  from 0 have  $\models \Gamma'' \Rightarrow \{\#F\# \}$  unfolding sequent-semantics-def entailment-def
 $\Gamma''$  by simp
  with SC-sound-complete have  $\Gamma'' \Rightarrow \{\#F\# \}$  by blast
  with SCND have set-mset  $\Gamma'' \cup \neg ' \text{ set-mset } \{\#F\# \} \vdash \perp .$ 
  thus ?n using ND.CC Weaken[OF - su[THEN insert-mono]] by force
qed

```

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

```

lemma ND-sound-complete-finite:
  assumes finite  $\Gamma$ 
  shows  $\Gamma \vdash F \longleftrightarrow \Gamma \models F$  (is ?n  $\longleftrightarrow$  ?s)
proof
  assume ?n thus ?s by (fact ND-sound)
next
  assume s: ?s
  then obtain  $\Gamma''$  where  $\Gamma'': \Gamma = \text{set-mset } \Gamma''$  using finite-set-mset-mset-set
    assms by blast
  have su: set-mset  $\Gamma'' \subseteq \Gamma$  using  $\Gamma''$  by fast
  have  $\models \Gamma'' \Rightarrow \{\#F\# \}$  using s unfolding sequent-semantics-def entailment-def
 $\Gamma''$  by auto
  with SC-sound-complete have  $\Gamma'' \Rightarrow \{\#F\# \}$  by blast
  with SCND have set-mset  $\Gamma'' \cup \neg ' \text{ set-mset } \{\#F\# \} \vdash \perp .$ 
  thus ?n using ND.CC Weaken[OF - su[THEN insert-mono]] by force
qed

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 -
  from fin obtain  $S'$  where  $S: S = \text{set-mset } S'$  using finite-set-mset-mset-set by
    blast
  have cnf:  $\forall F \in \text{set-mset } (\text{image-mset } (\text{cnf-form-of} \circ \text{nnf}) (\neg F, S')). \text{ is-cnf } F$ 

```

by (*simp add: cnf-form-of-is is-nnf-nnf*)
note *entailment-def[simp]*
from *val*
have $S \models \neg(\neg F)$ **by** *simp*
hence $S \models \neg(\text{nnf } (\neg F))$ **by** (*simp add: nnf-semantic*)
hence $S \models \neg(\text{cnf-form-of } (\text{nnf } (\neg F)))$ **by** (*simp add: cnf-form-semantic[OF is-nnf-nnf]*)
hence $\text{set-mset } (\text{image-mset } \text{nnf } S') \models \neg(\text{cnf-form-of } (\text{nnf } (\neg F)))$ **using** S **by** (*simp add: nnf-semantic*)
hence $\text{set-mset } (\text{image-mset } (\text{cnf-form-of } \circ \text{nnf}) S') \models \neg(\text{cnf-form-of } (\text{nnf } (\neg F)))$ **by** (*simp add: cnf-form-semantic[OF is-nnf-nnf]*)
hence $\text{image-mset } (\text{cnf-form-of } \circ \text{nnf}) S' \Rightarrow \{\#\neg(\text{cnf-form-of } (\text{nnf } (\neg F)))\#\}$
unfolding *SC-sound-complete sequent-intuitionistic-semantic* .
hence $\text{image-mset } (\text{cnf-form-of } \circ \text{nnf}) (\neg F, S') \Rightarrow \{\#\}$ **using** *NotR-inv* **by** *simp*
hence $\text{image-mset } (\text{cnf-form-of } \circ \text{nnf}) (\neg F, S') \Rightarrow_n$ **by** (*simp add: SC-LSC is-nnf-nnf nnf-cnf-form*)
with *CSC-Resolution-pre* **have** $\bigcup(\text{cnf } \text{' set-mset } (\text{image-mset } (\text{cnf-form-of } \circ \text{nnf}) (\neg F, S')) \vdash \square)$ **using** *cnf* .
thus *?thesis* **using** *cnf-cnf[where 'a='a, OF is-nnf-nnf]*
unfolding *set-image-mset image-comp comp-def S* **by** *simp*
qed

corollary *Resolution-complete-single:*

assumes $\models F$
shows $\text{cnf } (\text{nnf } (\neg F)) \vdash \square$
using *assms Resolution-complete'[OF finite.emptyI, of F]*
unfolding *entailment-def comp-def* **by** *simp*

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 $\bigcup((\text{cnf } \circ \text{nnf}) \text{' } (\{\neg F\} \cup S)) \vdash \square$

proof –

let $?mun = \lambda s. \bigcup((\text{cnf } \circ \text{nnf}) \text{' } s)$

from *compact-entailment[OF val]* **obtain** S'' **where** $\text{fin}: \text{finite } S''$ **and** $\text{su}: S'' \subseteq S$ **and** $\text{val}': S'' \models F$ **by** *blast*

from fin **obtain** S' **where** $S: S'' = \text{set-mset } S'$ **using** *finite-set-mset-mset-set* **by** *blast*

have $\text{cnf}: \forall F \in \text{set-mset } (\text{image-mset } (\text{cnf-form-of } \circ \text{nnf}) (\neg F, S')). \text{is-cnf } F$ **by** (*simp add: cnf-form-of-is is-nnf-nnf*)

```

note entailment-def[simp]
from val' have  $S'' \models \neg(\neg F)$  by simp
hence  $S' \Rightarrow \{\#\neg(\neg F)\#\}$ 
  unfolding SC-sound-complete sequent-intuitionistic-semantics S .
hence  $\neg F, S' \Rightarrow \{\#\}$  by (simp add: NotR-inv)
hence image-mset nnf  $(\neg F, S') \Rightarrow \{\#\}$  using LSC-NNF SC-LSC by blast
hence image-mset nnf  $(\neg F, S') \Rightarrow_n$  by (simp add: SC-LSC is-nnf-nnf)
with LSC-Resolution have  $\bigcup(\text{cnf ' nnf ' set-mset (image-mset nnf } (\neg F, S')))$ 
 $\vdash \square$  .
hence ?mun  $(\{\neg F\} \cup S'') \vdash \square$ 
  unfolding set-image-mset image-comp comp-def S is-nnf-nnf-id[OF is-nnf-nnf]
by simp
from Resolution-weaken[OF this, of ?mun S] show ?thesis using su by (metis
UN-Un Un-left-commute sup.order-iff)
qed

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: is-mini-mset (image-mset to-mini-formula Γ) **by** simp

lemma SC-full-to-mini:

defines tms \equiv image-mset to-mini-formula

assumes op: $\Gamma \Rightarrow \Delta$

shows tms $\Gamma \Rightarrow$ tms Δ

proof –

have tmsi[simp]: tms $(F, S) =$ to-mini-formula F , tms S **for** $F S$ **unfolding**
tms-def **by** simp

from op **show** ?thesis

proof(induction $\Gamma \Delta$ rule: SCp.induct)

case (BotL Γ)

hence $\perp \in\#$ tms Γ **unfolding** tms-def **by** force

thus ?case **using** SCp.BotL **by** blast

next

case (Ax $k \Gamma \Delta$)

hence Atom $k \in\#$ tms Γ Atom $k \in\#$ tms Δ **unfolding** tms-def **using** image-iff
by fastforce+

