# Propositional Resolution and Prime Implicates Generation 

Nicolas Peltier<br>Laboratory of Informatics of Grenoble/CNRS<br>University Grenoble Alps

September 13, 2023


#### Abstract

We provide formal proofs in Isabelle-HOL (using mostly structured Isar proofs) of the soundness and completeness of the Resolution rule in propositional logic. The completeness proofs take into account the usual redundancy elimination rules (namely tautology elimination and subsumption), and several refinements of the Resolution rule are considered: ordered resolution (with selection functions), positive and negative resolution, semantic resolution and unit resolution (the latter refinement is complete only for clause sets that are Horn-renamable). We also define a concrete procedure for computing saturated sets and establish its soundness and completeness. The clause sets are not assumed to be finite, so that the results can be applied to formulas obtained by grounding sets of first-order clauses (however, a total ordering among atoms is assumed to be given).

Next, we show that the unrestricted Resolution rule is deductivecomplete, in the sense that it is able to generate all (prime) implicates of any set of propositional clauses (i.e., all entailment-minimal, nonvalid, clausal consequences of the considered set). The generation of prime implicates is an important problem, with many applications in artificial intelligence and verification (for abductive reasoning, knowledge compilation, diagnosis, debugging etc.). We also show that implicates can be computed in an incremental way, by fixing an ordering among all the atoms and resolving upon these atoms one by one in the considered order (with no backtracking). This feature is critical for the efficient computation of prime implicates. Building on these results, we provide a procedure for computing such implicates and establish its soundness and completeness.


## Contents

3 Inference Rules ..... 5
3.1 Unrestricted Resolution ..... 6
3.2 Ordered Resolution ..... 7
3.3 Ordered Resolution with Selection ..... 8
3.4 Semantic Resolution ..... 8
3.5 Unit Resolution ..... 9
3.6 Positive and Negative Resolution ..... 10
4 Redundancy Elimination Rules ..... 12
5 Renaming ..... 17
6 Soundness ..... 24
7 Refutational Completeness ..... 26
7.1 Ordered Resolution ..... 27
7.2 Ordered Resolution with Selection ..... 35
7.3 Semantic Resolution ..... 42
7.4 Positive and Negative Resolution ..... 45
7.5 Unit Resolution and Horn Renamable Clauses ..... 46
8 Computation of Saturated Clause Sets ..... 47
9 Prime Implicates Generation ..... 56
9.1 Implicates and Prime Implicates ..... 56
9.2 Generation of Prime Implicates ..... 56
9.3 Incremental Prime Implicates Computation ..... 61

## 1 Syntax of Propositional Clausal Logic

We define the usual syntactic notions of clausal propositional logic. The set of atoms may be arbitrary (even uncountable), but a well-founded total order is assumed to be given.
theory Propositional-Resolution
imports Main
begin
locale propositional-atoms $=$
fixes atom-ordering :: ('at $\times$ 'at) set
assumes
atom-ordering-wf :(wf atom-ordering)
and atom-ordering-total: $(\forall x y .(x \neq y \longrightarrow((x, y) \in$ atom-ordering $\vee(y, x) \in$ atom-ordering)))

```
and atom-ordering-trans: }\forallxyz.(x,y)\in\mathrm{ atom-ordering }\longrightarrow(y,z)\in\mathrm{ atom-ordering
    \longrightarrow ( x , z ) \in \text { atom-ordering}
and atom-ordering-irrefl: }\forallxy.(x,y)\in\mathrm{ atom-ordering }\longrightarrow(y,x)\not\in\mathrm{ atom-ordering
begin
```

Literals are defined as usual and clauses and formulas are considered as sets. Clause sets are not assumed to be finite (so that the results can be applied to sets of clauses obtained by grounding first-order clauses).

```
datatype 'a Literal \(=\operatorname{Pos}{ }^{\prime} a \mid N e g ' a\)
definition atoms \(=\left\{x::^{\prime}\right.\) at. True \(\}\)
fun atom :: 'a Literal \(\Rightarrow{ }^{\prime} a\)
where
    \((\operatorname{atom}(\operatorname{Pos} A))=A \mid\)
\((\operatorname{atom}(\operatorname{Neg} A))=A\)
```

fun complement :: 'a Literal $\Rightarrow$ 'a Literal
where
$($ complement $(\operatorname{Pos} A))=(\operatorname{Neg} A) \mid$
$(\operatorname{complement}(\operatorname{Neg} A))=(\operatorname{Pos} A)$
lemma atom-property : $A=($ atom $L) \Longrightarrow(L=(\operatorname{Pos} A) \vee L=(\operatorname{Neg} A))$
by (metis atom.elims)
fun positive :: 'at Literal $\Rightarrow$ bool
where
(positive $($ Pos $A))=$ True $\mid$
$($ positive $(\operatorname{Neg} A))=$ False
fun negative :: 'at Literal $\Rightarrow$ bool
where
$($ negative $(\operatorname{Pos} A))=$ False $\mid$
$($ negative $(\operatorname{Neg} A))=$ True
type-synonym 'a Clause $=$ ' $a$ Literal set
type-synonym 'a Formula $=$ ' $a$ Clause set

Note that the clauses are not assumed to be finite (some of the properties below hold for infinite clauses).

The following functions return the set of atoms occurring in a clause or formula.
fun atoms-clause :: 'at Clause $\Rightarrow$ 'at set where atoms-clause $C=\{A . \exists L . L \in C \wedge A=\operatorname{atom}(L)\}$
fun atoms-formula :: 'at Formula $\Rightarrow$ 'at set
where atoms-formula $S=\{A . \exists C . C \in S \wedge A \in$ atoms-clause $(C)\}$
lemma atoms-formula-subset: S1 $\subseteq$ S2 $\Longrightarrow$ atoms-formula $S 1 \subseteq$ atoms-formula S2
by auto
lemma atoms-formula-union: atoms-formula $(S 1 \cup S 2)=$ atoms-formula $S 1 \cup$ atoms-formula S2
by auto
The following predicate is useful to state that every clause in a set fulfills some property.

```
definition all-fulfill :: ('at Clause \(\Rightarrow\) bool) \(\Rightarrow\) 'at Formula \(\Rightarrow\) bool
    where all-fulfill \(P S=(\forall C .(C \in S \longrightarrow(P C)))\)
```

The order on atoms induces a (non total) order among literals:
fun literal-ordering :: 'at Literal $\Rightarrow$ 'at Literal $\Rightarrow$ bool
where
$($ literal-ordering L1 L2 $)=(($ atom L1,atom L2 $) \in$ atom-ordering $)$

```
lemma literal-ordering-trans :
    assumes literal-ordering \(A B\)
    assumes literal-ordering \(B C\)
    shows literal-ordering A C
using assms(1) assms(2) atom-ordering-trans literal-ordering.simps by blast
definition strictly-maximal-literal :: 'at Clause \(\Rightarrow{ }^{\prime}\) 'at Literal \(\Rightarrow\) bool
where
    (strictly-maximal-literal \(S A) \equiv(A \in S) \wedge(\forall B .(B \in S \wedge A \neq B) \longrightarrow\)
(literal-ordering \(B A\) ))
```


## 2 Semantics

We define the notions of interpretation, satisfiability and entailment and establish some basic properties.

```
type-synonym 'a Interpretation \(=\) ' \(a\) set
fun validate-literal \(::\) 'at Interpretation \(\Rightarrow\) 'at Literal \(\Rightarrow\) bool (infix \(\models 65\) )
    where
        \((\) validate-literal \(I(\operatorname{Pos} A))=(A \in I) \mid\)
        \((\) validate-literal \(I(\operatorname{Neg} A))=(A \notin I)\)
fun validate-clause :: 'at Interpretation \(\Rightarrow\) 'at Clause \(\Rightarrow\) bool (infix \(=65\) )
    where
        \((\) validate-clause \(I C)=(\exists L .(L \in C) \wedge(\) validate-literal I L \() ~)\)
fun validate-formula :: 'at Interpretation \(\Rightarrow\) 'at Formula \(\Rightarrow\) bool (infix \(\models 65\) )
    where
```

$($ validate-formula $I S)=(\forall C .(C \in S \longrightarrow($ validate-clause $I C)))$
definition satisfiable :: 'at Formula $\Rightarrow$ bool
where

```
(satisfiable S) \equiv(\existsI. (validate-formula I S))
```

We define the usual notions of entailment between clauses and formulas.

```
definition entails :: 'at Formula \(\Rightarrow\) 'at Clause \(\Rightarrow\) bool
where
    \((\) entails \(S C) \equiv(\forall I .(\) validate-formula \(I S) \longrightarrow(\) validate-clause I C \())\)
```

lemma entails-member:
assumes $C \in S$
shows entails $S C$
using assms unfolding entails-def by simp
definition entails-formula :: 'at Formula $\Rightarrow$ 'at Formula $\Rightarrow$ bool
where (entails-formula S1 S2) $=(\forall C \in$ S2. (entails S1 C) $)$
definition equivalent :: 'at Formula $\Rightarrow$ 'at Formula $\Rightarrow$ bool
where $($ equivalent S1 S2 $)=($ entails-formula S1 S2 $\wedge$ entails-formula S2 S1)
lemma equivalent-symmetric: equivalent S1 S2 $\Longrightarrow$ equivalent S2 S1
by (simp add: equivalent-def)
lemma entailment-implies-validity:
assumes entails-formula S1 S2
assumes validate-formula I S1
shows validate-formula I S2
using assms entails-def entails-formula-def by auto
lemma validity-implies-entailment:
assumes $\forall I$. validate-formula $I S 1 \longrightarrow$ validate-formula I S2
shows entails-formula S1 S2
by (meson assms entails-def entails-formula-def validate-formula.elims(2))
lemma entails-transitive:
assumes entails-formula S1 S2
assumes entails-formula S2 S3
shows entails-formula S1 S3
by (meson assms entailment-implies-validity validity-implies-entailment)
lemma equivalent-transitive:
assumes equivalent S1 S2
assumes equivalent S2 S3
shows equivalent S1 S3
using assms entails-transitive equivalent-def by auto
lemma entailment-subset :

```
    assumes S2 \subseteqS1
    shows entails-formula S1 S2
proof -
    have }\forallLLLa.L\not\inLa\vee entails La L
        by (meson entails-member)
    thus ?thesis
        by (meson assms entails-formula-def rev-subsetD)
qed
lemma entailed-formula-entails-implicates:
    assumes entails-formula S1 S2
    assumes entails S2 C
    shows entails S1 C
using assms entailment-implies-validity entails-def by blast
```


## 3 Inference Rules

We first define an abstract notion of a binary inference rule.

```
type-synonym 'a BinaryRule \(=\) ' \(a\) Clause \(\Rightarrow\) 'a Clause \(\Rightarrow\) ' \(a\) Clause \(\Rightarrow\) bool
definition less-restrictive :: 'at BinaryRule \(\Rightarrow\) 'at BinaryRule \(\Rightarrow\) bool
where
    (less-restrictive R1 R2 \()=(\forall\) P1 P2 \(C .(\) R2 P1 P2 \(C) \longrightarrow((R 1 P 1 P 2 C) \vee(R 1\)
P2 P1 C)) )
```

The following functions allow to generate all the clauses that are deducible from a given clause set (in one step).
fun all-deducible-clauses:: 'at BinaryRule $\Rightarrow$ 'at Formula $\Rightarrow$ 'at Formula where all-deducible-clauses $R S=\{C . \exists P 1 P 2 . P 1 \in S \wedge P 2 \in S \wedge(R$ P1 P2 C) \}
fun add-all-deducible-clauses:: 'at BinaryRule $\Rightarrow$ 'at Formula $\Rightarrow$ 'at Formula where add-all-deducible-clauses $R S=(S \cup$ all-deducible-clauses $R S)$
definition derived-clauses-are-finite :: 'at BinaryRule $\Rightarrow$ bool
where derived-clauses-are-finite $R=$
$(\forall$ P1 P2 $C$. $($ finite $P 1 \longrightarrow$ finite $P 2 \longrightarrow(R$ P1 P2 $C) \longrightarrow$ finite $C))$
lemma less-restrictive-and-finite :
assumes less-restrictive R1 R2
assumes derived-clauses-are-finite R1
shows derived-clauses-are-finite R2
by (metis assms derived-clauses-are-finite-def less-restrictive-def)
We then define the unrestricted resolution rule and usual resolution refinements.

## 3．1 Unrestricted Resolution

```
definition resolvent :: 'at BinaryRule
    where
    (resolvent P1 P2 C) \(\equiv\)
    \((\exists A .((\operatorname{Pos} A) \in P 1 \wedge(N e g A) \in P 2 \wedge(C=((P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{\)
```

$\operatorname{Neg} A\})))$ )

For technical convience，we now introduce a slightly extended definition in which resolution upon a literal not occurring in the premises is allowed（the obtained resolvent is then redundant with the premises）．If the atom is fixed then this version of the resolution rule can be turned into a total function．

```
fun resolvent-upon :: 'at Clause }=>\mathrm{ 'at Clause }=>\mathrm{ 'at = 'at Clause
where
    (resolvent-upon P1 P2 A) =
    ((P1 - {Pos A}) \cup(P2 - {Neg A }))
lemma resolvent-upon-is-resolvent :
    assumes Pos A\inP1
    assumes Neg A G P2
    shows resolvent P1 P2 (resolvent-upon P1 P2 A)
using assms unfolding resolvent-def by auto
lemma resolvent-is-resolvent-upon :
    assumes resolvent P1 P2 C
    shows \existsA.C=resolvent-upon P1 P2 A
using assms unfolding resolvent-def by auto
lemma resolvent-is-finite :
    shows derived-clauses-are-finite resolvent
proof (rule ccontr)
    assume }\neg\mathrm{ derived-clauses-are-finite resolvent
    then have }\existsP1P2 C.\neg(resolvent P1 P2 C \longrightarrow finite P1 \longrightarrow finite P2 \longrightarrow
finite C)
            unfolding derived-clauses-are-finite-def by blast
    then obtain P1 P2 C where resolvent P1 P2 C finite P1 finite P2 and \negfinite
C by blast
    from〈resolvent P1 P2 C < finite P1〉〈finite P2` and <\negfinite C> show False
    unfolding resolvent-def using finite-Diff and finite-Union by auto
qed
```

In the next subsections we introduce various resolution refinements and show that they are more restrictive than unrestricted resolution．

## 3．2 Ordered Resolution

In the first refinement，resolution is only allowed on maximal literals．
definition ordered－resolvent ：：＇at Clause $\Rightarrow$＇at Clause $\Rightarrow$＇at Clause $\Rightarrow$ bool

## where

（ordered－resolvent P1 P2 $C$ ）$\equiv$
$(\exists A .((C=((P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{N e g A\})))$
$\wedge($ strictly－maximal－literal P1 $(\operatorname{Pos} A)) \wedge($ strictly－maximal－literal P2 $(N e g$ A））））

We now show that the maximal literal of the resolvent is always smaller than those of the premises．

```
lemma resolution-and-max-literal :
    assumes R= resolvent-upon P1 P2 A
    assumes strictly-maximal-literal P1 (Pos A)
    assumes strictly-maximal-literal P2 (Neg A)
    assumes strictly-maximal-literal R M
    shows (atom M,A)\inatom-ordering
proof -
    obtain MA where M=(Pos MA)\vee M = (Neg MA) using Literal.exhaust [of
M] by auto
    hence MA = (atom M) by auto
    from\strictly-maximal-literal R M〉 and <R = resolvent-upon P1 P2 A〉
        have M \inP1 - {Pos A }\veeM M P2 - {Neg A }
        unfolding strictly-maximal-literal-def by auto
    hence (MA,A)\in atom-ordering
    proof
        assume M\inP1-{ Pos A}
        from «M \inP1 - {Pos A }` and «strictly-maximal-literal P1 (Pos A)`
            have literal-ordering M (Pos A)
            unfolding strictly-maximal-literal-def by auto
        from <M = Pos MA\veeM = Neg MA> and <literal-ordering M (Pos A)>
        show (MA,A)\in atom-ordering by auto
    next
        assume M \inP2 - {Neg A }
        from <M \inP2 - {Neg A }〉 and <strictly-maximal-literal P2 (Neg A)>
        have literal-ordering M (Neg A) by (auto simp only: strictly-maximal-literal-def)
        from <M = Pos MA\veeM = Neg MA` and <literal-ordering M (Neg A)>
        show (MA,A)\in atom-ordering by auto
    qed
    from this and <MA = atom M\rangle show ?thesis by auto
qed
```