thus ?case **using** SCp.Ax[of k] **by** blast

next

case (NotR $F \Gamma \Delta$) **thus** ?case

unfolding tmsi to-mini-formula.simps

using weakenR SCp.ImpR **by** blast

next

```

    case (NotL  $\Gamma$   $F$   $\Delta$ ) from this(2) show ?case
      by(auto intro!: SCp.ImpL)
  next
    case ImpR thus ?case using SCp.ImpR by simp
  next
    case ImpL thus ?case using SCp.ImpL by simp
  next
    case AndR from AndR(3,4) show ?case
      using weakenR by(auto intro!: SCp.ImpR SCp.ImpL)
  next
    case AndL from AndL(2) show ?case
      using weakenR[where 'a='a] by(fastforce intro!: SCp.ImpR SCp.ImpL)
  next
    case OrR from OrR(2) show ?case
      using weakenR by(fastforce intro!: SCp.ImpR SCp.ImpL)
  next
    case (OrL  $F$   $\Gamma$   $\Delta$   $G$ )
    note SCp.ImpL
    moreover {
      have to-mini-formula  $F$ ,  $tms \Gamma \Rightarrow tms \Delta$  using OrL(3)[unfolded tmsi] .
      with weakenR have to-mini-formula  $F$ ,  $tms \Gamma \Rightarrow \perp$ ,  $tms \Delta$  by blast
      with SCp.ImpR have  $tms \Gamma \Rightarrow$  to-mini-formula  $F \rightarrow \perp$ ,  $tms \Delta$  . }
    moreover have to-mini-formula  $G$ ,  $tms \Gamma \Rightarrow tms \Delta$  using (tms ( $G$ ,  $\Gamma$ )  $\Rightarrow$ 
    tms  $\Delta$ ) unfolding tmsi .
    ultimately have (to-mini-formula  $F \rightarrow \perp$ )  $\rightarrow$  to-mini-formula  $G$ ,  $tms \Gamma \Rightarrow$ 
    tms  $\Delta$  .
    thus ?case unfolding tmsi to-mini-formula.simps .
  qed
qed

```

lemma *SC-mini-to-full*:

```

  defines tms  $\equiv$  image-mset to-mini-formula
  assumes op:  $tms \Gamma \Rightarrow tms \Delta$ 
  shows  $\Gamma \Rightarrow \Delta$ 
  proof -
    have tmsi[simp]:  $tms (F,S) =$  to-mini-formula  $F$ ,  $tms S$  for  $F S$  unfolding
    tms-def by simp
    note Impl-inv ImpR-inv[dest]
    have no:  $f \neq (\lambda F G. Not F) f \neq Or f \neq And$  if  $f F G$ ,  $S' = tms S$  for  $f F G$ 
     $S S'$ 
    by (metis that is-mini-formula.simps(4-6) msed-map-invR tms-def to-mini-is-mini
    union-commute)+
    note dr = no(1)[where f= $\lambda F G. Not F$ , simplified, dest!]
      no(2)[where f= $Or$ , simplified, dest!]
      no(3)[where f= $And$ , simplified, dest!]
    have whai:
      ( $\exists S2 H J. S = H \rightarrow J$ ,  $S2 \wedge F =$  to-mini-formula  $H \wedge G =$  to-mini-formula
       $J$ )  $\vee$ 
      ( $\exists S2 H J. S = H \vee J$ ,  $S2 \wedge F =$  (to-mini-formula  $H \rightarrow \perp$ )  $\wedge G =$ 

```

```

to-mini-formula J)  $\vee$ 
  ( $\exists S2 H J. S = H \wedge J, S2 \wedge F =$  to-mini-formula  $H \rightarrow$  to-mini-formula  $J \rightarrow$ 
 $\perp \wedge G = \perp$ )  $\vee$ 
  ( $\exists S2 H. S = \neg H, S2 \wedge F =$  to-mini-formula  $H \wedge G = \perp$ )
if  $F \rightarrow G, S1 =$  tms  $S$  for  $F G S1 S$  proof -
  note that[unfolded tms-def]
  then obtain  $S2$  pim where  $S2: S1 =$  image-mset to-mini-formula  $S2$ 
    and  $S: S =$  pim,  $S2$ 
    and pim:  $F \rightarrow G =$  to-mini-formula pim
  by (metis msed-map-invR union-commute)
  show ?thesis using pim unfolding  $S$  by(cases pim; simp; blast)
qed
from op show ?thesis
proof(induction tms  $\Gamma$  tms  $\Delta$  arbitrary:  $\Gamma \Delta$  rule: SCp.induct)
  have *: to-mini-formula  $F = \perp \implies F = \perp$  for  $F$  by(cases  $F$ ; simp)
  case BotL thus ?case unfolding tms-def using * SCp.BotL by (metis image-iff
multiset.set-map)
next
  have *: Atom  $k =$  to-mini-formula  $F \implies F =$  Atom  $k$  for  $F k$  by(cases  $F$ ;
simp)
  case (Ax -  $k$ ) thus ?case
    unfolding tms-def unfolding in-image-mset Set.image-iff
    apply(elim bexE)
    apply(drule *)+
    apply(intro SCp.Ax)
  by simp-all
next
case (ImpL  $\Gamma' F G$ )
  note whai[OF ImpL(5)]
  thus ?case proof(elim disjE exE conjE)
    fix  $S2 H J$ 
    assume *:  $\Gamma = H \rightarrow J, S2 F =$  to-mini-formula  $H G =$  to-mini-formula  $J$ 
    hence  $\Gamma' =$  tms  $S2 F, tms \Delta =$  tms  $(H, \Delta) G, \Gamma' =$  tms  $(J, S2)$  tms  $\Delta =$ 
tms  $\Delta$  using ImpL.hyps(5) add-left-imp-eq by auto
    from ImpL(2)[OF this(1,2)] ImpL(4)[OF this(3-)]
    show ?thesis using SCp.ImpL by(simp add: *)
  next
    fix  $S2 H J$ 
    assume *:  $\Gamma = H \vee J, S2 F =$  to-mini-formula  $H \rightarrow \perp G =$  to-mini-formula
 $J$ 
    hence  $\Gamma' =$  tms  $S2 F, tms \Delta =$  tms  $(H \rightarrow \perp, \Delta) G, \Gamma' =$  tms  $(J, S2)$  tms
 $\Delta =$  tms  $\Delta$  using ImpL.hyps(5) add-left-imp-eq by auto
    from ImpL(2)[OF this(1,2)] ImpL(4)[OF this(3-)]
    show ?thesis using Bot-delR by(force intro!: SCp.OrL dest!: ImpR-inv simp:
*)
  next
    fix  $S2 H J$ 
    assume *:  $\Gamma = H \wedge J, S2 F =$  to-mini-formula  $H \rightarrow$  to-mini-formula  $J \rightarrow$ 
 $\perp G = \perp$ 