## 3．3 Ordered Resolution with Selection

In the next restriction strategy，some negative literals are selected with high－ est priority for applying the resolution rule，regardless of the ordering．Re－ laxed ordering restrictions also apply．

```
definition (selected-part Sel C) ={L.L\inC^(\existsA\inSel.L=(Neg A)) }
definition ordered-sel-resolvent :: 'at set }=>\mathrm{ ' 'at Clause => 'at Clause = 'at Clause
=> bool
```


## where

(ordered-sel-resolvent Sel P1 P2 C) $\equiv$

```
\((\exists A .((C=((P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{\operatorname{Neg} A\})))\)
    \(\wedge(\) strictly-maximal-literal P1 \((\) Pos A) \() \wedge((\) selected-part Sel P1) \(=\{ \}) \wedge\)
    \((((\) strictly-maximal-literal P2 \((\) Neg A) \() \wedge(\) selected-part Sel P2 \()=\{ \})\)
    \(\vee(\) strictly-maximal-literal \((\) selected-part Sel P2) \((\) Neg A) \())))\)
```

lemma ordered-resolvent-is-resolvent : less-restrictive resolvent ordered-resolvent using less-restrictive-def ordered-resolvent-def resolvent-upon-is-resolvent strictly-maximal-literal-def by auto

The next lemma states that ordered resolution with selection coincides with ordered resolution if the selected part is empty.
lemma ordered-sel-resolvent-is-ordered-resolvent :
assumes ordered-resolvent P1 P2 C
assumes selected-part Sel P1 $=\{ \}$
assumes selected-part Sel P2 $=\{ \}$
shows ordered-sel-resolvent Sel P1 P2 C
using assms ordered-resolvent-def ordered-sel-resolvent-def by auto
lemma ordered-resolvent-upon-is-resolvent :
assumes strictly-maximal-literal P1 (Pos A)
assumes strictly-maximal-literal P2 (Neg A)
shows ordered-resolvent P1 P2 (resolvent-upon P1 P2 A)
using assms ordered-resolvent-def by auto

### 3.4 Semantic Resolution

In this strategy, resolution is applied only if one parent is false in some (fixed) interpretation. Note that ordering restrictions still apply, although they are relaxed.
definition validated-part :: 'at set $\Rightarrow$ 'at Clause $\Rightarrow$ 'at Clause
where (validated-part I $C)=\{L . L \in C \wedge($ validate-literal I L) $\}$
definition ordered-model-resolvent ::
'at Interpretation $\Rightarrow$ 'at Clause $\Rightarrow$ 'at Clause $\Rightarrow$ 'at Clause $\Rightarrow$ bool
where
(ordered-model-resolvent I P1 P2 C) $=$
$(\exists L .(C=(P 1-\{L\} \cup(P 2-\{$ complement $L\}))) \wedge$
$(($ validated-part I P1) $=\{ \} \wedge($ strictly-maximal-literal P1 L) $)$
$\wedge($ strictly-maximal-literal (validated-part I P2) $($ complement L) $)$ )
lemma ordered-model-resolvent-is-resolvent : less-restrictive resolvent (ordered-model-resolvent I)
proof (rule ccontr)
assume $\neg$ less-restrictive resolvent (ordered-model-resolvent I)
then obtain P1 P2 C where ordered-model-resolvent I P1 P2 C and $\neg$ resolvent P1 P2 $C$
and $\neg$ resolvent P2 P1 C unfolding less－restrictive－def by auto
from＜ordered－model－resolvent I P1 P2 C〉 obtain $L$
where strictly－maximal－literal P1 L
and strictly－maximal－literal（validated－part I P2）（complement L）
and $C=(P 1-\{L\}) \cup(P 2-\{$ complement $L\})$
using ordered－model－resolvent－def［of I P1 P2 C］by auto
from 〈strictly－maximal－literal P1 L〉 have $L \in$ P1 by（simp only：strictly－maximal－literal－def）
from 〈strictly－maximal－literal（validated－part I P2）（complement L）〉 have（complement
L）$\in P 2$
by（auto simp only：strictly－maximal－literal－def validated－part－def）
obtain $A$ where $L=$ Pos $A \vee L=N e g A$ using Literal．exhaust $[o f L]$ by auto
from this and $\langle C=(P 1-\{L\}) \cup(P 2-\{$ complement $L\})\rangle$ and $\langle L \in P 1\rangle$ and 〈（complement $L) \in P 2$ 〉
have resolvent P1 P2 $C \vee$ resolvent P2 P1 $C$ unfolding resolvent－def by auto
from this and $\langle\neg$ resolvent P2 P1 $C\rangle$ and $\langle\neg$ resolvent P1 P2 $C\rangle$ show False by auto
qed

## 3．5 Unit Resolution

Resolution is applied only if one parent is unit（this restriction is incomplete）．

```
definition Unit :: 'at Clause \(\Rightarrow\) bool
    where \((\) Unit \(C)=((\operatorname{card} C)=1)\)
```

definition unit-resolvent :: 'at BinaryRule
where (unit-resolvent P1 P2 $C)=((\exists L .(C=((P 1-\{L\}) \cup(P 2-\{$
complement $L$ \})))
$\wedge L \in P 1 \wedge($ complement $L) \in P 2) \wedge$ Unit P1)
lemma unit-resolvent-is-resolvent : less-restrictive resolvent unit-resolvent
proof (rule ccontr)
assume $\neg$ less-restrictive resolvent unit-resolvent
then obtain P1 P2 C where unit-resolvent P1 P2 C and $\neg$ resolvent P1 P2 C
and $\neg$ resolvent P2 P1 C unfolding less-restrictive-def by auto
from 〈unit-resolvent P1 P2 $C$ 〉 obtain $L$ where $L \in P 1$ and complement $L \in$
P2
and $C=(P 1-\{L\}) \cup(P 2-\{$ complement $L\})$
using unit-resolvent-def [of P1 P2 C] by auto
obtain $A$ where $L=P o s A \vee L=N e g A$ using Literal.exhaust $[o f L]$ by auto
from this and $\langle C=(P 1-\{L\}) \cup(P 2-\{$ complement $L\})\rangle$ and $\langle L \in P 1\rangle$
and <complement $L \in$ P2 〉
have resolvent P1 P2 $C \vee$ resolvent P2 P1 $C$ unfolding resolvent-def by auto
from this and $\langle\neg$ resolvent P2 P1 $C$ 〉 and $\langle\neg$ resolvent P1 P2 $C\rangle$ show False by
auto
qed

### 3.6 Positive and Negative Resolution

Resolution is applied only if one parent is positive (resp. negative). Again, relaxed ordering restrictions apply.

```
definition positive-part :: 'at Clause }=>\mathrm{ ' 'at Clause
where
    (positive-part C)}={L.(\existsA.L=PosA)\wedgeL\inC
definition negative-part :: 'at Clause => 'at Clause
where
    (negative-part C) ={L. (\existsA.L=Neg A)^L\inC }
```

lemma decomposition-clause-pos-neg :
$C=($ negative-part $C) \cup($ positive-part $C)$
proof
show $C \subseteq($ negative-part $C) \cup($ positive-part $C)$
proof
fix $x$ assume $x \in C$
obtain $A$ where $x=\operatorname{Pos} A \vee x=\operatorname{Neg} A$ using Literal.exhaust $[o f x]$ by auto
show $x \in($ negative-part $C) \cup($ positive-part $C)$
proof cases
assume $x=\operatorname{Pos} A$
from this and $\langle x \in C\rangle$ have $x \in$ positive-part $C$ unfolding positive-part-def
by auto
then show $x \in($ negative-part $C) \cup($ positive-part $C)$ by auto
next
assume $x \neq \operatorname{Pos} A$
from this and $<x=$ Pos $A \vee x=$ Neg $A\rangle$ have $x=\operatorname{Neg} A$ by auto
from this and $\langle x \in C\rangle$ have $x \in$ negative-part $C$ unfolding negative-part-def
by auto
then show $x \in($ negative-part $C) \cup($ positive-part $C)$ by auto
qed
qed
next
show (negative-part $C) \cup($ positive-part $C) \subseteq C$ unfolding negative-part-def
and positive-part-def by auto
qed
definition ordered-positive-resolvent :: 'at Clause $\Rightarrow$ 'at Clause $\Rightarrow$ 'at Clause $\Rightarrow$
bool
where
(ordered-positive-resolvent P1 P2 C) $=$
$(\exists L .(C=(P 1-\{L\} \cup(P 2-\{$ complement $L\}))) \wedge$
$(($ negative-part P1) $=\{ \} \wedge($ strictly-maximal-literal P1 L) $)$
$\wedge($ strictly-maximal-literal (negative-part P2) $($ complement $L)))$
definition ordered-negative-resolvent :: 'at Clause $\Rightarrow$ 'at Clause $\Rightarrow$ 'at Clause $\Rightarrow$
bool
where

```
(ordered-negative-resolvent P1 P2 C) =
```

    \((\exists L .(C=(P 1-\{L\} \cup(P 2-\{\) complement \(L\}))) \wedge\)
        \(((\) positive-part P1 \()=\{ \} \wedge(\) strictly-maximal-literal P1 L) \()\)
        \(\wedge(\) strictly-maximal-literal (positive-part P2) (complement L)))
    lemma positive－resolvent－is－resolvent ：less－restrictive resolvent ordered－positive－resolvent proof（rule ccontr）
assume $\neg$ less－restrictive resolvent ordered－positive－resolvent
then obtain P1 P2 $C$ where ordered－positive－resolvent P1 P2 $C$ and $\neg$ resolvent
P1 P2 C
and $\neg$ resolvent P2 P1 C unfolding less－restrictive－def by auto
from «ordered－positive－resolvent P1 P2 $C$ 〉 obtain $L$
where strictly－maximal－literal P1 L
and strictly－maximal－literal（negative－part P2）（complement L）
and $C=(P 1-\{L\}) \cup(P 2-\{$ complement $L\})$
using ordered－positive－resolvent－def［of P1 P2 C］by auto
from 〈strictly－maximal－literal P1 $L\rangle$ have $L \in P 1$ unfolding strictly－maximal－literal－def by auto
from 〈strictly－maximal－literal（negative－part P2）（complement L）〉 have（complement
L）$\in P 2$
unfolding strictly－maximal－literal－def negative－part－def by auto
obtain $A$ where $L=$ Pos $A \vee L=$ Neg $A$ using Literal．exhaust $[$ of $L]$ by auto
from this and $\langle C=(P 1-\{L\}) \cup(P 2-\{$ complement $L\})\rangle$ and $\langle L \in P 1\rangle$
and $\langle($ complement $L) \in P 2\rangle$
have resolvent P1 P2 $C \vee$ resolvent P2 P1 $C$ unfolding resolvent－def by auto
from this and $\langle\neg($ resolvent P2 P1 C）$\rangle$ and $\langle\neg($ resolvent P1 P2 $C)\rangle$ show False
by auto
qed
lemma negative－resolvent－is－resolvent ：less－restrictive resolvent ordered－negative－resolvent proof（rule ccontr）
assume $\neg$ less－restrictive resolvent ordered－negative－resolvent
then obtain P1 P2 C where（ordered－negative－resolvent P1 P2 $C$ ）and $\neg$（resolvent P1 P2 C）
and $\neg$（resolvent P2 P1 C）unfolding less－restrictive－def by auto
from＜ordered－negative－resolvent P1 P2 $C$ ¢ obtain $L$ where strictly－maximal－literal P1 L
and strictly－maximal－literal（positive－part P2）（complement L）
and $C=(P 1-\{L\}) \cup(P 2-\{$ complement $L\})$
using ordered－negative－resolvent－def $[$ of P1 P2 $C]$ by auto
from 〈strictly－maximal－literal P1 $L\rangle$ have $L \in P 1$ unfolding strictly－maximal－literal－def by auto
from 〈strictly－maximal－literal（positive－part P2）（complement L）〉 have（complement
$L) \in P 2$
unfolding strictly－maximal－literal－def positive－part－def by auto
obtain $A$ where $L=P$ os $A \vee L=N e g A$ using Literal．exhaust $[$ of $L]$ by auto
from this and $\langle C=(P 1-\{L\}) \cup(P 2-\{$ complement $L\})\rangle$ and $\langle L \in P 1\rangle$
and $\langle($ complement $L) \in P 2\rangle$
have resolvent P1 P2 $C \vee$ resolvent P2 P1 C unfolding resolvent－def by auto
from this and $\langle\neg$ resolvent P2 P1 $C\rangle$ and $\langle\neg$ resolvent P1 P2 $C\rangle$ show False by auto
qed

## 4 Redundancy Elimination Rules

We define the usual redundancy elimination rules.

```
definition tautology :: 'a Clause \(\Rightarrow\) bool
where
    \((\) tautology \(C) \equiv(\exists A .(\) Pos \(A \in C \wedge N e g A \in C))\)
definition subsumes \(::\) ' \(a\) Clause \(\Rightarrow{ }^{\prime} a\) Clause \(\Rightarrow\) bool
where
    (subsumes \(C D) \equiv(C \subseteq D)\)
definition redundant \(::\) ' \(a\) Clause \(\Rightarrow\) ' \(a\) Formula \(\Rightarrow\) bool
where
    redundant \(C S=((\) tautology \(C) \vee(\exists D .(D \in S \wedge\) subsumes \(D C)))\)
definition strictly-redundant :: 'a Clause \(\Rightarrow\) 'a Formula \(\Rightarrow\) bool
where
    strictly-redundant \(C S=((\) tautology \(C) \vee(\exists D .(D \in S \wedge(D \subset C))))\)
definition simplify :: 'at Formula \(\Rightarrow\) 'at Formula
where
    simplify \(S=\{C . C \in S \wedge \neg\) strictly-redundant \(C S\}\)
```

We first establish some basic syntactic properties.

```
lemma tautology-monotonous: (tautology C)\Longrightarrow(C\subseteqD)\Longrightarrow(tautology D)
unfolding tautology-def by auto
lemma simplify-involutive:
    shows simplify (simplify S)=( simplify S)
proof -
    show ?thesis unfolding simplify-def strictly-redundant-def by auto
qed
lemma simplify-finite:
    assumes all-fulfill finite S
    shows all-fulfill finite (simplify S)
using assms all-fulfill-def simplify-def by auto
lemma atoms-formula-simplify:
    shows atoms-formula (simplify S)\subseteqatoms-formula S
unfolding simplify-def using atoms-formula-subset by auto
lemma subsumption-preserves-redundancy :
    assumes redundant CS
```

```
    assumes subsumes C D
    shows redundant D S
using assms tautology-monotonous unfolding redundant-def subsumes-def by blast
lemma subsumption-and-max-literal :
    assumes subsumes C1 C2
    assumes strictly-maximal-literal C1 L1
    assumes strictly-maximal-literal C2 L2
    assumes A1 = atom L1
    assumes A2 = atom L2
    shows (A1 = A2) \vee (A1,A2) \in atom-ordering
proof -
    from }\langleA1=\mathrm{ atom L1` have L1 =(Pos A1) }\veeL1=(Neg A1) by (rul
atom-property)
    from <A2 = atom L2` have L2 = (Pos A2) \vee L2 = (Neg A2) by (rule
atom-property)
    from〈subsumes C1 C2` and «strictly-maximal-literal C1 L1` have L1 \inC2
        unfolding strictly-maximal-literal-def subsumes-def by auto
    from <strictly-maximal-literal C2 L2\rangle and <L1 \inC2` have L1 = L2 \vee lit-
eral-ordering L1 L2
    unfolding strictly-maximal-literal-def by auto
    thus ?thesis
    proof
        assume L1 = L2
        from \langleL1 = L2\rangle and }\langleA1=\mathrm{ atom L1〉 and }\langleA2 = atom L2\rangle show ?thesi
by auto
    next
        assume literal-ordering L1 L2
        from <literal-ordering L1 L2` and <L1 = (Pos A1) \vee L1 = (Neg A1)`
            and <L2 = (Pos A2) \veeL2 = (Neg A2 ) }
            show ?thesis by auto
    qed
qed
lemma superset-preserves-redundancy:
    assumes redundant C S
    assumes S\subseteq\mp@subsup{S}{}{\prime}
    shows redundant C S'
using assms unfolding redundant-def by blast
lemma superset-preserves-strict-redundancy:
    assumes strictly-redundant C S
    assumes S\subseteqSS
    shows strictly-redundant C SS
using assms unfolding strictly-redundant-def by blast
The following lemmas relate the above notions with that of semantic entailment and thus establish the soundness of redundancy elimination rules.
lemma tautologies-are-valid :
```

```
    assumes tautology C
    shows validate-clause I C
by (meson assms tautology-def validate-clause.simps validate-literal.simps(1)
    validate-literal.simps(2))
lemma subsumption-and-semantics :
    assumes subsumes C D
    assumes validate-clause I C
    shows validate-clause I D
using assms unfolding subsumes-def by auto
lemma redundancy-and-semantics :
    assumes redundant C S
    assumes validate-formula I S
    shows validate-clause I C
by
(meson assms redundant-def subsumption-and-semantics tautologies-are-valid vali-
date-formula.elims)
lemma redundancy-implies-entailment:
    assumes redundant C S
    shows entails S C
using assms entails-def redundancy-and-semantics by auto
lemma simplify-and-membership :
    assumes all-fulfill finite S
    assumes T = simplify S
    assumes C}\in
    shows redundant C T
proof -
    {
        fix n
        have }\forallC\mathrm{ . card C s n }\longrightarrowC\inS\longrightarrow redundant C T (is ?P n)
        proof (induction n)
            show ?P 0
            proof ((rule allI),(rule impI)+)
                            fix C assume card C \leq 0 and C \inS
                            from <card C \leq 0\rangle and \langleC GS\rangle and <all-fulfill finite S\rangle have C = {}
using card-0-eq
                    unfolding all-fulfill-def by auto
                    then have \neg strictly-redundant C S unfolding strictly-redundant-def tau-
tology-def by auto
            from this and }\langleC\inS\rangle\mathrm{ and }\langleT= simplify S〉 have C 价 using simplify-de
by auto
            from this show redundant C T unfolding redundant-def subsumes-def by
auto
            qed
        next
            fix n assume ?P n
```

```
        show ?P (Suc n)
    proof ((rule allI),(rule impI)+)
        fix C assume card C\leq(Suc n) and C\inS
        show redundant C T
        proof (rule ccontr)
            assume \negredundant C T
            from this have C\not\inT unfolding redundant-def subsumes-def by auto
            from this and }\langleT=\mathrm{ simplify }S\rangle\mathrm{ and }\langleC\inS\rangle\mathrm{ have strictly-redundant
CS
                unfolding simplify-def strictly-redundant-def by auto
            from this and «\negredundant C T〉 obtain D where D }\inS\mathrm{ and D }\subset
                unfolding redundant-def strictly-redundant-def by auto
                from}\langleD\subsetC\rangle\mathrm{ and }\langleC\inS\rangle\mathrm{ and <all-fulfill finite S> have card D<
card C
                unfolding all-fulfill-def
                using psubset-card-mono by auto
            from this and <card C \leq (Suc n)\rangle have card D\leqn by auto
            from this and 〈?P n\rangle and \langleD \inS\rangle have redundant D T by auto
            show False
            proof cases
                assume tautology D
                    from this and \langleD\subsetC\rangle have tautology C unfolding tautology-def
by auto
            then have redundant C T unfolding redundant-def by auto
                    from this and «\negredundant C T〉 show False by auto
                    next
                        assume \negtautology D
                    from this and <redundant D T〉 obtain E where E E T and E\subseteqD
                    unfolding redundant-def subsumes-def by auto
                    from this and }\langleD\subsetC\rangle\mathrm{ have }E\subseteqC\mathrm{ by auto
                    from this and }\langleE\inT\rangle\mathrm{ and <ᄀredundant C T〉 show False
                    unfolding redundant-def and subsumes-def by auto
                    qed
                    qed
            qed
        qed
    }
    from this and }\langleC\inS\rangle\mathrm{ show ?thesis by auto
qed
lemma simplify-preserves-redundancy:
    assumes all-fulfill finite S
    assumes redundant C S
    shows redundant C (simplify S)
by (meson assms redundant-def simplify-and-membership subsumption-preserves-redundancy)
lemma simplify-preserves-strict-redundancy:
    assumes all-fulfill finite S
    assumes strictly-redundant C S
```

```
    shows strictly-redundant C (simplify S)
proof ((cases tautology C),(auto simp add: strictly-redundant-def)[1])
next
    assume \negtautology C
    from this and assms(2) obtain D where D\subsetC and D ES unfolding
strictly-redundant-def by auto
    from }\langleD\inS\rangle\mathrm{ have redundant D S unfolding redundant-def subsumes-def by
auto
    from assms(1) this have redundant D (simplify S) using simplify-preserves-redundancy
by auto
    from <\negtautology C` and }\langleD\subsetC\rangle\mathrm{ have }\neg\mathrm{ tautology D unfolding tautology-def
by auto
    from this and «redundant D (simplify S)` obtain E where E E simplify S
            and subsumes E D unfolding redundant-def by auto
    from «subsumes E D` and «D\subsetC` have E\subsetC unfolding subsumes-def by
auto
    from this and }\langleE\in\mathrm{ simplify }S\rangle\mathrm{ show strictly-redundant C (simplify S)
        unfolding strictly-redundant-def by auto
qed
lemma simplify-preserves-semantic :
    assumes T= simplify S
    assumes all-fulfill finite S
    shows validate-formula IS \longleftrightarrow validate-formula I T
by (metis (mono-tags, lifting) assms mem-Collect-eq redundancy-and-semantics
simplify-and-membership
    simplify-def validate-formula.simps)
lemma simplify-preserves-equivalence :
    assumes T= simplify S
    assumes all-fulfill finite S
    shows equivalent ST
using assms equivalent-def simplify-preserves-semantic validity-implies-entailment
by auto
```

After simplification, the formula contains no strictly redundant clause:

```
definition non-redundant :: 'at Formula }=>\mathrm{ bool
    where non-redundant S = (\forallC. (C\inS\longrightarrow \negstrictly-redundant C S))
lemma simplify-non-redundant:
    shows non-redundant (simplify S)
by (simp add: non-redundant-def simplify-def strictly-redundant-def)
lemma deducible-clause-preserve-redundancy:
    assumes redundant C S
    shows redundant C (add-all-deducible-clauses R S)
using assms superset-preserves-redundancy by fastforce
```


## 5 Renaming

A renaming is a function changing the sign of some literals. We show that this operation preserves most of the previous syntactic and semantic notions.
definition rename-literal :: 'at set $\Rightarrow$ 'at Literal $\Rightarrow$ 'at Literal
where rename-literal $A L=($ if $(($ atom $L) \in A)$ then (complement $L)$ else $L$ )
definition rename-clause :: 'at set $\Rightarrow$ 'at Clause $\Rightarrow$ 'at Clause
where rename-clause $A C=\{L . \exists L L . L L \in C \wedge L=($ rename-literal $A L L)\}$
definition rename-formula :: 'at set $\Rightarrow$ 'at Formula $\Rightarrow$ 'at Formula
where rename-formula $A S=\{C . \exists C C . C C \in S \wedge C=($ rename-clause $A C C)\}$
lemma inverse-renaming : (rename-literal $A($ rename-literal $A L))=L$ proof -
obtain $A$ where at: $L=(\operatorname{Pos} A) \vee L=($ Neg $A)$ using Literal.exhaust $[$ of $L]$ by auto
from at show ?thesis unfolding rename-literal-def by auto
qed
lemma inverse-clause-renaming : (rename-clause $A($ rename-clause $A L))=L$
proof -
show ?thesis using inverse-renaming unfolding rename-clause-def by auto qed
lemma inverse-formula-renaming : rename-formula $A$ (rename-formula $A L)=L$ proof -
show ?thesis using inverse-clause-renaming unfolding rename-formula-def by auto
qed
lemma renaming-preserves-cardinality :
card $($ rename-clause $A C)=\operatorname{card} C$
proof -
have im: rename-clause $A C=($ rename-literal $A)$ ' $C$ unfolding rename-clause-def by auto
have inj-on (rename-literal A) $C$ by (metis inj-onI inverse-renaming)
from this and im show ?thesis using card-image by auto
qed
lemma renaming-preserves-literal-order :
assumes literal-ordering L1 L2
shows literal-ordering (rename-literal A L1) (rename-literal A L2)
proof -
obtain A1 where at1: L1 $=($ Pos A1 $) \vee L 1=($ Neg A1 $)$ using Literal.exhaust [of L1 ] by auto
obtain A2 where at2: L2 $=(\operatorname{Pos}$ A2) $) \vee L 2=(N e g$ A2 $)$ using Literal.exhaust [of L2 ] by auto
from assms and at1 and at2 show ?thesis unfolding rename-literal-def by

```
auto
qed
lemma inverse-renaming-preserves-literal-order :
    assumes literal-ordering (rename-literal A L1) (rename-literal A L2)
    shows literal-ordering L1 L2
by (metis assms inverse-renaming renaming-preserves-literal-order)
lemma renaming-is-injective:
    assumes rename-literal A L1 = rename-literal A L2
    shows L1 = L2
by (metis (no-types) assms inverse-renaming)
lemma renaming-preserves-strictly-maximal-literal :
    assumes strictly-maximal-literal C L
    shows strictly-maximal-literal (rename-clause A C) (rename-literal A L)
proof -
    from assms have (L\inC) and Lismax: ( }\forallB.(B\inC\wedgeL\not=B)\longrightarrow\mathrm{ (literal-ordering
B L))
    unfolding strictly-maximal-literal-def by auto
    from }\langleL\inC\rangle\mathrm{ have (rename-literal A L) 
        unfolding rename-literal-def and rename-clause-def by auto
    have
        B. (B\inrename-clause A C \longrightarrow rename-literal A L}=
            \longrightarrow ~ l i t e r a l - o r d e r i n g ~ B ~ ( r e n a m e - l i t e r a l ~ A ~ L ) ) ~
    proof (rule)+
        fix B assume B fename-clause A C and rename-literal A L\not=B
            from }\langleB\in\mathrm{ rename-clause }AC>\mathrm{ obtain }\mp@subsup{B}{}{\prime}\mathrm{ where }\mp@subsup{B}{}{\prime}\inC\mathrm{ and }B=re
name-literal A B'
            unfolding rename-clause-def by auto
        from 〈rename-literal }AL\not=B\rangle\mathrm{ and }\langleB=\mathrm{ rename-literal }A\mp@subsup{B}{}{\prime}
            have rename-literal A L}\not=\mathrm{ rename-literal A B' by auto
        hence L\not= B' by auto
        from this and }\langle\mp@subsup{B}{}{\prime}\inC\rangle\mathrm{ and Lismax have literal-ordering B' L by auto
        from this and }\langleB=(\mathrm{ rename-literal }A\mp@subsup{B}{}{\prime})
        show literal-ordering B (rename-literal A L) using renaming-preserves-literal-order
by auto
    qed
    from this and «(rename-literal A L) \in(rename-clause A C)> show ?thesis
        unfolding strictly-maximal-literal-def by auto
qed
lemma renaming-and-selected-part :
    selected-part UNIV C = rename-clause Sel (validated-part Sel (rename-clause Sel
C))
proof
    show selected-part UNIV C\subseteqrename-clause Sel (validated-part Sel (rename-clause
Sel C))
    proof
```

```
    fix }x\mathrm{ assume }x\in\mathrm{ selected-part UNIV C
    show }x\in\mathrm{ rename-clause Sel (validated-part Sel (rename-clause Sel C))
    proof -
    from <x \in selected-part UNIV C> obtain A where }x=NegA\mathrm{ and }x\in
        unfolding selected-part-def by auto
    from }\langlex\inC\rangle\mathrm{ have rename-literal Sel x f rename-clause Sel C
        unfolding rename-clause-def by blast
    show }x\in\mathrm{ rename-clause Sel (validated-part Sel (rename-clause Sel C))
    proof cases
        assume A \inSel
        from this and <x = Neg A> have rename-literal Sel x = Pos A
            unfolding rename-literal-def by auto
        from this and }\langleA\inSel\rangle have validate-literal Sel (rename-literal Sel x) by
auto
        from this and <rename-literal Sel x fename-clause Sel C>
        have rename-literal Sel x validated-part Sel (rename-clause Sel C)
                unfolding validated-part-def by auto
        thus }x\in\mathrm{ rename-clause Sel (validated-part Sel (rename-clause Sel C))
                using inverse-renaming rename-clause-def by auto
        next
        assume A}\not\inSe
        from this and }\langlex=Neg A> have rename-literal Sel x = Neg A
                unfolding rename-literal-def by auto
    from this and }\langleA\not\in\mathrm{ Sel` have validate-literal Sel (rename-literal Sel x) by
auto
        from this and <rename-literal Sel x frename-clause Sel C>
        have rename-literal Sel x validated-part Sel (rename-clause Sel C)
                unfolding validated-part-def by auto
        thus }x\in\mathrm{ rename-clause Sel (validated-part Sel (rename-clause Sel C))
            using inverse-renaming rename-clause-def by auto
        qed
        qed
    qed
    next
    show rename-clause Sel (validated-part Sel (rename-clause Sel C)) \subseteq(selected-part
UNIV C)
    proof
        fix }
        assume x f rename-clause Sel (validated-part Sel (rename-clause Sel C))
        from this obtain y where y\invalidated-part Sel (rename-clause Sel C)
            and x = rename-literal Sel y
            unfolding rename-clause-def validated-part-def by auto
        from }\langley\in\mathrm{ validated-part Sel (rename-clause Sel C)` have
        y\in rename-clause Sel C and validate-literal Sel y unfolding validated-part-def
by auto
    from }\langley\in\mathrm{ rename-clause Sel C> obtain z where z C C and y=rename-literal
Sel z
            unfolding rename-clause-def by auto
        obtain A where zA:z=Pos A\veez=Neg A using Literal.exhaust [of z] by
```

```
auto
    show }x\in\mathrm{ selected-part UNIV C
    proof cases
            assume A\inSel
            from this and zA and }\langley=\mathrm{ rename-literal Sel z〉 have }y=\mathrm{ complement z
                using rename-literal-def by auto
            from this and }\langleA\inSel\rangle and zA and «validate-literal Sel y> have y=Po
A
                and}z=NegA by aut
            from this and }\langleA\inSel\rangle and <x= rename-literal Sel y> have x=Neg 
                unfolding rename-literal-def by auto
                from this and }\langlez\inC\rangle\mathrm{ and }\langlez=Neg A\rangle show x\in selected-part UNIV C
                unfolding selected-part-def by auto
    next
                assume A}\not\inSe
                from this and zA and }\langley=\mathrm{ rename-literal Sel z> have }y=
                using rename-literal-def by auto
            from this and }\langleA\not\inSel\rangle and zA and <validate-literal Sel y> have y=Ne
                    A
                and}z=NegA by aut
            from this and }\langleA\not\inSel\rangle\mathrm{ and }\langlex=\mathrm{ rename-literal Sel }y\rangle\mathrm{ have }x=Neg 
                unfolding rename-literal-def by auto
            from this and }\langlez\inC\rangle\mathrm{ and }\langlez=Neg A\rangle show x\in selected-part UNIV C
                unfolding selected-part-def by auto
    qed
    qed
qed
lemma renaming-preserves-tautology:
    assumes tautology C
    shows tautology (rename-clause Sel C)
proof -
    from assms obtain A where Pos A\inC and Neg A\inC unfolding tautology-def
by auto
    from }\langlePos A\inC\rangle have rename-literal Sel (Pos A) \in rename-clause Sel C
    unfolding rename-clause-def by auto
    from <Neg A C C> have rename-literal Sel (Neg A) \in rename-clause Sel C
        unfolding rename-clause-def by auto
    show ?thesis
    proof cases
        assume A\inSel
        from this have rename-literal Sel (Pos A) = Neg A
            and rename-literal Sel (Neg A) = (Pos A)
            unfolding rename-literal-def by auto
    from <rename-literal Sel (Pos A) = (Neg A)〉 and <rename-literal Sel (Neg A)
= (Pos A)>
            and «rename-literal Sel (Pos A) \in (rename-clause Sel C)\rangle
            and 〈rename-literal Sel (Neg A) \in (rename-clause Sel C)〉
            show tautology (rename-clause Sel C) unfolding tautology-def by auto
```