```



```

hence  $\Gamma' = \text{tms } S2 F, \text{tms } \Delta = \text{tms } (H \rightarrow J \rightarrow \perp, \Delta) G, \Gamma' = \text{tms } (\perp, S2)$ 
 $\text{tms } \Delta = \text{tms } \Delta$  using ImpL.hyps(5) add-left-imp-eq by auto
from ImpL(2)[OF this(1,2)]
show ?thesis using Bot-delR SCp.AndL ImpR-inv * by (metis add-mset-remove-trivial
inL1)
next
fix S2 H
assume  $*$ :  $\Gamma = \neg H, S2 F = \text{to-mini-formula } H G = \perp$ 
hence  $\Gamma' = \text{tms } S2 F, \text{tms } \Delta = \text{tms } (H, \Delta) G, \Gamma' = \text{tms } (\perp, S2) \text{tms } \Delta =$ 
 $\text{tms } \Delta$  using ImpL.hyps(5) add-left-imp-eq by auto
from ImpL(2)[OF this(1,2)]
show ?thesis by(force intro!: SCp.NotL dest!: ImpR-inv simp: *)
qed
next
case (ImpR F G  $\Delta'$ )
note whai[OF ImpR(3)]
thus ?case proof(elim disjE exE conjE)
fix S2 H J
assume  $\Delta = H \rightarrow J, S2 F = \text{to-mini-formula } H G = \text{to-mini-formula } J$ 
thus ?thesis using ImpR.hyps(2,3) by (auto intro!: SCp.ImpR)
next
fix S2 H J
assume  $*$ :  $\Delta = H \vee J, S2 F = \text{to-mini-formula } H \rightarrow \perp G = \text{to-mini-formula}$ 
 $J$ 
hence  $\Delta' = \text{tms } S2 F, \text{tms } \Delta = \text{tms } (H \rightarrow \perp, \Delta) G, \Delta' = \text{tms } (J, S2) \text{tms}$ 
 $\Delta = \text{tms } \Delta$  using ImpR.hyps(3) add-left-imp-eq by auto
thus ?thesis using SCp.OrR[where 'a='a] * ImpR.hyps(2) by (simp add:
NotL-inv)
next
have botoff:  $\Gamma \Rightarrow H, \perp, S2 \Longrightarrow \Gamma \Rightarrow H, S2$  for  $\Gamma H S2$  using Bot-delR by
fastforce
fix S2 H J
assume  $*$ :  $\Delta = H \wedge J, S2 F = \text{to-mini-formula } H \rightarrow \text{to-mini-formula } J \rightarrow$ 
 $\perp G = \perp$ 
hence  $F, \text{tms } \Gamma = \text{tms } (H \rightarrow J \rightarrow \perp, \Gamma) G, \Delta' = \text{tms } (\perp, S2)$ 
using ImpR.hyps(3) by(auto)
from ImpR(2)[OF this] show ?thesis by(auto simp add: * intro!: SCp.intros(3-)
dest!: ImpL-inv botoff)
next

fix S2 H
assume  $\Delta = \neg H, S2 F = \text{to-mini-formula } H G = \perp$ 
then obtain S2 H where  $*$ :  $\Delta = \neg H, S2 F = \text{to-mini-formula } H \wedge G =$ 
 $\perp$  by blast
hence  $F, \text{tms } \Gamma = \text{tms } (H, \Gamma) G, \Delta' = \text{tms } (\perp, S2)$  using ImpR(3) by
simp-all
with ImpR(2) have  $H, \Gamma \Rightarrow \perp, S2$  .
hence  $\Gamma \Rightarrow \neg H, S2$  using SCp.NotR Bot-delR by fastforce
thus  $\Gamma \Rightarrow \Delta$  by(simp add: *)

```

qed
 qed auto
 qed

theorem *MiniSC-eq: image-mset to-mini-formula* $\Gamma \Rightarrow \text{image-mset to-mini-formula}$
 $\Delta \longleftrightarrow \Gamma \Rightarrow \Delta$
 using *SC-mini-to-full SC-full-to-mini* by blast

end

3.8.1 SC to HC

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

inductive-set *AX1* **where**

$F \in \text{AX0} \Longrightarrow F \in \text{AX1} \mid$
 $((F \rightarrow \perp) \rightarrow \perp) \rightarrow F \in \text{AX1}$

lemma *AX01: AX0 \subseteq AX1* by (*simp add: AX1.intros(1) subsetI*)

lemma *AX1-away: AX1 $\cup \Gamma = \text{AX0} \cup (\Gamma \cup \text{AX1})$* using *AX01* by blast

lemma *Deduction1: F $\triangleright (\text{AX1} \cup \Gamma) \vdash_H \perp \longleftrightarrow (\text{AX1} \cup \Gamma) \vdash_H (F \rightarrow \perp)$* **unfolding** *AX1-away* by (*metis Deduction-theorem HC.simps HC-mono Un-commute Un-insert-left insertI1 subset-insertI*)

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

proof

assume *l*: ?l

with *Deduction-theorem*[**where** $\Gamma = \text{AX1} \cup \Gamma$ **and** $F = F \rightarrow \perp$ **and** $G = \perp$]

have $\text{AX1} \cup \Gamma \vdash_H (F \rightarrow \perp) \rightarrow \perp$ **unfolding** *AX1-away* by (*simp add: Un-commute*)

moreover have $\text{AX1} \cup \Gamma \vdash_H ((F \rightarrow \perp) \rightarrow \perp) \rightarrow F$ using *AX1.intros(2)*

HC.Ax by blast

ultimately show ?r using *HC.MP* by blast

next

assume *r*: ?r **thus** ?l by (*meson HC.simps HC-mono insertI1 subset-insertI*)

qed

lemma

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

proof(*induction* $\Gamma \Delta$ *rule: SCp.induct*)

case (*ImpL* $\Gamma F \Delta G$)

from *ImpL.prem*s **have** *is-mini-mset* Γ *is-mini-mset* (F, Δ) by *simp-all*

with *ImpL.IH(1)* **have** *IH1: set-mset* $\Gamma \cup (\lambda F. F \rightarrow \perp)$ ' *set-mset* $(F, \Delta) \cup \text{AX1} \vdash_H \perp$.

hence *IH1: F $\rightarrow \perp \triangleright \text{set-mset } \Gamma \cup (\lambda F. F \rightarrow \perp)$ ' set-mset $\Delta \cup \text{AX1} \vdash_H \perp$
 by *simp**

from *ImpL.prem*s **have** *is-mini-mset* (G, Γ) *is-mini-mset* Δ by *simp-all*

with *ImpL.IH(2)* **have** *IH2: set-mset (G, Γ) \cup ($\lambda F. F \rightarrow \perp$) ‘ set-mset $\Delta \cup AX1 \vdash_H \perp$.*
hence *IH2: G \triangleright (set-mset $\Gamma \cup$ ($\lambda F. F \rightarrow \perp$) ‘ set-mset $\Delta \cup AX1) \vdash_H \perp$ **by** *simp**
have *R: F \rightarrow G \triangleright AX1 \cup $\Gamma \vdash_H \perp$ **if** *G \triangleright AX1 \cup $\Gamma \vdash_H \perp$ F $\rightarrow \perp \triangleright$ AX1 \cup $\Gamma \vdash_H \perp$ **for** Γ*
using that by (*metis Ax Deduction1 Deduction2 HC-mono MP insertI1 subset-insertI*)

from *R[where $\Gamma =$ set-mset $\Gamma \cup$ ($\lambda F. F \rightarrow \perp$) ‘ set-mset Δ]*
have *F \rightarrow G \triangleright (set-mset $\Gamma \cup$ ($\lambda F. F \rightarrow \perp$) ‘ set-mset $\Delta \cup AX1) \vdash_H \perp$ **using** *IH2 IH1 by(simp add: Un-commute)**
thus *?case by simp*
next
case (*ImpR F Γ G Δ*)
hence *is-mini-mset (F, Γ) is-mini-mset (G, Δ) **by** *simp-all**
with *ImpR.IH **have** *IH: F \triangleright G $\rightarrow \perp \triangleright$ set-mset $\Gamma \cup$ ($\lambda F. F \rightarrow \perp$) ‘ set-mset $\Delta \cup AX1 \vdash_H \perp$ **by**(simp add: insert-commute)**
have *R: (F \rightarrow G) $\rightarrow \perp \triangleright$ $\Gamma \cup$ AX1 $\vdash_H \perp$ **if** *F \triangleright G $\rightarrow \perp \triangleright$ $\Gamma \cup$ AX1 $\vdash_H \perp$ **for** Γ **using that***
by (*metis AX1-away Deduction2 Deduction-theorem Un-commute Un-insert-right*)
thus *?case using IH by simp*
next
case (*Ax k Γ Δ*)
have *R: F \triangleright F $\rightarrow \perp \triangleright$ $\Gamma \cup$ AX1 $\vdash_H \perp$ **for** *F :: 'a formula **and** Γ **by** (meson HC.simps insert-iff)*
from *R[where F=Atom k **and** $\Gamma =$ set-mset $\Gamma \cup$ ($\lambda F. F \rightarrow \perp$) ‘ set-mset Δ]*
show *?case using Ax.hyps*
by (*simp add: insert-absorb*)
next
case *BotL **thus** ?case by (simp add: HC.Ax)*
qed *simp-all*