## next

assume $A \notin S e l$
from this have rename－literal Sel $(\operatorname{Pos} A)=P o s A$ and rename－literal Sel $(\operatorname{Neg} A)=(N e g A)$
unfolding rename－literal－def by auto
from $\langle$ rename－literal Sel $(\operatorname{Pos} A)=\operatorname{Pos} A\rangle$ and $\langle$ rename－literal Sel $(\operatorname{Neg} A)=$ （Neg A）〉
and 〈rename－literal Sel $($ Pos $A) \in$ rename－clause Sel $C$ 〉
and $\langle$ rename－literal Sel $(N e g A) \in$ rename－clause Sel $C\rangle$
show tautology（rename－clause Sel C）unfolding tautology－def by auto qed
qed
lemma rename－union ：rename－clause Sel $(C \cup D)=$ rename－clause Sel $C \cup$ rename－clause Sel D
unfolding rename－clause－def by auto
lemma renaming－set－minus－subset ：
rename－clause Sel $(C-\{L\}) \subseteq$ rename－clause Sel $C-\{$ rename－literal Sel $L\}$ proof
fix $x$ assume $x \in$ rename－clause Sel $(C-\{L\})$
then obtain $y$ where $y \in C-\{L\}$ and $x=$ rename－literal Sel $y$ unfolding rename－clause－def by auto
from $\langle y \in C-\{L\}\rangle$ and $\langle x=$ rename－literal Sel $y\rangle$ have $x \in$ rename－clause Sel C
unfolding rename－clause－def by auto
have $x \neq$ rename－literal Sel L
proof
assume $x=$ rename－literal Sel L
hence rename－literal Sel $x=L$ using inverse－renaming by auto
from this and $\langle x=$ rename－literal Sel $y\rangle$ have $y=L$ using inverse－renaming
by auto
from this and $\langle y \in C-\{L\}\rangle$ show False by auto
qed
from $\langle x \neq$ rename－literal Sel $L\rangle$ and $\langle x \in$ rename－clause Sel $C\rangle$
show $x \in($ rename－clause Sel $C)-\{$ rename－literal Sel $L\}$ by auto
qed
lemma renaming－set－minus ：rename－clause Sel $(C-\{L\})$
$=($ rename－clause Sel $C)-\{$ rename－literal Sel $L\}$
proof
show rename－clause Sel $(C-\{L\}) \subseteq($ rename－clause Sel $C)-\{$ rename－literal Sel L \}
using renaming－set－minus－subset by auto
next
show（rename－clause Sel $C$ ）－\｛rename－literal Sel $L\} \subseteq$ rename－clause Sel $(C$
－\｛ L \} )
proof－
have rename－clause Sel（（rename－clause Sel C）－\｛（rename－literal Sel L）\})
$\subseteq($ rename-clause Sel $($ rename-clause Sel $C))-\{$ rename-literal Sel (rename-literal Sel L) \}
using renaming-set-minus-subset by auto
from this
have rename-clause Sel ( (rename-clause Sel C) - $\{($ rename-literal Sel L) $\})$
$\subseteq(C-\{L\})$
using inverse-renaming inverse-clause-renaming by auto
from this
have rename-clause Sel (rename-clause Sel ( (rename-clause Sel C) - \{ (rename-literal Sel L) \}))
$\subseteq($ rename-clause Sel $(C-\{L\}))$ using rename-clause-def by auto
from this
show $($ rename-clause Sel $C)-\{($ rename-literal Sel $L)\} \subseteq$ rename-clause Sel (C-\{L\})
using inverse-renaming inverse-clause-renaming by auto
qed
qed
definition rename-interpretation :: 'at set $\Rightarrow$ 'at Interpretation $\Rightarrow$ 'at Interpretation
where
rename-interpretation Sel $I=\{A .(A \in I \wedge A \notin$ Sel $)\} \cup\{A .(A \notin I \wedge A \in$ Sel) \}
lemma renaming-preserves-semantic :
assumes validate-literal I L
shows validate-literal (rename-interpretation Sel I) (rename-literal Sel L)
proof -
let $? J=$ rename-interpretation Sel $I$
obtain $A$ where $L=\operatorname{Pos} A \vee L=N e g A$ using Literal.exhaust [of $L$ ] by auto from $\langle L=P o s A \vee L=N e g A\rangle$ have atom $L=A$ by auto show ?thesis
proof cases
assume $A \in S e l$
from this and $\langle$ atom $L=A\rangle$ have rename-literal Sel $L=$ complement $L$
unfolding rename-literal-def by auto
show ?thesis
proof cases
assume $L=\operatorname{Pos} A$
from this and $\langle$ validate-literal $I L\rangle$ have $A \in I$ by auto
from this and $\langle A \in S e l\rangle$ have $A \notin ? J$ unfolding rename-interpretation-def
by blast
from this and $\langle L=P o s A\rangle$ and $\langle$ rename-literal Sel $L=$ complement $L\rangle$
show ?thesis by auto
next
assume $L \neq \operatorname{Pos} A$
from this and $\langle L=\operatorname{Pos} A \vee L=N e g A\rangle$ have $L=N e g A$ by auto
from this and «validate-literal $I L\rangle$ have $A \notin I$ by auto
from this and $\langle A \in S e l\rangle$ have $A \in ? J$ unfolding rename-interpretation-def
by blast
from this and $\langle L=N e g A\rangle$ and $\langle$ rename－literal Sel $L=$ complement $L\rangle$
show ？thesis by auto
qed
next
assume $A \notin$ Sel
from this and $\langle$ atom $L=A\rangle$ have rename－literal Sel $L=L$
unfolding rename－literal－def by auto
show ？thesis
proof cases
assume $L=\operatorname{Pos} A$
from this and $\langle$ validate－literal $I L\rangle$ have $A \in I$ by auto
from this and $\langle A \notin S e l\rangle$ have $A \in ?$ ？unfolding rename－interpretation－def by blast
from this and $\langle L=P o s A\rangle$ and $\langle r e n a m e-l i t e r a l$ Sel $L=L\rangle$ show ？thesis by auto
next
assume $L \neq \operatorname{Pos} A$
from this and $\langle L=P o s A \vee L=N e g A\rangle$ have $L=N e g A$ by auto
from this and $\langle$ validate－literal $I L\rangle$ have $A \notin I$ by auto
from this and $\langle A \notin S e l\rangle$ have $A \notin ? J$ unfolding rename－interpretation－def
by blast
from this and $\langle L=N e g A\rangle$ and $\langle r e n a m e-l i t e r a l$ Sel $L=L\rangle$ show ？thesis by auto
qed
qed
qed
lemma renaming－preserves－satisfiability：
assumes satisfiable $S$
shows satisfiable（rename－formula Sel $S$ ）
proof－
from assms obtain $I$ where validate－formula $I S$ unfolding satisfiable－def by auto
let ？$J=$ rename－interpretation Sel I
have validate－formula？（rename－formula Sel S）
proof（rule ccontr）
assume $\neg$ validate－formula ？J（rename－formula Sel S）
then obtain $C$ where $C \in S$ and $\neg$（validate－clause ？J（rename－clause Sel C））
unfolding rename－formula－def by auto
from $\langle C \in S\rangle$ and $\langle$ validate－formula $I S\rangle$ obtain $L$ where $L \in C$
and validate－literal I L by auto
from 〈validate－literal I L〉 have validate－literal ？J（rename－literal Sel L）
using renaming－preserves－semantic by auto
from this and $\langle L \in C\rangle$ and $\left\langle\neg\right.$ validate－clause？${ }^{\text {？}}$（rename－clause Sel $C$ ）〉 show False
unfolding rename－clause－def by auto
qed
from this show ?thesis unfolding satisfiable-def by auto
qed
lemma renaming-preserves-subsumption:
assumes subsumes $C D$
shows subsumes (rename-clause Sel C) (rename-clause Sel D)
using assms unfolding subsumes-def rename-clause-def by auto

## 6 Soundness

In this section we prove that all the rules introduced in the previous section are sound. We first introduce an abstract notion of soundness.
definition Sound :: 'at BinaryRule $\Rightarrow$ bool where
$($ Sound Rule $) \equiv \forall$ I P1 P2 C. (Rule P1 P2 C $\longrightarrow$ (validate-clause I P1) $\longrightarrow$ (validate-clause I P2)
$\longrightarrow$ (validate-clause I C))
lemma soundness-and-entailment :
assumes Sound Rule
assumes Rule P1 P2 C
assumes $P 1 \in S$
assumes $P \mathcal{Z} \in S$
shows entails $S C$
using Sound-def assms entails-def by auto
lemma all-deducible-sound:
assumes Sound $R$
shows entails-formula $S$ (all-deducible-clauses $R S$ )
proof (rule ccontr)
assume $\neg$ entails-formula $S$ (all-deducible-clauses $R S$ )
then obtain $C$ where $C \in$ all-deducible-clauses $R S$ and $\neg$ entails $S C$
unfolding entails-formula-def by auto
from $\langle C \in$ all-deducible-clauses $R S$ obtain P1 P2 where $R$ P1 P2 $C$ and P1 $\in S$ and $P 2 \in S$
by auto
from $\langle R$ P1 P2 $C\rangle$ and $\operatorname{assms}(1)$ and $\langle P 1 \in S\rangle$ and $\langle P 2 \in S\rangle$ and $\langle\neg$ entails $S C$ >
show False using soundness-and-entailment by auto
qed
lemma add-all-deducible-sound:
assumes Sound $R$
shows entails-formula $S$ (add-all-deducible-clauses $R S$ )
by (metis UnE add-all-deducible-clauses.simps all-deducible-sound assms entails-formula-def entails-member)

If a rule is more restrictive than a sound rule then it is necessarily sound.

```
lemma less-restrictive-correct:
    assumes less-restrictive R1 R2
    assumes Sound R1
    shows Sound R2
using assms unfolding less-restrictive-def Sound-def by blast
```

We finally establish usual concrete soundness results．

```
theorem resolution-is-correct:
    (Sound resolvent)
proof (rule ccontr)
    assume \(\neg\) (Sound resolvent)
    then obtain I P1 P2 \(C\) where
    resolvent P1 P2 C validate-clause I P1 validate-clause I P2 and \(\neg\) validate-clause
I C
    unfolding Sound-def by blast
    from 〈resolvent P1 P2 \(C\) 〉 obtain \(A\) where
        \((\operatorname{Pos} A) \in P 1\) and \((\operatorname{Neg} A) \in P 2\) and \(C=((P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{\)
\(\operatorname{Neg} A\})\) )
        unfolding resolvent-def by auto
    show False
    proof cases
            assume \(A \in I\)
            hence \(\neg\) validate-literal \(I(\operatorname{Neg} A)\) by auto
            from 〈 \(\neg\) validate-literal I (Neg A)〉 and 〈validate-clause I P2〉
                have validate-clause I (P2 - \{Neg A \}) by auto
            from «validate-clause \(I(P 2-\{N e g A\})\rangle\) and \(\langle C=((P 1-\{\operatorname{Pos} A\})\)
\(\cup(P 2-\{\operatorname{Neg} A\}))\rangle\)
            and 〈 \(\neg\) validate-clause \(I C\) 〉show False by auto
    next
            assume \(A \notin I\)
            hence \(\neg\) validate-literal I (Pos A) by auto
            from 〈 \(\neg\) validate-literal I (Pos A) > and «validate-clause I P1〉
                have validate-clause \(I(P 1-\{\operatorname{Pos} A\})\) by auto
            from \(\langle\) validate-clause \(I(P 1-\{\operatorname{Pos} A\})\rangle\) and \(\langle C=((P 1-\{\operatorname{Pos} A\})\)
\(\cup(P 2-\{\operatorname{Neg} A\}))\rangle\)
            and 〈ᄀvalidate-clause I C〉
            show False by auto
    qed
qed
```

theorem ordered－resolution－correct ：Sound ordered－resolvent
using resolution－is－correct and ordered－resolvent－is－resolvent less－restrictive－correct by auto
theorem ordered－model－resolution－correct ：Sound（ordered－model－resolvent I）
using resolution－is－correct ordered－model－resolvent－is－resolvent less－restrictive－correct by auto
theorem ordered－positive－resolution－correct ：Sound ordered－positive－resolvent
using less-restrictive-correct positive-resolvent-is-resolvent resolution-is-correct by auto
theorem ordered-negative-resolution-correct : Sound ordered-negative-resolvent using less-restrictive-correct negative-resolvent-is-resolvent resolution-is-correct by auto
theorem unit-resolvent-correct : Sound unit-resolvent
using less-restrictive-correct resolution-is-correct unit-resolvent-is-resolvent by auto

## 7 Refutational Completeness

In this section we establish the refutational completeness of the previous inference rules (under adequate restrictions for the unit resolution rule). Completeness is proven w.r.t. redundancy elimination rules, i.e., we show that every saturated unsatisfiable clause set contains the empty clause.

We first introduce an abstract notion of saturation.
definition saturated-binary-rule :: 'a BinaryRule $\Rightarrow$ 'a Formula $\Rightarrow$ bool where
(saturated-binary-rule Rule $S) \equiv(\forall$ P1 P2 $C .(((P 1 \in S) \wedge(P 2 \in S) \wedge($ Rule P1 P2 C)) )
$\longrightarrow$ redundant $C$ S)
definition Complete :: 'at BinaryRule $\Rightarrow$ bool
where
$($ Complete Rule $)=(\forall S .(($ saturated-binary-rule Rule $S) \longrightarrow($ all-fulfill finite $S)$ $\longrightarrow(\} \notin S) \longrightarrow$ satisfiable $S))$

If a set of clauses is saturated under some rule then it is necessarily saturated under more restrictive rules, which entails that if a rule is less restrictive than a complete rule then it is also complete.
lemma less-restrictive-saturated:
assumes less-restrictive R1 R2
assumes saturated-binary-rule R1 S
shows saturated-binary-rule R2 $S$
using assms unfolding less-restrictive-def Complete-def saturated-binary-rule-def by blast
lemma less-restrictive-complete:
assumes less-restrictive R1 R2
assumes Complete R2
shows Complete R1
using assms less-restrictive-saturated Complete-def by auto

### 7.1 Ordered Resolution

We define a function associating every set of clauses $S$ with a "canonic" interpretation constructed from $S$. If $S$ is saturated under ordered resolution and does not contain the empty clause then the interpretation is a model of $S$. The interpretation is defined by mean of an auxiliary function that maps every atom to a function indicating whether the atom occurs in the interpretation corresponding to a given clause set. The auxiliary function is defined by induction on the set of atoms.

```
function canonic-int-fun-ordered :: 'at \(\Rightarrow\) ('at Formula \(\Rightarrow\) bool)
where
    \((\) canonic-int-fun-ordered \(A)=\)
        \((\lambda S .(\exists C .(C \in S) \wedge(\) strictly-maximal-literal \(C(\) Pos \(A))\)
        \(\wedge(\forall B .(\) Pos \(B \in C \longrightarrow(B, A) \in\) atom-ordering \(\longrightarrow(\neg\) (canonic-int-fun-ordered
B) \(S)\) )
    \(\wedge(\forall B .(\operatorname{Neg} B \in C \longrightarrow(B, A) \in\) atom-ordering \(\longrightarrow((c a n o n i c-\) int-fun-ordered
B) \(S())\) ))
by auto
termination apply (relation atom-ordering)
by auto (simp add: atom-ordering-wf)
```

definition canonic-int-ordered $::$ 'at Formula $\Rightarrow$ 'at Interpretation
where
$($ canonic-int-ordered $S)=\{A .(($ canonic-int-fun-ordered $A) S)\}$

We first prove that the canonic interpretation validates every clause having a positive strictly maximal literal

```
lemma int-validate-cl-with-pos-max :
    assumes strictly-maximal-literal C (Pos A)
    assumes C\inS
    shows validate-clause (canonic-int-ordered S) C
proof cases
    assume c1:(\forallB.( Pos B\inC\longrightarrow(B,A)\in atom-ordering
                        \longrightarrow ( \neg ( \text { canonic-int-fun-ordered B)S)))}
    show ?thesis
    proof cases
        assume c2: ( }\forallB.(Neg B\inC\longrightarrow(B,A)\in\mathrm{ atom-ordering
                        \longrightarrow ( ( \text { canonic-int-fun-ordered B) S)))}
        have ((canonic-int-fun-ordered A)S)
        proof (rule ccontr)
            assume }\neg((\mathrm{ canonic-int-fun-ordered A)S)
            from «\neg ((canonic-int-fun-ordered A)S)`
            have e:\neg(\existsC.(C\inS)^(strictly-maximal-literal C (Pos A))
        \wedge ( \forall B . ( P o s ~ B \in C \longrightarrow ( B , A ) \in ~ a t o m - o r d e r i n g \longrightarrow ( \neg ( c a n o n i c - i n t - f u n - o r d e r e d ~
B) S)))
    \wedge ( \forall B . ( N e g B \in C \longrightarrow ( B , A ) \in a t o m - o r d e r i n g \longrightarrow ( ( c a n o n i c - i n t - f u n - o r d e r e d ~
B) S))))
            by ((simp only:canonic-int-fun-ordered.simps[of A]), blast)
```

```
    from e and c1 and c2 and «(C \inS)>and «(strictly-maximal-literal C (Pos
A))>
    show False by blast
    qed
    from «((canonic-int-fun-ordered A) S)> have A \in(canonic-int-ordered S)
        unfolding canonic-int-ordered-def by blast
        from}\langleA\in(\mathrm{ canonic-int-ordered S)> and «(strictly-maximal-literal C (Pos
A))>
        show ?thesis
        unfolding strictly-maximal-literal-def by auto
    next
        assume not-c2: }\neg(\forallB.(Neg B\inC\longrightarrow(B,A)\in\mathrm{ atom-ordering
                        \longrightarrow ( ( \text { canonic-int-fun-ordered B) S)))}
        from not-c2 obtain B where Neg B}\inC\mathrm{ and }\neg((canonic-int-fun-ordered
B) S)
        by blast
        from <\neg((canonic-int-fun-ordered B) S)> have B}\not\in(\mathrm{ canonic-int-ordered S)
            unfolding canonic-int-ordered-def by blast
        with }\langleNeg B\inC\rangle\mathrm{ show ?thesis by auto
    qed
    next
    assume not-c1: }\neg(\forallB.( Pos B\inC\longrightarrow(B,A)\in atom-ordering
                            \longrightarrow ( \neg ( \text { canonic-int-fun-ordered B)S)))}
    from not-c1 obtain B where Pos B}\inC\mathrm{ and ((canonic-int-fun-ordered B)
S)
    by blast
    from <((canonic-int-fun-ordered B) S)> have B (canonic-int-ordered S)
        unfolding canonic-int-ordered-def by blast
    with \langlePos B C C` show ?thesis by auto
qed
lemma strictly-maximal-literal-exists :
```

```
    \forallC. (((finite C)^(card C) = n ^ n = 0 ^ (\neg(tautology C))))
```

    \forallC. (((finite C)^(card C) = n ^ n = 0 ^ (\neg(tautology C))))
    \longrightarrow(\existsA.(strictly-maximal-literal C A)) (is ?P n)
    \longrightarrow(\existsA.(strictly-maximal-literal C A)) (is ?P n)
    proof (induction n)
proof (induction n)
show (?P 0) by auto
show (?P 0) by auto
next
next
fix n assume ?P n
fix n assume ?P n
show ?P (Suc n)
show ?P (Suc n)
proof
proof
fix C
fix C
show (finite C ^ card C=Suc n ^ Suc n = 0 ^ ᄀ(tautology C))
show (finite C ^ card C=Suc n ^ Suc n = 0 ^ ᄀ(tautology C))
\longrightarrow ( \exists A . ( s t r i c t l y - m a x i m a l - l i t e r a l ~ C ~ A ) )
\longrightarrow ( \exists A . ( s t r i c t l y - m a x i m a l - l i t e r a l ~ C ~ A ) )
proof
proof
assume finite C ^ card C=Suc n ^ Suc n = 0 ^ ᄀ(tautology C)
assume finite C ^ card C=Suc n ^ Suc n = 0 ^ ᄀ(tautology C)
hence (finite C) and (card C) = (Suc n) and ( }\neg(\mathrm{ tautology C)) by
hence (finite C) and (card C) = (Suc n) and ( }\neg(\mathrm{ tautology C)) by
auto

```
auto
```

```
have C\not={}
proof
    assume C={}
    from〈finite C〉 and 〈C={}〉 have card C=0 using card-0-eq by
auto
    from <card C=0\rangle and <card C=Suc n\rangle show False by auto
    qed
    then obtain L where L\inC by auto
    from «\negtautology C> have \negtautology (C - {L }) using tautol-
ogy-monotonous
    by auto
    from }\langleL\inC\rangle\mathrm{ and <finite C> have Suc (card (C - {L })) = card C
        using card-Suc-Diff1 by metis
        with <card C=Suc n\rangle have card (C-{L})=n by auto
    show \existsA.(strictly-maximal-literal C A)
    proof cases
        assume card C=1
            from this and «card C=Suc n〉 have n=0 by auto
            from this and <finite C〉 and <card (C-{L}) = n〉 have C - {
L} = {}
                using card-0-eq by auto
    from this and }<L\inC\rangle\mathrm{ show ?thesis unfolding strictly-maximal-literal-def
by auto
        next
        assume card C\not=1
            from〈finite C> have finite (C - {L }) by auto
            from 〈Suc (card (C-{L})) = card C> and 〈card C\not= 1〉
                and }\langle(\operatorname{card}(C-{L}))=n\rangle have n\not=0 by aut
            from this and<finite (C-{L})\rangle and <card (C-{L})=n>
                and «\negtautology (C - {L })> and «?P n>
            obtain A where strictly-maximal-literal (C-{L })A by metis
            show }\existsM\mathrm{ . strictly-maximal-literal C M
            proof cases
                assume (atom L, atom A)\in atom-ordering
                    from this have literal-ordering LA by auto
                    from this and <strictly-maximal-literal (C - {L }) A>
                        have strictly-maximal-literal C A
                    unfolding strictly-maximal-literal-def by blast
                    thus ?thesis by auto
                next
                assume (atom L, atom A) # atom-ordering
                    have l-cases:}L=(\mathrm{ Pos (atom L)) }\veeL=(Neg (atom L))
                        by ((rule atom-property [of (atom L)]), auto)
                    have a-cases:A=(Pos (atom A))\veeA=(Neg (atom A))
                    by ((rule atom-property [of (atom A)]), auto)
                    from l-cases and a-cases and <(strictly-maximal-literal (C - {
L }) A)>
                    and }\langle\neg(\mathrm{ tautology C)> and }\langleL\inC
```

```
            have atom L # atom A
            unfolding strictly-maximal-literal-def and tautology-def by auto
            from this and «(atom L, atom A) & atom-ordering> and
atom-ordering-total
                        have (atom A,atom L)\in atom-ordering by auto
                    hence literal-ordering A L by auto
                    from this and }\langleL\inC\rangle\mathrm{ and <strictly-maximal-literal (C - {L
}) A>
                                    and literal-ordering-trans
                        have strictly-maximal-literal C L unfolding strictly-maximal-literal-def
                        unfolding strictly-maximal-literal-def by blast
                            thus ?thesis by auto
                    qed
                qed
                qed
        qed
qed
```

We then deduce that all clauses are validated.
lemma canonic-int-validates-all-clauses :
assumes saturated-binary-rule ordered-resolvent $S$
assumes all-fulfill finite $S$
assumes $\} \notin S$
assumes $C \in S$
shows validate-clause (canonic-int-ordered $S$ ) $C$
proof cases
assume (tautology $C$ )
thus ?thesis using tautologies-are-valid $[$ of $C$ (canonic-int-ordered $S$ ) ] by auto next
assume $\neg$ tautology $C$
from 〈all-fulfill finite $S\rangle$ and $\langle C \in S\rangle$ have finite $C$ using all-fulfill-def by auto
from $\langle\} \notin S\rangle$ and $\langle C \in S\rangle$ and $\langle$ finite $C\rangle$ have card $C \neq 0$ using card-0-eq by auto
from $\langle\neg$ tautology $C$ 〉 and $\langle$ finite $C\rangle$ and $\langle$ card $C \neq 0\rangle$ obtain $L$
where strictly-maximal-literal C L using strictly-maximal-literal-exists by blast
obtain $A$ where $A=$ atom $L$ by auto
have inductive-lemma:
$\forall C L .((C \in S) \longrightarrow($ strictly-maximal-literal $C L) \longrightarrow(A=($ atom $L))$ $\longrightarrow($ validate-clause (canonic-int-ordered $S) C))($ is $(? Q A))$
proof ((rule wf-induct [of atom-ordering ?Q A]),(rule atom-ordering-wf)) next
fix $x$
assume hyp-induct: $\forall y .(y, x) \in$ atom-ordering $\longrightarrow(? Q y)$
show? $Q x$
proof (rule) + fix $C L$ assume $C \in S$ strictly-maximal-literal $C L x=($ atom $L)$

```
    show validate-clause (canonic-int-ordered S) C
    proof cases
    assume L = Pos x
    from }\langleL=Pos x\rangle and <strictly-maximal-literal C L\rangle and \langleC \inS
        show validate-clause (canonic-int-ordered S) C
        using int-validate-cl-with-pos-max by auto
    next
    assume L}L=P\mathrm{ Pos }
    have L}=(Negx) using <L\not=Pos x\rangle\langlex= atom L\rangle atom-property by
fastforce
    show (validate-clause (canonic-int-ordered S)C)
    proof (rule ccontr)
        assume \neg(validate-clause(canonic-int-ordered S)C)
        from «(L= (Neg x))\rangle and «(strictly-maximal-literal C L)\rangle
            and <( }\neg(\mathrm{ validate-clause (canonic-int-ordered S) C))>
            have }x\in\mathrm{ canonic-int-ordered S unfolding strictly-maximal-literal-def
by auto
            from <x \in canonic-int-ordered S> have (canonic-int-fun-ordered x) S
            unfolding canonic-int-ordered-def by blast
            from <(canonic-int-fun-ordered x) S`
                have (\existsC. (C\inS)^(strictly-maximal-literal C (Pos x))
    \wedge(\forallB.(Pos B CC\longrightarrow(B,x)\inatom-ordering \longrightarrow( ᄀ(canonic-int-fun-ordered
B) S)))
    \wedge(\forallB.( Neg B CC\longrightarrow(B,x)\inatom-ordering \longrightarrow((canonic-int-fun-ordered
B) S))))
            by (simp only: canonic-int-fun-ordered.simps [of x])
                            then obtain D
    where (D\inS) and (strictly-maximal-literal D (Pos x))
    and a:(\forallB.(Pos B D 
        \longrightarrow ( \neg ( \text { canonic-int-fun-ordered B)S)))}
    and b:(\forallB.(Neg B\inD\longrightarrow(B,x)\in atom-ordering
                \longrightarrow ( ( \text { canonic-int-fun-ordered B)S)))}
    by blast
    obtain R where R=(resolvent-upon D Cx) by auto
    from }\langleR=\mathrm{ resolvent-upon D C x〉 and <strictly-maximal-literal D (Pos
x)>
DCR
    unfolding strictly-maximal-literal-def using resolvent-upon-is-resolvent
by auto
from \(\langle R=\) resolvent-upon \(D C x\rangle\) and «strictly-maximal-literal \(D\) (Pos
x)>
    and <strictly-maximal-literal C L\rangle and }\langleL=Neg x
    have ordered-resolvent D C R
    using ordered-resolvent-upon-is-resolvent by auto
    have \neg validate-clause (canonic-int-ordered S)R
    proof
```

```
    assume validate-clause (canonic-int-ordered S) R
    from <validate-clause (canonic-int-ordered S) R> obtain M
            where (M\inR) and validate-literal (canonic-int-ordered S) M
            by auto
            from }\langleM\inR\rangle\mathrm{ and }<R=\mathrm{ resolvent-upon D C x>
                        have (M\in(D-{Posx}))\vee (M\in(C-{Negx})) by auto
            thus False
        proof
            assume M\in(D - { Pos x })
            show False
            proof cases
                    assume }\existsAA.M=(Pos AA
                    from this obtain AA where M=Pos AA by auto
                    from}\langleM\inD-{\operatorname{Pos}x}\rangle\mathrm{ and «strictly-maximal-literal D (Pos
x)>
                and }\langle(M=\operatorname{Pos}AA)
                            have (AA,x)\in atom-ordering unfolding strictly-maximal-literal-def
by auto
            from a and }\langle(AA,x)\in\mathrm{ atom-ordering> and }<M=(Pos AA)\rangle and
<M \in(D - { Pos x })>
            have }\neg(\mathrm{ canonic-int-fun-ordered AA) S by blast
            from <\neg(canonic-int-fun-ordered AA)S` have AA & canonic-int-ordered
S
                    unfolding canonic-int-ordered-def by blast
            from}\langleAA\not\in\mathrm{ canonic-int-ordered S` and }\langleM=Pos AA
                    and «validate-literal (canonic-int-ordered S) M>
                    show False by auto
            next
                assume }\neg(\existsAA.M=(\operatorname{Pos}AA)
                    obtain AA where M=(Pos AA)\veeM=(Neg AA) using
Literal.exhaust [of M] by auto
                            from this and }\langle\neg(\existsAA.M=(\operatorname{Pos}AA))\rangle\mathrm{ have }M=(NegAA) b
auto
                            from}<M\in(D-{\operatorname{Pos}x})\rangle\mathrm{ and <strictly-maximal-literal D (Pos
x)>
                            and }\langleM=(Neg AA)
                            have (AA,x)\in atom-ordering unfolding strictly-maximal-literal-def
by auto
                            from b and }\langle(AA,x)\in\mathrm{ atom-ordering> and }<M=(Neg AA)\rangle and
<M \in(D - { Pos x })>
                            have (canonic-int-fun-ordered AA) S by blast
    from «(canonic-int-fun-ordered AA)S` have AA\in canonic-int-ordered
S
                                    unfolding canonic-int-ordered-def by blast
                from }\langleAA\in\mathrm{ canonic-int-ordered S> and <M = (Neg AA)>
                    and<validate-literal (canonic-int-ordered S) M〉 show False by
auto
            qed
            next
```

```
            assume M\in(C-{Negx })
            from «\negvalidate-clause(canonic-int-ordered S) C` and }<M\in(C-
Neg x })>
            and<validate-literal (canonic-int-ordered S) M> show False by auto
        qed
    qed
    from «\negvalidate-clause (canonic-int-ordered S) R` have \negtautology R
        using tautologies-are-valid by auto
    from <ordered-resolvent D C R\rangle and }\langleD\inS\rangle\mathrm{ and }\langleC\inS
        and <saturated-binary-rule ordered-resolvent S`
        have redundant R S unfolding saturated-binary-rule-def by auto
    from this and «\negtautology R〉 obtain }\mp@subsup{R}{}{\prime}\mathrm{ where }\mp@subsup{R}{}{\prime}\inS\mathrm{ and subsumes
R'R
        unfolding redundant-def by auto
    from }\langleR=\mathrm{ resolvent-upon D C x〉 and <strictly-maximal-literal D (Pos
x)>
        and 〈strictly-maximal-literal C L\rangle and <L = (Neg x)\rangle
    have resolvent D CR unfolding strictly-maximal-literal-def
        using resolvent-upon-is-resolvent by auto
    from «all-fulfill finite S` and }\langleC\inS` have finite C using all-fulfill-de
by auto
    from〈all-fulfill finite S` and }\langleD\inS\rangle\mathrm{ have finite D using all-fulfill-def
by auto
    from〈finite C> and <finite D> and <(resolvent D C R)\rangle have finite R
    using resolvent-is-finite unfolding derived-clauses-are-finite-def by blast
        from 〈finite R〉 and <subsumes }\mp@subsup{R}{}{\prime}R\mathrm{ \ have finite }\mp@subsup{R}{}{\prime}\mathrm{ unfolding
subsumes-def
    using finite-subset by auto
    from }\langle\mp@subsup{R}{}{\prime}\inS\rangle\mathrm{ and }\langle{}\not\inS\rangle\mathrm{ and «(subsumes }\mp@subsup{R}{}{\prime}R)\rangle\mathrm{ have }\mp@subsup{R}{}{\prime}\not={
        unfolding subsumes-def by auto
    from〈finite }\mp@subsup{R}{}{\prime}\rangle\mathrm{ and }\langle\mp@subsup{R}{}{\prime}\not={}\rangle have card R'\not=0 using card-0-eq by
auto
    from <subsumes R' R〉 and «\negtautology R〉 have \negtautology R'
        unfolding subsumes-def
        using tautology-monotonous by auto
    from «\negtautology R'〉 and <finite R'〉 and <card R'}=0\rangle\mathrm{ obtain LR'
    where strictly-maximal-literal R' LR' using strictly-maximal-literal-exists
        by blast
    from 〈finite R〉 and 〈finite R'〉 and <card R'}=0\rangle\mathrm{ and 〈subsumes }\mp@subsup{R}{}{\prime}R
        have card R\not=0
        unfolding subsumes-def by auto
    from 〈\negtautology R\rangle and 〈finite R\rangle and <card R}\not=0\rangle\mathrm{ obtain LR
    where strictly-maximal-literal R LR using strictly-maximal-literal-exists
by blast
    obtain AR and AR' where AR=atom LR and AR' = atom LR' by
auto
    from <subsumes R' R\rangle and }\langleAR=\mathrm{ atom LR> and }\langleA\mp@subsup{R}{}{\prime}=\mathrm{ atom }L\mp@subsup{R}{}{\prime}
        and «(strictly-maximal-literal R LR)>
```