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 Θ . atoms F*

lemma *interpolation-equal-styles:*

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

\longleftrightarrow

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

proof(*intro iffI allI impI, goal-cases*)
case (1 $\Gamma \Delta$)
hence $\Gamma + \{\#\} \Rightarrow \{\#\} + \Delta \longrightarrow (\exists F. \Gamma \Rightarrow F, \{\#\} \wedge F, \{\#\} \Rightarrow \Delta \wedge \text{atoms } F \subseteq \text{atoms-mset } (\Gamma + \{\#\}) \wedge \text{atoms } F \subseteq \text{atoms-mset } (\{\#\} + \Delta))$ **by** *presburger*
with 1 **show** *?case by auto*
next
case (2 $\Gamma \Delta \Gamma' \Delta'$)
from 2(2) **have** $\Gamma \Rightarrow \Delta + \text{image-mset Not } \Gamma' + \Delta'$ **by**(*induction* Γ' *arbitrary:* Γ ; *simp add:* *SCp.NotR*)
hence $\Gamma + \text{image-mset Not } \Delta \Rightarrow \text{image-mset Not } \Gamma' + \Delta'$ **by**(*induction* Δ *arbitrary:* Δ' ; *simp add:* *SCp.NotL*) (*metis* *SCp.NotL union-mset-add-mset-right*)
from 2(1)[*THEN spec, THEN spec, THEN mp, OF this*]
have $\exists F. \Gamma + \text{image-mset } \neg \Delta \Rightarrow \{\#F\# \} \wedge \{\#F\# \} \Rightarrow \text{image-mset } \neg \Gamma' + \Delta' \wedge \text{atoms } F \subseteq \text{atoms-mset } (\Gamma + \text{image-mset } \neg \Delta) \wedge \text{atoms } F \subseteq \text{atoms-mset } (\text{image-mset } \neg \Gamma' + \Delta')$.
then obtain F **where** $n: \Gamma + \text{image-mset } \neg \Delta \Rightarrow \{\#F\# \}$ **and** $p: \{\#F\# \} \Rightarrow \text{image-mset } \neg \Gamma' + \Delta'$ **and** $at: \text{atoms } F \subseteq \text{atoms-mset } (\Gamma + \Delta) \text{ atoms } F \subseteq \text{atoms-mset } (\Gamma' + \Delta')$ **by** *auto*
from n **have** $n: \Gamma \Rightarrow F, \Delta$ **by**(*induction* Δ *arbitrary:* Γ ; *simp add:* *NotL-inv inR1*)
from p **have** $p: F, \Gamma' \Rightarrow \Delta'$ **by**(*induction* Γ' *arbitrary:* Δ' ; *simp add:* *NotR-inv inL1*)
show *?case using p at n by blast*
qed

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**

$\Gamma \Rightarrow F, \Delta$

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

$\text{atoms } F \subseteq \text{atoms-mset } (\Gamma + \Delta)$

$\text{atoms } F \subseteq \text{atoms-mset } (\Gamma' + \Delta')$

proof –

have *split-seq:* $(\exists H'. H = f F J, H') \vee (\exists I'. I = f F J, I')$ **if** $f F J, G = H + I$
for $f F G H I J$

proof –

from *that* **have** $f F J \in \# H + I$ **by**(*metis* (*mono-tags*) *add-ac(2)* *union-single-eq-member*)

thus *?thesis* **by** (*meson multi-member-split union-iff*)

qed

have *mini:* $\exists F. \Gamma \Rightarrow F, \Delta \wedge F, \Gamma' \Rightarrow \Delta' \wedge$

$\text{atoms } F \subseteq \text{atoms-mset } (\Gamma + \Delta) \wedge \text{atoms } F \subseteq \text{atoms-mset } (\Gamma' + \Delta') \wedge$
is-mini-formula F

if $\Gamma + \Gamma' \Rightarrow \Delta + \Delta'$ *is-mini-mset* $(\Gamma + \Gamma' + \Delta + \Delta')$ **for** $\Gamma \Gamma' \Delta \Delta'$

using *that* **proof**(*induction* $\Gamma + \Gamma' \Delta + \Delta'$ *arbitrary:* $\Gamma \Gamma' \Delta \Delta'$ *rule:* *SCp.induct*)

case *BotL* **thus** *?case* **proof**(*cases; intro exI*)

assume $\perp \in \# \Gamma$ **with** *BotL*

```

show  $\Gamma \Rightarrow \perp, \Delta \wedge \perp, \Gamma' \Rightarrow \Delta' \wedge \text{atoms } \perp \subseteq \text{atoms-mset } (\Gamma + \Delta) \wedge \text{atoms}$ 
 $\perp \subseteq \text{atoms-mset } (\Gamma' + \Delta') \wedge \text{is-mini-formula } \perp$ 
  by(simp add: SCp.BotL)
next
  assume  $\neg(\perp \in\# \Gamma)$  with BotL
  show  $\Gamma \Rightarrow \top, \Delta \wedge \top, \Gamma' \Rightarrow \Delta' \wedge \text{atoms } \top \subseteq \text{atoms-mset } (\Gamma + \Delta) \wedge \text{atoms}$ 
 $\top \subseteq \text{atoms-mset } (\Gamma' + \Delta') \wedge \text{is-mini-formula } \top$ 
  by(auto simp add: Top-def SCp.ImpR SCp.ImpL SCp.BotL intro!: SCp.intros(3-))
qed
next
  case (Ax k)
  let  $?ss = \lambda F. (\Gamma \Rightarrow F, \Delta \wedge F, \Gamma' \Rightarrow \Delta' \wedge \text{is-mini-formula } F)$ 
  have  $\text{ff}: ?ss \perp$  if Atom k  $\in\# \Gamma$  Atom k  $\in\# \Delta$ 
    using SCp.BotL SCp.Ax[of k] that by auto
  have  $\text{fs}: ?ss (\text{Atom } k)$  if Atom k  $\in\# \Gamma$  Atom k  $\in\# \Delta'$ 
    using that by(auto intro!: SCp.Ax[where k=k])
  have  $\text{sf}: ?ss ((\text{Atom } k) \rightarrow \perp)$  if Atom k  $\in\# \Gamma'$  Atom k  $\in\# \Delta$ 
    using that by(auto intro!: SCp.ImpR SCp.ImpL intro: SCp.Ax[where k=k])
SCp.BotL
  have  $\text{ss}: ?ss \top$  if Atom k  $\in\# \Gamma'$  Atom k  $\in\# \Delta'$ 
  unfolding Top-def using that SCp.ImpR SCp.Ax BotL-canonical by fastforce
  have in-sumE:  $\llbracket A \in\# (F + G); A \in\# F \Longrightarrow P; A \in\# G \Longrightarrow P \rrbracket \Longrightarrow P$  for
A F G P by fastforce
  have trust-firstE:  $P F \Longrightarrow Q F \Longrightarrow \exists F. P F \wedge Q F$  for P Q F by blast
  from Ax show ?case by(elim in-sumE) (frule (1) ff fs sf ss; elim conjE
trust-firstE; force)+
next
  case (ImpR F G Δ'')
  note split-seq[of Imp, OF ImpR(3)]
  thus ?case proof(elim disjE exE)
    fix H' assume  $\Delta: \Delta = F \rightarrow G, H'$ 
    have  $F, \Gamma + \Gamma' = (F, \Gamma) + \Gamma' G, \Delta'' = (G, \Delta - \{\#F \rightarrow G\# \}) + \Delta'$ 
is-mini-mset  $((F, \Gamma) + \Gamma' + (G, \Delta - \{\#F \rightarrow G\# \}) + \Delta')$ 
    using that ImpR(3-) by (simp-all add: union-assoc Δ)
    from ImpR(2)[OF this] obtain Fa where Fam:
       $F, \Gamma \Rightarrow Fa, G, H' Fa, \Gamma' \Rightarrow \Delta' \text{ is-mini-formula } Fa$ 
       $\text{atoms } Fa \subseteq \text{atoms-mset } ((F, \Gamma) + (G, H'))$   $\text{atoms } Fa \subseteq \text{atoms-mset } (\Gamma' +$ 
 $\Delta')$  unfolding  $\Delta$  by auto
    thus ?thesis unfolding  $\Delta$  proof(intro exI[where x=Fa] conjI (is-mini-formula
Fa))
      show  $\Gamma \Rightarrow Fa, F \rightarrow G, H'$  using Fam by(intro SCp.ImpR[THEN inR1];
fast)
      show  $Fa, \Gamma' \Rightarrow \Delta'$  using Fam by blast
      show  $\text{atoms } Fa \subseteq \text{atoms-mset } (\Gamma + (F \rightarrow G, H'))$   $\text{atoms } Fa \subseteq \text{atoms-mset}$ 
 $(\Gamma' + \Delta')$  using Fam by auto
    qed
next
  fix I' assume  $\Delta': \Delta' = F \rightarrow G, I'$ 
  have  $F, \Gamma + \Gamma' = \Gamma + (F, \Gamma') G, \Delta'' = \Delta + (G, I') \text{ is-mini-mset } (\Gamma + (F,$ 