and $\left\langle\left(\right.\right.$ strictly－maximal－literal $\left.\left.R^{\prime} L R^{\prime}\right)\right\rangle$ have $\left(A R^{\prime}=A R\right) \vee\left(A R^{\prime}, A R\right)$ $\in$ atom－ordering
using subsumption－and－max－literal by auto
from $\langle R=($ resolvent－upon $D C x)\rangle$ and $\langle A R=$ atom $L R\rangle$
and 〈strictly－maximal－literal $R L R\rangle$
and «strictly－maximal－literal $D($ Pos $x)\rangle$
and 〈strictly－maximal－literal $C L\rangle$ and $\langle L=(N e g x)\rangle$
have $(A R, x) \in$ atom－ordering using resolution－and－max－literal by auto
from $\left\langle(A R, x) \in\right.$ atom－ordering and $\left\langle\left(A R^{\prime}=A R\right) \vee\left(A R^{\prime}, A R\right) \in\right.$ atom－ordering〉
have $\left(A R^{\prime}, x\right) \in$ atom－ordering using atom－ordering－trans by auto
from this and hyp－induct and $\left\langle R^{\prime} \in S\right\rangle$ and $\left\langle\right.$ strictly－maximal－literal $R^{\prime}$ $L R^{\prime}$
and $\left\langle A R^{\prime}=\right.$ atom $\left.L R^{\prime}\right\rangle$ have validate－clause（canonic－int－ordered $S$ ）
$R^{\prime}$ by auto
from this and $\left\langle\right.$ subsumes $\left.R^{\prime} R\right\rangle$ and $\langle\neg$ validate－clause（canonic－int－ordered
S）$R>$
show False using subsumption－and－semantics by blast
qed
qed
qed
qed
from inductive－lemma and $\langle C \in S\rangle$ and $\langle$ strictly－maximal－literal $C L\rangle$ and $\langle A$ $=$ atom $L>$ show ？thesis by blast
qed
theorem ordered－resolution－is－complete ：
Complete ordered－resolvent
proof（rule ccontr）
assume $\neg$ Complete ordered－resolvent
then obtain $S$ where saturated－binary－rule ordered－resolvent $S$
and all－fulfill finite $S$ and $\} \notin S$ and $\neg$ satisfiable $S$ unfolding Complete－def
by auto
have validate－formula（canonic－int－ordered $S$ ）$S$
proof（rule ccontr）
assume $\neg$ validate－formula（canonic－int－ordered $S$ ）$S$
from this obtain $C$ where $C \in S$ and $\neg$ validate－clause（canonic－int－ordered
S）$C$ by auto
from 〈saturated－binary－rule ordered－resolvent $S$ 〉 and 〈all－fulfill finite $S$ 〉 and $\langle\} \notin S\rangle$
and $\langle C \in S\rangle$ and $\langle\neg$ validate－clause（canonic－int－ordered $S$ ）$C\rangle$
show False using canonic－int－validates－all－clauses by auto
qed
from 〈validate－formula（canonic－int－ordered $S$ ）$S\rangle$ and $\neg \neg$ satisfiable $S\rangle$ show False
unfolding satisfiable－def by blast qed

## 7．2 Ordered Resolution with Selection

We now consider the case where some negative literals are considered with highest priority．The proof reuses the canonic interpretation defined in the previous section．The interpretation is constructed using only clauses with no selected literals．By the previous result，all such clauses must be satisfied． We then show that the property carries over to the clauses with non empty selected part．
definition empty－selected－part Sel $S=\{C . C \in S \wedge($ selected－part Sel $C)=\{ \}\}$
lemma saturated－ordered－sel－res－empty－sel ：
assumes saturated－binary－rule（ordered－sel－resolvent Sel）$S$
shows saturated－binary－rule ordered－resolvent（empty－selected－part Sel S）
proof－
show ？thesis
proof（rule ccontr）
assume $\neg$ saturated－binary－rule ordered－resolvent（empty－selected－part Sel S）
then obtain P1 and P2 and $C$
where P1 $\in$ empty－selected－part Sel $S$ and P2 $\in$ empty－selected－part Sel $S$
and ordered－resolvent P1 P2 C
and $\neg$ redundant $C$（empty－selected－part Sel $S$ ）
unfolding saturated－binary－rule－def by auto from «ordered－resolvent P1 P2 $C$ 〉 obtain $A$ where $C=((P 1-\{$ Pos $A\})$ $\cup(P 2-\{\operatorname{Neg} A\}))$ and strictly－maximal－literal P1（Pos A）and strictly－maximal－literal P2（Neg A）
unfolding ordered－resolvent－def by auto
from $\langle P 1 \in$ empty－selected－part Sel $S\rangle$ have selected－part Sel P1 $=\{ \}$
unfolding empty－selected－part－def by auto
from $\langle P 2 \in$ empty－selected－part Sel $S\rangle$ have selected－part Sel $P 2=\{ \}$
unfolding empty－selected－part－def by auto
from $\langle C=((P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{$ Neg $A\}))\rangle$ and $\langle$ strictly－maximal－literal P1（Pos A））
and $\langle$ strictly－maximal－literal P2（Neg A）» and «（selected－part Sel P1）$=\{ \}\rangle$
and $\langle$ selected－part Sel P2 $=\{ \}\rangle$
have ordered－sel－resolvent Sel P1 P2 C unfolding ordered－sel－resolvent－def by auto
from 〈saturated－binary－rule（ordered－sel－resolvent Sel）$S\rangle$
have $\forall P 1 P 2 C .(P 1 \in S \wedge P 2 \in S \wedge($ ordered－sel－resolvent Sel P1 P2 C）$)$
$\longrightarrow$ redundant C S
unfolding saturated－binary－rule－def by auto
from this and $\langle P 1 \in($ empty－selected－part Sel S $)\rangle$ and $\langle P 2 \in($ empty－selected－part Sel S）＞
and «ordered－sel－resolvent Sel P1 P2 $C$ 〉 have tautology $C \vee(\exists D . D \in S \wedge$ subsumes $D C$ ）
unfolding empty－selected－part－def redundant－def by auto
from this and «tautology $C \vee(\exists D . D \in S \wedge$ subsumes $D C)$ 〉 and « $\neg$ redundant $C$（empty－selected－part Sel $S$ ）〉
obtain $D$ where $D \in S$ and subsumes $D C$ and $D \notin$ empty－selected－part Sel $S$
unfolding redundant－def by auto
from $\langle D \notin$ empty－selected－part $S$ el $S\rangle$ and $\langle D \in S\rangle$ obtain $B$ where $B \in$ Sel and Neg $B \in D$
unfolding empty－selected－part－def selected－part－def by auto
from $\langle N e g B \in D\rangle$ this and $\langle$ subsumes $D C$ have Neg $B \in C$ unfolding subsumes－def by auto
from this and $\langle C=((P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{\operatorname{Neg} A\}))\rangle$ have Neg $B$ $\in(P 1 \cup P 2)$ by auto
from 〈Neg $B \in(P 1 \cup P 2)\rangle$ and $\langle P 1 \in$ empty－selected－part Sel $S\rangle$
and $\langle P 2 \in$ empty－selected－part Sel $S\rangle$ and $\langle B \in$ Sel $\rangle$ show False
unfolding empty－selected－part－def selected－part－def by auto
qed
qed
definition ordered－sel－resolvent－upon ：：＇at set $\Rightarrow$＇at Clause $\Rightarrow$＇at Clause $\Rightarrow$＇at Clause $\Rightarrow$＇at $\Rightarrow$ bool
where
ordered－sel－resolvent－upon Sel P1 P2 C A 三
$(((C=((P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{\operatorname{Neg} A\})))$
$\wedge($ strictly－maximal－literal P1 $($ Pos A $)) \wedge(($ selected－part Sel P1 $)=\{ \})$
$\wedge((($ strictly－maximal－literal P2 $($ Neg A $)) \wedge($ selected－part Sel P2 $)=\{ \})$
$\vee($ strictly－maximal－literal（selected－part Sel P2）$($ Neg A）））））
lemma ordered－sel－resolvent－upon－is－resolvent：
assumes ordered－sel－resolvent－upon Sel P1 P2 C A
shows ordered－sel－resolvent Sel P1 P2 C
using assms unfolding ordered－sel－resolvent－upon－def and ordered－sel－resolvent－def by auto
lemma resolution－decreases－selected－part：
assumes ordered－sel－resolvent－upon Sel P1 P2 C A
assumes $N e g A \in P 2$
assumes finite P1
assumes finite P2
assumes card（selected－part Sel P2）$=$ Suc $n$
shows card（selected－part Sel C）$=n$
proof－
from〈finite P2〉 have finite（selected－part Sel P2）unfolding selected－part－def by auto
from 〈card $($ selected－part Sel P2）$)=($ Suc n）$\rangle$ have card $($ selected－part Sel P2）$\neq$ 0 by auto
from this and 〈finite（selected－part Sel P2）〉 have selected－part Sel P2 $\neq\{ \}$
using card－ 0 －eq by auto
from this and＜ordered－sel－resolvent－upon Sel P1 P2 $C$ A〉 have
$C=(P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{\operatorname{Neg} A\})$
and selected－part Sel P1 $=\{ \}$ and strictly－maximal－literal（selected－part Sel P2）（Neg A）
unfolding ordered－sel－resolvent－upon－def by auto
from 〈strictly－maximal－literal（selected－part Sel P2）（Neg A）〉
have Neg $A \in$ selected－part Sel P2
unfolding strictly－maximal－literal－def by auto
from this have $A \in S e l$ unfolding selected－part－def by auto
from 〈selected－part Sel P1 $=\{ \}>$ have selected－part Sel $(P 1-\{\operatorname{Pos} A\})=\{ \}$
unfolding selected－part－def by auto
from $\langle N e g ~ A \in($ selected－part Sel P2）$\rangle$
have selected－part Sel $(P 2-\{\operatorname{Neg} A\})=($ selected－part Sel P2）$-\{\operatorname{Neg} A\}$
unfolding selected－part－def by auto
from $\langle C=((P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{\operatorname{Neg} A\}))\rangle$ have
selected－part Sel C
$=($ selected－part Sel $(P 1-\{\operatorname{Pos} A\})) \cup($ selected－part Sel $(P 2-\{\operatorname{Neg} A\}))$
unfolding selected－part－def by auto
from this and «selected－part Sel $(P 1-\{\operatorname{Pos} A\})=\{ \}$ 〉
and «selected－part Sel（P2－$\{$ Neg A\}) $=$ selected－part Sel P2－$\{\operatorname{Neg} A\}\rangle$
have selected－part Sel $C=$ selected－part Sel P2－$\{$ Neg A $\}$ by auto
from $\langle N e g A \in P 2\rangle$ and $\langle A \in S e l\rangle$ have Neg $A \in$ selected－part Sel P2
unfolding selected－part－def by auto
from this and «selected－part Sel $C=($ selected－part Sel P2）$-\{$ Neg $A\}$ 〉 and 〈finite（selected－part Sel P2）〉
have card（selected－part Sel C）＝card（selected－part Sel P2）－ 1 by auto
from this and＜card（selected－part Sel P2）＝Suc n〉 show ？thesis by auto qed
lemma canonic－int－validates－all－clauses－sel：
assumes saturated－binary－rule（ordered－sel－resolvent Sel）$S$
assumes all－fulfill finite $S$
assumes $\} \notin S$
assumes $C \in S$
shows validate－clause（canonic－int－ordered（empty－selected－part Sel S））C
proof－
let ？nat－order $=\{(x::$ nat，$y::$ nat $) . x<y\}$
let ？SE $=$ empty－selected－part Sel $S$
let ？I＝canonic－int－ordered ？SE
obtain $n$ where $n=$ card（selected－part Sel C）by auto
have $\forall C$ ．card（selected－part Sel $C$ ）$=n \longrightarrow C \in S \longrightarrow$ validate－clause ？I $C$（is ？P n）
proof（（rule wf－induct［of ？nat－order ？P n］），（simp add：wf））
next
fix $n$ assume ind－hyp：$\forall m .(m, n) \in$ ？nat－order $\longrightarrow(? P m)$
show（？P $n$ ）
proof（rule＋）
fix $C$ assume card（selected－part Sel $C$ ）$=n$ and $C \in S$
from 〈all－fulfill finite $S\rangle$ and $\langle C \in S\rangle$ have finite $C$ unfolding all－fulfill－def by auto
from this have finite（selected－part Sel C）unfolding selected－part－def by auto
show validate－clause ？I C
proof（rule nat．exhaust［of n］）
assume $n=0$
from this and $\langle$ card（selected－part Sel $C$ ）$=n\rangle$ and $\langle$ finite（selected－part Sel C）＞
have selected－part Sel $C=\{ \}$ by auto
from 〈saturated－binary－rule（ordered－sel－resolvent Sel）S〉
have saturated－binary－rule ordered－resolvent？？SE
using saturated－ordered－sel－res－empty－sel by auto
from $\langle\} \notin S\rangle$ have $\} \notin ? S E$ unfolding empty－selected－part－def by auto
from 〈selected－part Sel $C=\{ \}\rangle\langle C \in S\rangle$ have $C \in$ ？SE unfolding empty－selected－part－def
by auto
from 〈all－fulfill finite $S$ 〉 have all－fulfill finite ？SE
unfolding empty－selected－part－def all－fulfill－def by auto
from this and «saturated－binary－rule ordered－resolvent ？SE〉 and $\langle\} \notin$ ？SE $\rangle$ and $\langle C \in$ ？SE $\rangle$
show validate－clause ？I $C$ using canonic－int－validates－all－clauses by auto next
fix $m$ assume $n=$ Suc $m$
from this and $\langle$ card（selected－part Sel $C$ ）$=n\rangle$ have selected－part Sel $C \neq$ \｛\} by auto
show validate－clause ？I C
proof（rule ccontr）
assume $\neg$ validate－clause ？I $C$
show False
proof（cases）
assume tautology $C$
from 〈tautology $C\rangle$ and $\langle\neg$ validate－clause ？I $C\rangle$ show False using tautologies－are－valid by auto
next
assume $\neg$（tautology $C$ ）
hence $\neg$（tautology（selected－part Sel C））
unfolding selected－part－def tautology－def by auto
from 〈selected－part Sel $C \neq\{ \}\rangle$ and 〈finite（selected－part Sel C）〉 have card（selected－part Sel $C$ ）$\neq 0$ by auto
from this and $\langle\neg($ tautology（selected－part Sel C））$\rangle$ and $\langle$ finite（selected－part Sel C）＞
obtain $L$ where strictly－maximal－literal（selected－part Sel C）L
using strictly－maximal－literal－exists［of card（selected－part Sel C）］by
blast
from 〈strictly－maximal－literal（selected－part Sel C）L〉 have $L \in$ （selected－part Sel C） and $L \in C$ unfolding strictly－maximal－literal－def selected－part－def by auto
from this and $\langle\neg$ validate－clause ？I $C$ 〉 have $\neg($ validate－literal ？I $L)$ by
from $\langle L \in($ selected－part Sel $C)\rangle$ obtain $A$ where $L=(N e g A)$ and $A$
$\in S e l$
unfolding selected－part－def by auto
from $\langle\neg($ validate－literal ？I $L)\rangle$ and $\langle L=($ Neg $A)\rangle$ have $A \in ? I$ by auto
from this have（（canonic－int－fun－ordered A）？SE）unfolding canonic－int－ordered－def
by blast
have $((\exists C .(C \in$ ？SE $) \wedge($ strictly－maximal－literal $C(\operatorname{Pos} A))$
$\wedge(\forall B .(\operatorname{Pos} B \in C \longrightarrow(B, A) \in$ atom－ordering $\longrightarrow(\neg($ canonic－int－fun－ordered $B)$ ？$S E)))$
$\wedge(\forall B .(\operatorname{Neg} B \in C \longrightarrow(B, A) \in$ atom－ordering $\longrightarrow(($ canonic－int－fun－ordered B）？SE）））））（is ？exp）
proof（rule ccontr）
assume $\neg$ ？exp
from this have $\neg(($ canonic－int－fun－ordered $A)$ ？SE $)$ by（（simp only：canonic－int－fun－ordered．simps［of A］），blast）
from this and 〈（canonic－int－fun－ordered $A)$ ？SE〉 show False by blast
qed
then obtain $D$ where
$D \in ? S E$ and strictly－maximal－literal $D(\operatorname{Pos} A)$
and c1：$(\forall B$ ．（Pos $B \in D \longrightarrow(B, A) \in$ atom－ordering $\longrightarrow(\neg($ canonic－int－fun－ordered $B)$ ？SE $)))$
and c2：$(\forall B .(\operatorname{Neg} B \in D \longrightarrow(B, A) \in$ atom－ordering $\longrightarrow(($ canonic－int－fun－ordered B）？SE））$)$
by blast
from $\langle D \in$ ？SE have（selected－part Sel $D)=\{ \}$ and $D \in S$
unfolding empty－selected－part－def by auto
from $\langle D \in$ ？SE $\rangle$ and＜all－fulfill finite $S\rangle$ have finite $D$
unfolding empty－selected－part－def all－fulfill－def by auto
let $? R=(D-\{\operatorname{Pos} A\}) \cup(C-\{\operatorname{Neg} A\})$
from 〈strictly－maximal－literal $D($ Pos $A)\rangle$
and $\langle$ strictly－maximal－literal（selected－part Sel C）L〉
and $\langle L=(\operatorname{Neg} A)\rangle$ and $\langle($ selected－part Sel $D)=\{ \}\rangle$
have（ordered－sel－resolvent－upon Sel D C？R A）
unfolding ordered－sel－resolvent－upon－def by auto
from this have ordered－sel－resolvent Sel D C？R
by（rule ordered－sel－resolvent－upon－is－resolvent）
from «（ordered－sel－resolvent－upon Sel D $C$ ？$R$ A $)\rangle\langle($ card（selected－part
Sel $C))={ }_{n}$ 〉
and $\langle n=S u c m\rangle$ and $\langle L \in C\rangle$ and $\langle L=($ Neg $A)\rangle$ and $\langle$ finite $D\rangle$ and 〈finite $C$ 〉
have card（selected－part Sel ？R）$=m$
using resolution－decreases－selected－part by auto
from＜ordered－sel－resolvent Sel $D C ? R\rangle$ and $\langle D \in S\rangle$ and $\langle C \in S\rangle$
and «saturated－binary－rule（ordered－sel－resolvent Sel）S〉
have redundant ？$R S$ unfolding saturated－binary－rule－def by auto
hence tautology ？$R \vee(\exists R R .(R R \in S \wedge($ subsumes $R R$ ？$R)))$
unfolding redundant－def by auto
hence validate－clause ？I ？R
proof
assume tautology？R
thus validate－clause ？I ？R by（rule tautologies－are－valid）

```
    next
    assume }\exists\mp@subsup{R}{}{\prime}.\mp@subsup{R}{}{\prime}\inS\wedge(\mathrm{ subsumes }\mp@subsup{R}{}{\prime}\mathrm{ ? R)
    then obtain }\mp@subsup{R}{}{\prime}\mathrm{ where }\mp@subsup{R}{}{\prime}\inS\mathrm{ and subsumes }\mp@subsup{R}{}{\prime}?R\mathrm{ by auto
    from〈finite C>and<finite D> have finite ?R by auto
    from this have finite (selected-part Sel ?R) unfolding selected-part-def
by auto
    from <subsumes R'?R` have selected-part Sel R'\subseteq selected-part Sel
?R
    unfolding selected-part-def and subsumes-def by auto
    from this and <finite (selected-part Sel ?R)>
                                have card (selected-part Sel R')}\leq\mathrm{ card (selected-part Sel ?R)
                            using card-mono by auto
                            from this and <card (selected-part Sel ?R) =m` and <n=Suc m>
                        have card (selected-part Sel R') < n by auto
                            from this and ind-hyp and }\langle\mp@subsup{R}{}{\prime}\inS\rangle\mathrm{ have validate-clause ?I R' by
auto
    from this and <subsumes R' ?R` show validate-clause ?I ?R
    using subsumption-and-semantics [of R' ?R ?I] by auto
qed
from this obtain }\mp@subsup{L}{}{\prime}\mathrm{ where }\mp@subsup{L}{}{\prime}\in?R\mathrm{ and validate-literal ?I L' by auto
    have L'}\mp@subsup{L}{}{\prime}\not\inD-{\operatorname{Pos}A
    proof
    assume L'}\mp@subsup{L}{}{\prime
    from this have }\mp@subsup{L}{}{\prime}\inD\mathrm{ by auto
    let ?A' = atom }\mp@subsup{L}{}{\prime
```



```
L\ by auto
    thus False
    proof
    assume L' = (Pos ? A')
    from this and <strictly-maximal-literal D (Pos A)\rangle and }\langle\mp@subsup{L}{}{\prime}\inD
{Pos A }>
    have (?A',A) \in atom-ordering unfolding strictly-maximal-literal-def
by auto
    from c1
    have c1': Pos ?'A'\inD\longrightarrow(?A', A)\in atom-ordering
                                    \longrightarrow ( \neg ( \text { canonic-int-fun-ordered ?A') ?SE }
        by blast
    from }\langle\mp@subsup{L}{}{\prime}\inD\rangle\mathrm{ and }\langle\mp@subsup{L}{}{\prime}=Pos ? A'> have Pos ? A ' ' A D by aut
    from c1' and }\langlePos?\mp@subsup{A}{}{\prime}\inD\rangle\mathrm{ and }\langle(?\mp@subsup{A}{}{\prime},A)\in\mathrm{ atom-ordering>
    have }\neg(\mathrm{ canonic-int-fun-ordered ?A') ?SE by blast
                            from this have ?A' & ?I unfolding canonic-int-ordered-def by
blast
            from this have }\neg(validate-literal ?I (Pos ?A')) by aut
                            from this and }\langle\mp@subsup{L}{}{\prime}=Pos ?A'\rangle and <validate-literal ?I L'\rangle show
False by auto
    next
        assume L' = Neg ? A'
        from this and <strictly-maximal-literal D (Pos A)\rangle and }\langle\mp@subsup{L}{}{\prime}\inD
```

$\{\operatorname{Pos} A\}>$
have $\left(? A^{\prime}, A\right) \in$ atom－ordering unfolding strictly－maximal－literal－def
by auto

```
from \(c 2\)
    have \(c 2^{\prime}: N e g ? A^{\prime} \in D \longrightarrow\left(? A^{\prime}, A\right) \in\) atom-ordering
```

                            \(\longrightarrow\left(\right.\) canonic-int-fun-ordered ? \(A^{\prime}\) ) ?SE
    by blast
    from \(\left\langle L^{\prime} \in D\right\rangle\) and \(\left\langle L^{\prime}=\left(N e g ? A^{\prime}\right)\right\rangle\) have \(N e g ? A^{\prime} \in D\) by auto
    from \(c 2^{\prime}\) and \(\left\langle N e g ? A^{\prime} \in D\right\rangle\) and \(\left\langle\left(? A^{\prime}, A\right) \in\right.\) atom-ordering \(\rangle\)
    have (canonic-int-fun-ordered ? \(A^{\prime}\) ) ?SE by blast
    from this have ? \(A^{\prime} \in\) ?I unfolding canonic-int-ordered-def by
    blast
from this have $\neg$ validate-literal ?I (Neg ? $A^{\prime}$ ) by auto
from this and $\left\langle L^{\prime}=N e g ? A^{\prime}\right\rangle$ and $\left\langle\right.$ validate-literal ?I $\left.L^{\prime}\right\rangle$ show
False by auto
qed
qed
from this and $\left\langle L^{\prime} \in ? R\right\rangle$ have $L^{\prime} \in C$ by auto
from this and 〈validate-literal ?I $\left.L^{\prime}\right\rangle$ and 〈ᄀvalidate-clause ?I $C$ 〉 show
False by auto
qed
qed
qed
qed
qed
from 〈?P $n\rangle$ and $\langle n=$ card (selected-part Sel $C$ )〉 and $\langle C \in S\rangle$ show ?thesis by
auto
qed
theorem ordered-resolution-is-complete-ordered-sel :
Complete (ordered-sel-resolvent Sel)
proof (rule ccontr)
assume $\neg$ Complete (ordered-sel-resolvent Sel)
then obtain $S$ where saturated-binary-rule (ordered-sel-resolvent Sel) $S$
and all-fulfill finite $S$
and $\} \notin S$
and $\neg$ satisfiable $S$ unfolding Complete-def by auto
let ?SE = empty-selected-part Sel $S$
let ?I = canonic-int-ordered ?SE
have validate-formula ?I $S$
proof (rule ccontr)
assume $\neg$ (validate-formula ?I $S$ )
from this obtain $C$ where $C \in S$ and $\neg$ (validate-clause ?I $C)$ by auto
from 〈saturated-binary-rule (ordered-sel-resolvent Sel) $S\rangle$ and «all-fulfill finite
$S$ 〉
and $\langle\} \notin S\rangle$ and $\langle C \in S\rangle$ and $\langle\neg($ validate-clause ?I $C)\rangle$
show False using canonic-int-validates-all-clauses-sel [of Sel S C] by auto
qed
from $\langle($ validate-formula ?I $S)\rangle$ and $\langle\neg($ satisfiable $S)\rangle$ show False
unfolding satisfiable-def by blast
qed

### 7.3 Semantic Resolution

We show that under some particular renaming, model resolution simulates ordered resolution where all negative literals are selected, which immediately entails the refutational completeness of model resolution.

```
lemma ordered-res-with-selection-is-model-res:
    assumes ordered-sel-resolvent UNIV P1 P2 C
    shows ordered-model-resolvent Sel (rename-clause Sel P1) (rename-clause Sel
P2)
        (rename-clause Sel C)
proof -
    from assms obtain A
    where c-def:C=((P1 - {Pos A }) \cup (P2 - {Neg A }))
        and selected-part UNIV P1 = {}
        and strictly-maximal-literal P1 (Pos A)
        and disj:((strictly-maximal-literal P2 (Neg A)) ^(selected-part UNIV P2) =
{})
        \checkmark ~ s t r i c t l y - m a x i m a l - l i t e r a l ~ ( s e l e c t e d - p a r t ~ U N I V ~ P 2 ) ~ ( N e g ~ A ) ~
    unfolding ordered-sel-resolvent-def by blast
    have rename-clause Sel ((P1 - {Pos A }) \cup (P2 - {Neg A }))
        =(rename-clause Sel (P1 - {Pos A })) \cup rename-clause Sel (P2 - {(Neg
A) })
    using rename-union [of Sel P1 - {Pos A } P2 - {Neg A }] by auto
    from this and c-def have ren-c:(rename-clause Sel C)=
        (rename-clause Sel (P1 - {Pos A })) U rename-clause Sel (P2 - {(Neg A)
}) by auto
        have m1:(rename-clause Sel (P1 - { Pos A })) =(rename-clause Sel P1)
            - { rename-literal Sel (Pos A) }
        using renaming-set-minus by auto
    have m2: rename-clause Sel (P2 - {Neg A }) = (rename-clause Sel P2)
            - {rename-literal Sel (Neg A) }
        using renaming-set-minus by auto
    from m1 and m2 and ren-c have
    rc-def:(rename-clause Sel C)=
        ((rename-clause Sel P1) - { rename-literal Sel (Pos A) })
        \cup((rename-clause Sel P2) - { rename-literal Sel (Neg A) })
    by auto
    have }\neg((\mathrm{ strictly-maximal-literal P2 (Neg A)) ^(selected-part UNIV P2) ={})
    proof
    assume (strictly-maximal-literal P2 (Neg A)) ^(selected-part UNIV P2) ={}
    from this have strictly-maximal-literal P2 (Neg A) and selected-part UNIV P2
= {} by auto
    from <strictly-maximal-literal P2 (Neg A)> have Neg A EP2
            unfolding strictly-maximal-literal-def by auto
        from this and <selected-part UNIV P2 = {}` show False unfolding se-
lected-part-def by auto
```

qed
from this and disj have strictly－maximal－literal（selected－part UNIV P2）（Neg A）by auto
from this have strictly－maximal－literal（rename－clause Sel（validated－part Sel （rename－clause Sel P2）））（Neg A）
using renaming－and－selected－part by auto
from this have
strictly－maximal－literal（rename－clause Sel（rename－clause Sel（validated－part Sel（rename－clause Sel P2））））
（rename－literal Sel（Neg A））using renaming－preserves－strictly－maximal－literal
by auto
from this have
p1：strictly－maximal－literal（validated－part Sel（rename－clause Sel P2））
（rename－literal Sel（Neg A））
using inverse－clause－renaming by auto
from 〈strictly－maximal－literal P1（Pos A）〉
have p2：strictly－maximal－literal（rename－clause Sel P1）（rename－literal Sel（Pos A））
using renaming－preserves－strictly－maximal－literal by auto
from 〈（selected－part UNIV P1）$=\{ \}\rangle$ have rename－clause Sel（validated－part Sel（rename－clause Sel P1））$=\{ \}$ using renaming－and－selected－part by auto
from this have $q$ ：validated－part Sel（rename－clause Sel P1）$=\{ \}$ unfolding rename－clause－def by auto
have $r$ ：rename－literal Sel $(\operatorname{Neg} A)=$ complement（rename－literal Sel（Pos A）） unfolding rename－literal－def by auto
from $r$ and $q$ and $p 1$ and $p 2$ and $r c$－def show
ordered－model－resolvent Sel（rename－clause Sel P1）（rename－clause Sel P2）（rename－clause
Sel C）
using ordered－model－resolvent－def［of Sel rename－clause Sel P1 rename－clause
Sel P2
rename－clause Sel C］by auto
qed
theorem ordered－resolution－is－complete－model－resolution：
Complete（ordered－model－resolvent Sel）
proof（rule ccontr）
assume $\neg$ Complete（ordered－model－resolvent Sel）
then obtain $S$ where saturated－binary－rule（ordered－model－resolvent Sel）$S$
and $\} \notin S$ and all－fulfill finite $S$ and $\neg($ satisfiable $S)$ unfolding Complete－def by auto
let $? S^{\prime}=$ rename－formula Sel $S$
have $\left\} \notin ? S^{\prime}\right.$
proof
assume $\left\} \in ? S^{\prime}\right.$
then obtain $V$ where $V \in S$ and rename－clause Sel $V=\{ \}$ unfolding rename－formula－def by auto
from 〈rename－clause Sel $V=\{ \}\rangle$ have $V=\{ \}$ unfolding rename－clause－def by auto
from this and $\langle V \in S\rangle$ and $\langle\} \notin S\rangle$ show False by auto qed
from «all－fulfill finite $S$ 〉 have all－fulfill finite ？$S^{\prime}$
unfolding all－fulfill－def rename－formula－def rename－clause－def by auto
have saturated－binary－rule（ordered－sel－resolvent UNIV）？S＇
proof（rule ccontr）
assume $\neg$（saturated－binary－rule（ordered－sel－resolvent UNIV）？$S^{\prime}$ ）
from this obtain $P 1$ and $P 2$ and $C$ where $P 1 \in ? S^{\prime}$ and $P 2 \in ? S^{\prime}$
and ordered－sel－resolvent UNIV P1 P2 $C$ and $\neg$ tautology $C$
and not－subsumed：$\forall D .\left(D \in\right.$ ？$S^{\prime} \longrightarrow \neg$ subsumes $\left.D C\right)$
unfolding saturated－binary－rule－def redundant－def by auto
from 〈ordered－sel－resolvent UNIV P1 P2 C＞
have ord－ren：ordered－model－resolvent Sel（rename－clause Sel P1）（rename－clause Sel P2）
（rename－clause Sel C）
using ordered－res－with－selection－is－model－res by auto
have $\neg$ tautology（rename－clause Sel C）
using renaming－preserves－tautology inverse－clause－renaming
by（metis $\prec \neg$ tautology $C>$ inverse－clause－renaming renaming－preserves－tautology）
from $\left\langle P 1 \in ? S^{\prime}\right\rangle$ have rename－clause Sel P1 $\in$ rename－formula Sel ？$S^{\prime}$ unfolding rename－formula－def by auto
hence rename－clause Sel P1 $\in S$ using inverse－formula－renaming by auto
from $\left\langle P 2 \in ? S^{\prime}\right\rangle$ have rename－clause Sel P2 $\in$ rename－formula Sel ？$S^{\prime}$
unfolding rename－formula－def by auto
hence rename－clause Sel $P 2 \in S$ using inverse－formula－renaming by auto
from 〈ᄀtautology（rename－clause Sel C）〉 and ord－ren
and «saturated－binary－rule（ordered－model－resolvent Sel）S〉
and 〈rename－clause Sel P1 $\in S$ 〉 and 〈rename－clause Sel P2 $\in S$ 〉
obtain $D^{\prime}$ where $D^{\prime} \in S$ and subsumes $D^{\prime}$（rename－clause Sel C）
unfolding saturated－binary－rule－def redundant－def by blast
from 〈subsumes $D^{\prime}($ rename－clause Sel C）〉
have subsumes（rename－clause Sel $D^{\prime}$ ）（rename－clause Sel（rename－clause Sel
C））
using renaming－preserves－subsumption by auto
hence subsumes（rename－clause Sel $D^{\prime}$ ）$C$ using inverse－clause－renaming by auto
from $\left\langle D^{\prime} \in S\right\rangle$ have rename－clause Sel $D^{\prime} \in ? S^{\prime}$ unfolding rename－formula－def by auto
from this and not－subsumed and $\left\langle\right.$ subsumes（rename－clause Sel $D^{\prime}$ ）C〉show False by auto
qed
from this and $\left\langle\left\} \notin ? S^{\prime}\right\rangle\right.$ and $\left\langle\right.$ all－fulfill finite $\left.? S^{\prime}\right\rangle$ have satisfiable ？$S^{\prime}$
using ordered－resolution－is－complete－ordered－sel unfolding Complete－def by auto
hence satisfiable（rename－formula Sel ？$S^{\prime}$ ）using renaming－preserves－satisfiability by auto
from this and $\neg \neg$ satisfiable $S\rangle$ show False using inverse－formula－renaming by auto
qed