```

```

 $\Gamma') + \Delta + (G, I')$ 
  using ImpR(3-) by (simp-all add: add.left-commute  $\Delta'$ )
  from ImpR(2)[OF this] obtain Fa m where Fam:
     $\Gamma \Rightarrow Fa, \Delta Fa, F, \Gamma' \Rightarrow G, I'$  is-mini-formula Fa
     $atoms Fa \subseteq atoms-mset (\Gamma + \Delta) atoms Fa \subseteq atoms-mset ((F, \Gamma') + (G,$ 
 $I'))$  unfolding  $\Delta'$  by auto
  show ?thesis unfolding  $\Delta'$  proof(intro exI[where  $x=Fa$ ] conjI (is-mini-formula
Fa))
    show  $\Gamma \Rightarrow Fa, \Delta$  using Fam by fast
    show  $Fa, \Gamma' \Rightarrow F \rightarrow G, I'$  using Fam by (simp add: SCp.ImpR inL1)
    show  $atoms Fa \subseteq atoms-mset (\Gamma + \Delta) atoms Fa \subseteq atoms-mset (\Gamma' + (F$ 
 $\rightarrow G, I'))$  using Fam by auto
  qed
  qed
  next
  case (ImpL  $\Gamma'' F G$ )
  note split-seq[of Imp, OF ImpL(5)]
  thus ?case proof(elim disjE exE)
    fix H' assume  $\Gamma: \Gamma = F \rightarrow G, H'$ 
    from  $\Gamma$  have  $*$ :  $\Gamma'' = \Gamma' + H' F, \Delta + \Delta' = \Delta' + (F, \Delta)$ 
      using ImpL(5-) by (simp-all add: union-assoc  $\Gamma$ )
    hence is-mini-mset ( $\Gamma' + H' + \Delta' + (F, \Delta)$ ) using ImpL(6) by(auto simp
add:  $\Gamma$ )
    from ImpL(2)[OF * this] obtain Fa where IH1:  $\Gamma' \Rightarrow Fa, \Delta' Fa, H' \Rightarrow F,$ 
 $\Delta$ 
       $atoms Fa \subseteq atoms-mset (H' + (F, \Delta)) atoms Fa \subseteq atoms-mset (\Gamma' + \Delta')$ 
is-mini-formula Fa by blast
    from  $\Gamma$  have  $*$ :  $G, \Gamma'' = (G, H') + \Gamma' \Delta + \Delta' = \Delta + \Delta'$ 
      using ImpL(5-) by (simp-all add: union-assoc)
    hence is-mini-mset ( $((G, H') + \Gamma' + \Delta + \Delta')$ ) using ImpL(6) by(simp add:
 $\Gamma$ )
    from ImpL(4)[OF * this] obtain Ga where IH2:  $G, H' \Rightarrow Ga, \Delta Ga, \Gamma'$ 
 $\Rightarrow \Delta'$ 
       $atoms Ga \subseteq atoms-mset ((G, H') + \Delta) atoms Ga \subseteq atoms-mset (\Gamma' + \Delta')$ 
is-mini-formula Ga by blast

```

A big part of the difficulty of this proof is finding a way to instantiate the IHs. Interestingly, this is not the only way that works. Originally, we used $\Gamma'' = H' + \Gamma'$ and $F, \Delta + \Delta' = (F, \Delta) + \Delta'$ to instantiate the IH (which is in some sense more natural, less use of commutativity). This gave us a *Fa* that verifies $H' \Rightarrow Fa, F, \Delta$ and $Fa, \Gamma' \Rightarrow \Delta'$ and resulted in the interpolant *to-mini-formula* ($Fa \vee Ga$).

```

  thm ImpL(2)[of H'  $\Gamma'$  F,  $\Delta \Delta'$ ]
  let  $?w = Fa \rightarrow Ga$ 
  show ?thesis proof(intro exI[where  $x=?w$ ] conjI)
    from IH1(1) IH2(2) show  $?w, \Gamma' \Rightarrow \Delta'$ 
      by (simp add: SCp.ImpL)
    from IH1(2) IH2(1) show  $\Gamma \Rightarrow ?w, \Delta$  unfolding  $\Gamma$ 
    by(intro SCp.ImpL inR1[OF SCp.ImpR] SCp.ImpR) (simp-all add: weakenR

```

```

weakenL)
  show atoms ?w ⊆ atoms-mset (Γ + Δ) atoms ?w ⊆ atoms-mset (Γ' + Δ')
  using IH1(3-) IH2(3-) unfolding Γ by auto
  show is-mini-formula ?w using ⟨is-mini-formula Fa⟩ ⟨is-mini-formula Ga⟩
by simp
qed
next
fix I' assume Γ': Γ' = F → G, I' note Impl(5)[unfolded Γ']
from Γ' have *: Γ'' = Γ + I' F, Δ + Δ' = Δ + (F, Δ')
  using Impl(5-) by(simp-all add: union-assoc add-ac(2,3))
  hence is-mini-mset (Γ + I' + Δ + (F, Δ')) using Impl(6) by(auto simp
add: Γ')
from Impl(2)[OF * this] obtain Fa
  where IH1: Γ ⇒ Fa, Δ Fa, I' ⇒ F, Δ' is-mini-formula Fa
  atoms Fa ⊆ atoms-mset (I' + (F, Δ')) atoms Fa ⊆ atoms-mset (Γ + Δ) by
blast
from Γ' have *: G, Γ'' = Γ + (G, I') Δ + Δ' = Δ + Δ'
  using Impl(5) by (simp-all add: add-ac ⟨Γ'' = Γ + I'⟩)
  have is-mini-mset (Γ + (G, I') + Δ + Δ') using Impl(6) by(auto simp
add: Γ')
from Impl(4)[OF * this] obtain Ga l
  where IH2: Γ ⇒ Ga, Δ Ga, G, I' ⇒ Δ' is-mini-formula Ga
  atoms Ga ⊆ atoms-mset ((G, I') + Δ') atoms Ga ⊆ atoms-mset (Γ + Δ)
by blast

```

Same thing as in the other case, just with G , $\Gamma'' = (G, I') + \Gamma$, $\Delta + \Delta' = \Delta' + \Delta$, $\Gamma'' = I' + \Gamma$, and $F, \Delta + \Delta' = (F, \Delta') + \Delta$ resulting in *to-mini-formula* $(\neg (Fa \vee Ga))$

```

let ?w = (Ga → (Fa → ⊥)) → ⊥
  have ?w = to-mini-formula (Ga ∧ Fa) by (simp add: IH1(3) IH2(3)
mini-to-mini)
show ?thesis proof(intro exI[of - ?w] conjI)
  from IH1(1) IH2(1) show Γ ⇒ ?w, Δ
  by(intro SCp.ImpR SCp.ImpL) (simp-all add: inR1 weakenR BotL-canonical)
  from IH1(2) IH2(2) show ?w, Γ' ⇒ Δ' unfolding Γ'
  by(blast intro!: SCp.ImpL SCp.ImpR dest: weakenL weakenR)+
  show atoms ?w ⊆ atoms-mset (Γ + Δ)
  atoms ?w ⊆ atoms-mset (Γ' + Δ') using IH1(3-) IH2(3-) unfolding
Γ' by auto
  show is-mini-formula ?w using ⟨is-mini-formula Fa⟩ ⟨is-mini-formula Ga⟩
by simp
qed
qed
next

```

The rest is just those cases that can't happen because of the mini formula property.

qed (*metis add.commute is-mini-formula.simps union-iff union-single-eq-member*)+

```

def tms  $\equiv$  image-mset to-mini-formula :: 'a formula multiset  $\Rightarrow$  'a formula
multiset
have [simp]: tms (A + B) = tms A + tms B tms {#F#} = {#to-mini-formula
F#} for A B F unfolding tms-def by simp-all
have [simp]: atoms-mset (tms  $\Gamma$ ) = atoms-mset  $\Gamma$  for  $\Gamma$  unfolding tms-def
using mini-formula-atoms by fastforce
have imm: is-mini-mset (tms  $\Gamma$  + tms  $\Gamma'$  + tms  $\Delta$  + tms  $\Delta'$ ) unfolding tms-def
by auto
from assms have tms  $\Gamma$  + tms  $\Gamma' \Rightarrow$  tms  $\Delta$  + tms  $\Delta'$  unfolding tms-def using
SC-full-to-mini by force
from mini[OF this imm] obtain F where hp:
tms  $\Gamma \Rightarrow$  F, tms  $\Delta$  F, tms  $\Gamma' \Rightarrow$  tms  $\Delta'$ 
and su: atoms F  $\subseteq$  atoms-mset (tms  $\Gamma$  + tms  $\Delta$ ) atoms F  $\subseteq$  atoms-mset (tms
 $\Gamma' +$  tms  $\Delta'$ )
and mf: is-mini-formula F by blast
from hp mf have tms  $\Gamma \Rightarrow$  tms (F,  $\Delta$ ) tms (F,  $\Gamma') \Rightarrow$  tms  $\Delta'$  using mini-to-mini[where
'a='a] unfolding tms-def by simp-all
hence  $\Gamma \Rightarrow$  F,  $\Delta$  F,  $\Gamma' \Rightarrow$   $\Delta'$  using SC-mini-to-full unfolding tms-def by
blast+
with su show ?thesis using  $\langle \wedge \Gamma. \text{atoms-mset} (\text{tms } \Gamma) = \text{atoms-mset } \Gamma \rangle$  image-mset-union
that by auto
qed

```

Note that there is an extension to Craig interpolation: One can show that atoms that only appear positively/negatively in the original formulas will only appear positively/negatively in the interpolant.

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

theorem *SC-Craig-interpolation-pn*:

```

assumes  $\Gamma + \Gamma' \Rightarrow \Delta + \Delta'$ 
obtains F where
   $\Gamma \Rightarrow$  F,  $\Delta$ 
  F,  $\Gamma' \Rightarrow \Delta'$ 
  fst (pn-atoms F)  $\subseteq$  (patoms-mset  $\Gamma \cup$  natoms-mset  $\Delta$ )  $\cap$  (natoms-mset  $\Gamma' \cup$ 
patoms-mset  $\Delta'$ )
  snd (pn-atoms F)  $\subseteq$  (natoms-mset  $\Gamma \cup$  patoms-mset  $\Delta$ )  $\cap$  (patoms-mset  $\Gamma' \cup$ 
natoms-mset  $\Delta'$ )
proof –
have split-seq:  $(\exists H'. H = f F J, H') \vee (\exists I'. I = f F J, I')$  if f F J, G = H + I
for f F G H I J
proof –
from that have f F J  $\in$  # H + I by (metis (mono-tags) add-ac(2) union-single-eq-member)
thus ?thesis by (meson multi-member-split union-iff)
qed
have mini:  $\exists F ::$  'a formula.  $\Gamma \Rightarrow$  F,  $\Delta \wedge$  F,  $\Gamma' \Rightarrow \Delta' \wedge$ 
fst (pn-atoms F)  $\subseteq$  (patoms-mset  $\Gamma \cup$  natoms-mset  $\Delta$ )  $\cap$  (natoms-mset  $\Gamma' \cup$ 
patoms-mset  $\Delta'$ )  $\wedge$ 
snd (pn-atoms F)  $\subseteq$  (natoms-mset  $\Gamma \cup$  patoms-mset  $\Delta$ )  $\cap$  (patoms-mset  $\Gamma' \cup$ 

```



```

natoms-mset  $\Delta'$ )  $\wedge$  is-mini-formula  $F$ 
  if  $\Gamma + \Gamma' \Rightarrow \Delta + \Delta'$  is-mini-mset ( $\Gamma + \Gamma' + \Delta + \Delta'$ ) for  $\Gamma \Gamma' \Delta \Delta'$ 
  using that proof(induction  $\Gamma + \Gamma' \Delta + \Delta'$  arbitrary:  $\Gamma \Gamma' \Delta \Delta'$  rule: SCp.induct)
  case BotL
    let  $?om = \lambda F. \text{fst } (pn\text{-atoms } F) \subseteq (patoms\text{-mset } \Gamma \cup natoms\text{-mset } \Delta) \cap$ 
    (natoms-mset  $\Gamma' \cup patoms\text{-mset } \Delta')$   $\wedge$ 
       $\text{snd } (pn\text{-atoms } F) \subseteq (natoms\text{-mset } \Gamma \cup patoms\text{-mset } \Delta) \cap (patoms\text{-mset } \Gamma'$ 
     $\cup natoms\text{-mset } \Delta')$   $\wedge$  is-mini-formula ( $F :: 'a \text{ formula}$ )
    show  $?case$  proof(cases; intro exI)
    assume  $\perp \in\# \Gamma$  with BotL
    show  $\Gamma \Rightarrow \perp, \Delta \wedge \perp, \Gamma' \Rightarrow \Delta' \wedge ?om \perp$  by(simp add: SCp.BotL)
  next
    assume  $\neg(\perp \in\# \Gamma)$  with BotL
    show  $\Gamma \Rightarrow \top, \Delta \wedge \top, \Gamma' \Rightarrow \Delta' \wedge ?om \top$ 
    by(auto simp add: Top-def SCp.ImpR SCp.ImpL SCp.BotL prod-unions-def
intro!: SCp.intros( $\beta-$ ))
  qed
next
  case (Ax k)
    let  $?ss = \lambda F. (\Gamma \Rightarrow F, \Delta \wedge F, \Gamma' \Rightarrow \Delta' \wedge \text{fst } (pn\text{-atoms } F) \subseteq (patoms\text{-mset}$ 
     $\Gamma \cup natoms\text{-mset } \Delta) \cap (natoms\text{-mset } \Gamma' \cup patoms\text{-mset } \Delta') \wedge$ 
       $\text{snd } (pn\text{-atoms } F) \subseteq (natoms\text{-mset } \Gamma \cup patoms\text{-mset } \Delta) \cap (patoms\text{-mset } \Gamma'$ 
     $\cup natoms\text{-mset } \Delta') \wedge \text{is-mini-formula } F)$ 
    have  $ff: ?ss \perp$  if  $Atom\ k \in\# \Gamma$   $Atom\ k \in\# \Delta$ 
    using SCp.BotL SCp.Ax[of k] that by auto
    have  $fs: ?ss (Atom\ k)$  if  $Atom\ k \in\# \Gamma$   $Atom\ k \in\# \Delta'$ 
    using that by(force intro!: SCp.Ax[where  $k=k$ ])
    have  $sf: ?ss ((Atom\ k) \rightarrow \perp)$  if  $Atom\ k \in\# \Gamma'$   $Atom\ k \in\# \Delta$ 
    using that by(auto intro!: SCp.ImpR SCp.ImpL intro: SCp.Ax[where  $k=k$ ]
SCp.BotL exI[where  $x=Atom\ k$ ] simp add: prod-unions-def; force)
    have  $ss: ?ss \top$  if  $Atom\ k \in\# \Gamma'$   $Atom\ k \in\# \Delta'$ 
    unfolding Top-def using that SCp.ImpR by (auto simp add: prod-unions-def
SCp.Ax)
    have in-sumE:  $\llbracket A \in\# (F + G); A \in\# F \Longrightarrow P; A \in\# G \Longrightarrow P \rrbracket \Longrightarrow P$  for
     $A\ F\ G\ P$  by fastforce
    have trust-firstE:  $P\ F \Longrightarrow Q\ F \Longrightarrow \exists F. P\ F \wedge Q\ F$  for  $P\ Q\ F$  by blast
    from Ax show  $?case$  by(elim in-sumE) (frule (1)  $ff\ fs\ sf\ ss$ ; elim conjE
trust-firstE; force)+
  next
next
  case (ImpR F G  $\Delta''$ )
  note split-seq[of Imp, OF ImpR( $\beta$ )]
  thus  $?case$  proof(elim disjE exE)
    fix  $H'$  assume  $\Delta: \Delta = F \rightarrow G, H'$ 
    have  $F, \Gamma + \Gamma' = (F, \Gamma) + \Gamma', G, \Delta'' = (G, \Delta - \{\#F \rightarrow G\# \}) + \Delta'$ 
is-mini-mset ( $(F, \Gamma) + \Gamma' + (G, \Delta - \{\#F \rightarrow G\# \}) + \Delta'$ )
    using that ImpR( $\beta-$ ) by (simp-all add: union-assoc  $\Delta$ )
    from ImpR( $\beta$ )[OF this] obtain  $Fa$  where  $Fam$ :
       $F, \Gamma \Rightarrow Fa, G, H' Fa, \Gamma' \Rightarrow \Delta' \text{ is-mini-formula } Fa$ 

```

```

      fst (pn-atoms Fa) ⊆ (patoms-mset (F, Γ) ∪ natoms-mset (G, H')) ∩
(natoms-mset Γ' ∪ patoms-mset Δ')
      snd (pn-atoms Fa) ⊆ (natoms-mset (F, Γ) ∪ patoms-mset (G, H')) ∩
(patoms-mset Γ' ∪ natoms-mset Δ') unfolding Δ by auto
      thus ?thesis unfolding Δ proof(intro exI[where x=Fa] conjI (is-mini-formula
Fa))
      show Γ ⇒ Fa, F → G, H' using Fam by(intro SCp.ImpR[THEN inR1];
fast)
      show Fa, Γ' ⇒ Δ' using Fam by blast
      show fst (pn-atoms Fa) ⊆ (patoms-mset Γ ∪ natoms-mset (F → G, H'))
∩ (natoms-mset Γ' ∪ patoms-mset Δ')
      snd (pn-atoms Fa) ⊆ (natoms-mset Γ ∪ patoms-mset (F → G, H')) ∩
(patoms-mset Γ' ∪ natoms-mset Δ')
      using Fam(4-) by (auto simp: prod-unions-def split: prod.splits)
    qed
  next
  fix I' assume Δ': Δ' = F → G, I'
  have F, Γ + Γ' = Γ + (F,Γ') G, Δ'' = Δ + (G, I') is-mini-mset (Γ + (F,
Γ') + Δ + (G, I'))
  using ImpR(3-) by (simp-all add: add.left-commute Δ')
  from ImpR(2)[OF this] obtain Fa m where Fam:
  Γ ⇒ Fa, Δ Fa, F, Γ' ⇒ G, I' is-mini-formula Fa
  fst (pn-atoms Fa) ⊆ (patoms-mset Γ ∪ natoms-mset Δ) ∩ (natoms-mset
(F, Γ') ∪ patoms-mset (G, I'))
  snd (pn-atoms Fa) ⊆ (natoms-mset Γ ∪ patoms-mset Δ) ∩ (patoms-mset
(F, Γ') ∪ natoms-mset (G, I')) unfolding Δ' by auto
  show ?thesis unfolding Δ' proof(intro exI[where x=Fa] conjI (is-mini-formula
Fa))
  show Γ ⇒ Fa, Δ using Fam by fast
  show Fa, Γ' ⇒ F → G, I' using Fam by (simp add: SCp.ImpR inL1)
  show fst (pn-atoms Fa) ⊆ (patoms-mset Γ ∪ natoms-mset Δ) ∩ (natoms-mset
Γ' ∪ patoms-mset (F → G, I'))
  snd (pn-atoms Fa) ⊆ (natoms-mset Γ ∪ patoms-mset Δ) ∩ (patoms-mset
Γ' ∪ natoms-mset (F → G, I'))
  using Fam by (auto simp: prod-unions-def split: prod.splits)
    qed
  qed
  next
  next
  case (ImpL Γ'' F G)
  note split-seq[of Imp, OF ImpL(5)]
  thus ?case proof(elim disjE exE)
  fix H' assume Γ: Γ = F → G, H'
  from Γ have *: Γ'' = Γ' + H' F, Δ + Δ' = Δ' + (F,Δ)
  using ImpL(5-) by (simp-all add: union-assoc Γ)
  hence is-mini-mset (Γ' + H' + Δ' + (F, Δ)) using ImpL(6) by(auto simp
add: Γ)
  from ImpL(2)[OF * this] obtain Fa where IH1: Γ' ⇒ Fa, Δ' Fa, H' ⇒ F,
Δ

```

$\text{fst } (pn\text{-atoms } Fa) \subseteq (\text{patoms-mset } \Gamma' \cup \text{natoms-mset } \Delta') \cap (\text{natoms-mset } H' \cup \text{patoms-mset } (F, \Delta))$
 $\text{snd } (pn\text{-atoms } Fa) \subseteq (\text{natoms-mset } \Gamma' \cup \text{patoms-mset } \Delta') \cap (\text{patoms-mset } H' \cup \text{natoms-mset } (F, \Delta))$ *is-mini-formula* Fa **by** *blast*
from Γ **have** $*$: $G, \Gamma'' = (G, H') + \Gamma' \Delta + \Delta' = \Delta + \Delta'$
using *Impl(5-)* **by** (*simp-all add: union-assoc*)
hence *is-mini-mset* $((G, H') + \Gamma' + \Delta + \Delta')$ **using** *Impl(6)* **by** (*simp add:*
 Γ)
from *Impl(4)*[*OF * this*] **obtain** Ga **where** *IH2*: $G, H' \Rightarrow Ga, \Delta Ga, \Gamma' \Rightarrow \Delta'$
 $\text{fst } (pn\text{-atoms } Ga) \subseteq (\text{patoms-mset } (G, H') \cup \text{natoms-mset } \Delta) \cap (\text{natoms-mset } \Gamma' \cup \text{patoms-mset } \Delta')$
 $\text{snd } (pn\text{-atoms } Ga) \subseteq (\text{natoms-mset } (G, H') \cup \text{patoms-mset } \Delta) \cap (\text{patoms-mset } \Gamma' \cup \text{natoms-mset } \Delta')$ *is-mini-formula* Ga **by** *blast*
thm *Impl(2)*[*of H' Γ' F, Δ Δ'*]
let $?w = Fa \rightarrow Ga$
show *?thesis proof* (*intro exI*[**where** $x=?w$] *conjI*)
from *IH1(1)* *IH2(2)* **show** $?w, \Gamma' \Rightarrow \Delta'$
by (*simp add: SCp.Impl*)
from *IH1(2)* *IH2(1)* **show** $\Gamma \Rightarrow ?w, \Delta$ **unfolding** Γ
by (*intro SCp.Impl inR1*[*OF SCp.ImplR*] *SCp.ImplR*) (*simp-all add: weakenR weakenL*)
show $\text{fst } (pn\text{-atoms } ?w) \subseteq (\text{patoms-mset } \Gamma \cup \text{natoms-mset } \Delta) \cap (\text{natoms-mset } \Gamma' \cup \text{patoms-mset } \Delta')$
 $\text{snd } (pn\text{-atoms } ?w) \subseteq (\text{natoms-mset } \Gamma \cup \text{patoms-mset } \Delta) \cap (\text{patoms-mset } \Gamma' \cup \text{natoms-mset } \Delta')$
using *IH1(3-)* *IH2(3-)* **unfolding** Γ **by** (*auto simp: prod-unions-def split: prod.splits*)
show *is-mini-formula* $?w$ **using** $\langle \text{is-mini-formula } Fa \rangle \langle \text{is-mini-formula } Ga \rangle$
by *simp*
qed
next
fix I' **assume** $\Gamma': \Gamma' = F \rightarrow G, I'$ **note** *Impl(5)*[*unfolded Γ'*]
from Γ' **have** $*$: $\Gamma'' = \Gamma + I' F, \Delta + \Delta' = \Delta + (F, \Delta')$
using *Impl(5-)* **by** (*simp-all add: union-assoc add-ac(2,3)*)
hence *is-mini-mset* $(\Gamma + I' + \Delta + (F, \Delta'))$ **using** *Impl(6)* **by** (*auto simp add: Γ'*)
from *Impl(2)*[*OF * this*] **obtain** Fa
where *IH1*: $\Gamma \Rightarrow Fa, \Delta Fa, I' \Rightarrow F, \Delta'$ *is-mini-formula* Fa
 $\text{fst } (pn\text{-atoms } Fa) \subseteq (\text{patoms-mset } \Gamma \cup \text{natoms-mset } \Delta) \cap (\text{natoms-mset } I' \cup \text{patoms-mset } (F, \Delta'))$
 $\text{snd } (pn\text{-atoms } Fa) \subseteq (\text{natoms-mset } \Gamma \cup \text{patoms-mset } \Delta) \cap (\text{patoms-mset } I' \cup \text{natoms-mset } (F, \Delta'))$ **by** *blast*
from Γ' **have** $*$: $G, \Gamma'' = (G, I') + \Gamma \Delta + \Delta' = \Delta' + \Delta$
using *Impl(5)* **by** (*simp-all add: add-ac* $\langle \Gamma'' = \Gamma + I' \rangle$)
have *is-mini-mset* $((G, I') + \Gamma + \Delta' + \Delta)$ **using** *Impl(6)* **by** (*auto simp add: Γ'*)
from *Impl(4)*[*OF * this*] **obtain** Ga
where *IH2*: $G, I' \Rightarrow Ga, \Delta' Ga, \Gamma \Rightarrow \Delta$ *is-mini-formula* Ga

```

    fst (pn-atoms Ga) ⊆ (patoms-mset (G, I') ∪ natoms-mset Δ') ∩ (natoms-mset
Γ ∪ patoms-mset Δ)
    snd (pn-atoms Ga) ⊆ (natoms-mset (G, I') ∪ patoms-mset Δ') ∩ (patoms-mset
Γ ∪ natoms-mset Δ) by blast
    let ?w = (Fa → Ga) → ⊥
    have ?w = to-mini-formula (¬(Fa → Ga)) unfolding to-mini-formula.simps
mini-to-mini[OF IH1(3)] mini-to-mini[OF IH2(3)] by (simp add: IH1(3) IH2(3)
)
    show ?thesis proof(intro exI[of - ?w] conjI)
      from IH1(1) IH2(2) show Γ ⇒ ?w, Δ
      by(intro SCp.ImpR SCp.ImpL) (simp-all add: inR1 weakenR BotL-canonical)
      from IH1(2) IH2(1) show ?w, Γ' ⇒ Δ' unfolding Γ'
      by(blast intro!: SCp.ImpL SCp.ImpR dest: weakenL weakenR)+
      show fst (pn-atoms ?w) ⊆ (patoms-mset Γ ∪ natoms-mset Δ) ∩ (natoms-mset
Γ' ∪ patoms-mset Δ')
      snd (pn-atoms ?w) ⊆ (natoms-mset Γ ∪ patoms-mset Δ) ∩ (patoms-mset
Γ' ∪ natoms-mset Δ')
      using IH1(4-) IH2(4-) unfolding Γ' by (auto simp: prod-unions-def
split: prod.splits)
      show is-mini-formula ?w using ⟨is-mini-formula Fa⟩ ⟨is-mini-formula Ga⟩
by simp
      qed
      qed
      next
      qed (metis add.commute is-mini-formula.simps union-iff union-single-eq-member)+
      def tms ≡ image-mset to-mini-formula :: 'a formula multiset ⇒ 'a formula
multiset
      have [simp]: tms (A + B) = tms A + tms B tms {#F#} = {#to-mini-formula
F#} for A B F unfolding tms-def by simp-all
      have imm: is-mini-mset (tms Γ + tms Γ' + tms Δ + tms Δ') unfolding tms-def
by auto
      from assms have tms Γ + tms Γ' ⇒ tms Δ + tms Δ' unfolding tms-def using
SC-full-to-mini by force
      from mini[OF this imm] obtain F where hp:
        tms Γ ⇒ F, tms Δ F, tms Γ' ⇒ tms Δ'
      and su: fst (pn-atoms F) ⊆ (patoms-mset (tms Γ) ∪ natoms-mset (tms Δ))
∩ (natoms-mset (tms Γ') ∪ patoms-mset (tms Δ'))
      and sn: snd (pn-atoms F) ⊆ (natoms-mset (tms Γ) ∪ patoms-mset (tms Δ)) ∩
(patoms-mset (tms Γ') ∪ natoms-mset (tms Δ'))
      and mf: is-mini-formula F by blast
      from hp mf have tms Γ ⇒ tms (F, Δ) tms (F, Γ') ⇒ tms Δ' using mini-to-mini[where
'a='a] unfolding tms-def by simp-all
      hence *: Γ ⇒ F, Δ F, Γ' ⇒ Δ' using SC-mini-to-full unfolding tms-def by
blast+
      have pn-atoms (to-mini-formula F) = pn-atoms F for F :: 'a formula by(induction
F; simp add: prod-unions-def split: prod.splits)
      hence pn-tms: patoms-mset (tms Γ) = patoms-mset Γ natoms-mset (tms Γ) =
natoms-mset Γ for Γ unfolding tms-def by simp-all
      from su[unfolded pn-tms] show ?thesis using that[of F, OF * -] by auto

```

qed

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.