### 7.4 Positive and Negative Resolution

We show that positive and negative resolution simulate model resolution with some specific interpretation. Then completeness follows from previous results.
lemma empty-interpretation-validate :
validate-literal $\} L=(\exists A .(L=N e g A))$
by (meson empty-iff validate-literal.elims(2) validate-literal.simps(2))
lemma universal-interpretation-validate :
validate-literal UNIV $L=(\exists A .(L=\operatorname{Pos} A))$
by (meson UNIV-I validate-literal.elims(2) validate-literal.simps(1))
lemma negative-part-lemma:
$($ negative-part $C)=($ validated-part $\{ \} C)$
unfolding negative-part-def validated-part-def using empty-interpretation-validate by blast
lemma positive-part-lemma:
(positive-part $C)=($ validated-part UNIV C)
unfolding positive-part-def validated-part-def using universal-interpretation-validate by blast
lemma negative-resolvent-is-model-res:
less-restrictive ordered-negative-resolvent (ordered-model-resolvent UNIV)
unfolding ordered-negative-resolvent-def ordered-model-resolvent-def less-restrictive-def
using positive-part-lemma by auto
lemma positive-resolvent-is-model-res:
less-restrictive ordered-positive-resolvent (ordered-model-resolvent \{\})
unfolding ordered-positive-resolvent-def ordered-model-resolvent-def less-restrictive-def
using negative-part-lemma by auto
theorem ordered-positive-resolvent-is-complete : Complete ordered-positive-resolvent using ordered-resolution-is-complete-model-resolution less-restrictive-complete pos-itive-resolvent-is-model-res by auto
theorem ordered-negative-resolvent-is-complete: Complete ordered-negative-resolvent using ordered-resolution-is-complete-model-resolution less-restrictive-complete neg-ative-resolvent-is-model-res by auto

### 7.5 Unit Resolution and Horn Renamable Clauses

Unit resolution is complete if the considered clause set can be transformed into a Horn clause set by renaming. This result is proven by showing that unit resolution simulates semantic resolution for Horn-renamable clauses (for
some specific interpretation）．

```
definition Horn :: 'at Clause \(\Rightarrow\) bool
    where \((\) Horn \(C)=((\operatorname{card}(\) positive-part \(C)) \leq 1)\)
```

definition Horn-renamable-formula :: 'at Formula $\Rightarrow$ bool
where Horn-renamable-formula $S=(\exists$ I. (all-fulfill Horn $($ rename-formula $I S)))$
theorem unit-resolvent-complete-for-Horn-renamable-set:
assumes saturated-binary-rule unit-resolvent $S$
assumes all-fulfill finite $S$
assumes $\} \notin S$
assumes Horn-renamable-formula $S$
shows satisfiable $S$
proof -
from $«$ Horn-renamable-formula $S\rangle$ obtain $I$ where all-fulfill Horn (rename-formula
IS)
unfolding Horn-renamable-formula-def by auto
have saturated-binary-rule (ordered-model-resolvent I) $S$
proof (rule ccontr)
assume $\neg$ saturated-binary-rule (ordered-model-resolvent I) $S$
then obtain P1 P2 C where ordered-model-resolvent I P1 P2 C and P1 $\in S$
and $P 2 \in S$
and $\neg$ redundant $C S$
unfolding saturated-binary-rule-def by auto
from «ordered-model-resolvent I P1 P2 C» obtain $L$
where def-c: $C=((P 1-\{L\}) \cup(P 2-\{($ complement $L)\}))$
and strictly-maximal-literal P1 L and validated-part I P1 = \{\}
and strictly-maximal-literal (validated-part I P2) (complement L)
unfolding ordered-model-resolvent-def by auto
from 〈strictly-maximal-literal P1 $L\rangle$ have $L \in P 1$
unfolding strictly-maximal-literal-def by auto
from 〈strictly-maximal-literal (validated-part I P2) (complement L) 〉 have com-
plement $L \in P 2$
unfolding strictly-maximal-literal-def validated-part-def by auto
have selected-part UNIV (rename-clause I P1)
$=$ rename-clause I (validated-part I (rename-clause I (rename-clause I P1)))
using renaming-and-selected-part [of rename-clause I P1 I] by auto
then have selected-part UNIV (rename-clause I P1) = rename-clause $I$
(validated-part I P1)
using inverse-clause-renaming by auto
from this and $\langle$ validated-part I P1 $=\{ \}$ • have selected-part UNIV (rename-clause
I P1) $=\{ \}$
unfolding rename-clause-def by auto
then have negative-part (rename-clause I P1) $=\{ \}$
unfolding selected-part-def negative-part-def by auto
from «all-fulfill Horn (rename-formula I $S$ ) $\rangle$ and $\langle P 1 \in S\rangle$ have Horn (rename-clause
I P1)
unfolding all-fulfill-def and rename-formula-def by auto
then have card（positive－part（rename－clause I P1）） 1 unfolding Horn－def by auto
from 〈negative－part（rename－clause I P1）＝\｛\}〉
have rename－clause I P1 $=($ positive－part（rename－clause I P1））
using decomposition－clause－pos－neg by auto
from this and 〈card（positive－part（rename－clause I P1））$\leq 1$ 〉
have card（rename－clause I P1）$\leq 1$ by auto
from 〈strictly－maximal－literal P1 L〉 have P1 $\neq\{ \}$
unfolding strictly－maximal－literal－def by auto
then have rename－clause I P1 $\neq\{ \}$ unfolding rename－clause－def by auto
from 〈all－fulfill finite $S\rangle$ and $\langle P 1 \in S\rangle$ have finite P1 unfolding all－fulfill－def by auto
then have finite（rename－clause I P1）unfolding rename－clause－def by auto
from this and 〈rename－clause I P1 $\neq\{ \}\rangle$ have card（rename－clause I P1）$\neq 0$
using card－ $0-e q$ by auto
from this and 〈card（rename－clause I P1）$\leq 1\rangle$ have card（rename－clause I $P 1)=1$ by auto
then have card P1＝ 1 using renaming－preserves－cardinality by auto
then have Unit P1 unfolding Unit－def using card－image by auto
from this and $\langle L \in P 1\rangle$ and $\langle$ complement $L \in P 2\rangle$ and def－c have unit－resolvent P1 P2 $C$
unfolding unit－resolvent－def by auto
from this and $\langle\neg($ redundant $C S)\rangle$ and $\langle P 1 \in S\rangle$ and $\langle P 2 \in S\rangle$
and «saturated－binary－rule unit－resolvent $S$ 〉
show False unfolding saturated－binary－rule－def by auto
qed
from this and «all－fulfill finite $S\rangle$ and $\langle\} \notin S\rangle$ show ？thesis
using ordered－resolution－is－complete－model－resolution unfolding Complete－def by auto
qed

## 8 Computation of Saturated Clause Sets

We now provide a concrete（rather straightforward）procedure for computing saturated clause sets．Starting from the initial set，we define a sequence of clause sets，where each set is obtained from the previous one by applying the resolution rule in a systematic way，followed by redundancy elimination rules．The algorithm is generic，in the sense that it applies to any binary inference rule．
fun inferred－clause－sets ：：＇at BinaryRule $\Rightarrow$＇at Formula $\Rightarrow$ nat $\Rightarrow{ }^{\prime}$＇at Formula where
$($ inferred－clause－sets $R$ S 0$)=($ simplify $S) \mid$
（inferred－clause－sets $R S($ Suc $N)$ ）$=$
（simplify（add－all－deducible－clauses $R$（inferred－clause－sets $R S N)$ ）
The saturated set is constructed by considering the set of persistent clauses， i．e．，the clauses that are generated and never deleted．

```
fun saturation :: 'at BinaryRule = 'at Formula }=>\mathrm{ 'at Formula
    where
    saturation R S = {C.\existsN.(\forallM.(M\geqN\longrightarrowC\in inferred-clause-sets R S M))
}
```

We prove that all inference rules yield finite clauses．
theorem ordered－resolvent－is－finite ：derived－clauses－are－finite ordered－resolvent using less－restrictive－and－finite ordered－resolvent－is－resolvent resolvent－is－finite by auto
theorem model－resolvent－is－finite ：derived－clauses－are－finite（ordered－model－resolvent I） using less－restrictive－and－finite ordered－model－resolvent－is－resolvent resolvent－is－finite
by auto
theorem positive－resolvent－is－finite ：derived－clauses－are－finite ordered－positive－resolvent using less－restrictive－and－finite positive－resolvent－is－resolvent resolvent－is－finite by auto
theorem negative－resolvent－is－finite ：derived－clauses－are－finite ordered－negative－resolvent using less－restrictive－and－finite negative－resolvent－is－resolvent resolvent－is－finite by auto
theorem unit－resolvent－is－finite ：derived－clauses－are－finite unit－resolvent using less－restrictive－and－finite unit－resolvent－is－resolvent resolvent－is－finite by auto
lemma all－deducible－clauses－are－finite：
assumes derived－clauses－are－finite $R$
assumes all－fulfill finite $S$
shows all－fulfill finite（all－deducible－clauses $R S$ ）
proof（rule ccontr）
assume $\neg$ all－fulfill finite（all－deducible－clauses $R S$ ）
from this obtain $C$ where $C \in$ all－deducible－clauses $R S$ and $\neg$ finite $C$ unfolding all－fulfill－def by auto
from 〈 $C \in$ all－deducible－clauses $R S\rangle$ have $\exists$ P1 P2．R P1 P2 $C \wedge P 1 \in S \wedge$ $P 2 \in S$ by auto
then obtain P1 P2 where $R P 1 P 2 C$ and $P 1 \in S$ and $P 2 \in S$ by auto
from $\langle P 1 \in S\rangle$ and $\langle$ all－fulfill finite $S\rangle$ have finite $P 1$ unfolding all－fulfill－def by auto
from $\langle P 2 \in S\rangle$ and «all－fulfill finite $S\rangle$ have finite P2 unfolding all－fulfill－def by auto
from〈finite P1〉 and〈finite P2〉 and 〈derived－clauses－are－finite $R$ 〉 and 〈 ${ }^{2}$ P1 P2 $C$ 〉 and $\neg \neg$ finite $C$ 〉 show False
unfolding derived－clauses－are－finite－def by metis
qed
This entails that all the clauses occurring in the sets in the sequence are finite．
lemma all-inferred-clause-sets-are-finite:
assumes derived-clauses-are-finite $R$
assumes all-fulfill finite $S$
shows all-fulfill finite (inferred-clause-sets $R$ S N)
proof (induction $N$ )
from assms show all-fulfill finite (inferred-clause-sets R S 0)
using simplify-finite by auto
next
fix $N$ assume all-fulfill finite (inferred-clause-sets $R S N$ )
then have all-fulfill finite (all-deducible-clauses $R$ (inferred-clause-sets $R S N$ ) using assms(1) all-deducible-clauses-are-finite [of $R$ inferred-clause-sets $R$ S N ] by auto
from this and «all-fulfill finite (inferred-clause-sets $R S N$ )〉
have all-fulfill finite (add-all-deducible-clauses $R$ (inferred-clause-sets $R S N$ )) using all-fulfill-def by auto
then show all-fulfill finite (inferred-clause-sets $R S(S u c N)$ )
using simplify-finite by auto
qed
lemma add-all-deducible-clauses-finite:
assumes derived-clauses-are-finite $R$
assumes all-fulfill finite $S$
shows all-fulfill finite (add-all-deducible-clauses $R$ (inferred-clause-sets $R S N$ )
proof -
from assms have all-fulfill finite (all-deducible-clauses $R$ (inferred-clause-sets $R$ $S N)$ )
using all-deducible-clauses-are-finite [of $R$ inferred-clause-sets $R$ S $N$ ] all-inferred-clause-sets-are-finite [of $R S N]$ by metis
then show all-fulfill finite (add-all-deducible-clauses $R$ (inferred-clause-sets $R S$ $N)$ )
using assms all-fulfill-def all-inferred-clause-sets-are-finite [of $R S N]$ by auto qed

We show that the set of redundant clauses can only increase.
lemma sequence-of-inferred-clause-sets-is-monotonous:
assumes derived-clauses-are-finite $R$
assumes all-fulfill finite $S$
shows $\forall C$. redundant $C$ (inferred-clause-sets $R S N$ )
$\longrightarrow$ redundant $C$ (inferred-clause-sets $R S(N+M:: n a t))$
proof ((induction $M$ ), auto simp del: inferred-clause-sets.simps)
fix $M C$ assume ind-hyp: $\forall C$. redundant $C$ (inferred-clause-sets $R S N$ )
$\longrightarrow$ redundant $C$ (inferred-clause-sets $R S(N+M:: n a t))$
assume redundant $C$ (inferred-clause-sets $R S N$ )
from this and ind-hyp have redundant $C$ (inferred-clause-sets $R S(N+M)$ ) by auto
then have redundant $C$ (add-all-deducible-clauses $R$ (inferred-clause-sets $R S$ $(N+M))$ )
using deducible-clause-preserve-redundancy by auto
then have all－fulfill finite（add－all－deducible－clauses $R$（inferred－clause－sets $R S$ $(N+M))$ ）
using assms add－all－deducible－clauses－finite［of $R S N+M$ ］by auto
from 〈redundant $C$（inferred－clause－sets $R S N$ ）〉 and ind－hyp
have redundant $C$（inferred－clause－sets $R S(N+M)$ ）by auto
from 〈redundant $C$（inferred－clause－sets $R S(N+M)$ ）〉
have redundant $C$（add－all－deducible－clauses $R$（inferred－clause－sets $R S(N+M))$ ）
using deducible－clause－preserve－redundancy by blast
from this and «all－fulfill finite（add－all－deducible－clauses $R$（inferred－clause－sets $R S(N+M)))$ 〉
have redundant $C$（simplify（add－all－deducible－clauses $R$（inferred－clause－sets $R$ $S(N+M)))$ ）
using simplify－preserves－redundancy by auto
thus redundant $C$（inferred－clause－sets $R S(S u c(N+M)))$ by auto qed

We show that non－persistent clauses are strictly redundant in some element of the sequence．
lemma non－persistent－clauses－are－redundant：
assumes $D \in$ inferred－clause－sets $R S N$
assumes $D \notin$ saturation $R S$
assumes all－fulfill finite $S$
assumes derived－clauses－are－finite $R$
shows $\exists M$ ．strictly－redundant $D$（inferred－clause－sets $R S M$ ）
proof（rule ccontr）
assume hyp：$\neg(\exists M$ ．strictly－redundant $D$（inferred－clause－sets $R S M)$ ）
\｛
fix $M$
have $D \in($ inferred－clause－sets $R S(N+M))$
proof（induction $M$ ）
show $D \in$ inferred－clause－sets $R S(N+0)$ using $\operatorname{assms}(1)$ by auto next
fix $M$ assume $D \in$ inferred－clause－sets $R S(N+M)$
from this have $D \in$ add－all－deducible－clauses $R$（inferred－clause－sets $R S$ $(N+M))$ by auto
show $D \in($ inferred－clause－sets $R S(N+(S u c M)))$
proof（rule ccontr）
assume $D \notin($ inferred－clause－sets $R S(N+($ Suc $M)))$
from this and $\langle D \in$ add－all－deducible－clauses $R$（inferred－clause－sets $R S$ $(N+M)$ ）＞
have strictly－redundant $D$（add－all－deducible－clauses $R$（inferred－clause－sets $R S(N+M)))$
using simplify－def by auto
then have all－fulfill finite（add－all－deducible－clauses $R$（inferred－clause－sets $R S(N+M)))$
using assms（4）assms（3）add－all－deducible－clauses－finite［of R S N＋M］
by auto
from this
and $\langle$ strictly-redundant $D$ (add-all-deducible-clauses $R$ (inferred-clause-sets $R S(N+M))$ )
have strictly-redundant $D$ (inferred-clause-sets $R S(N+(S u c M)))$
using simplify-preserves-strict-redundancy by auto
from this and hyp show False by blast qed
qed
\}
from assms(2) and $\operatorname{assms}(1)$ have $\neg\left(\forall M^{\prime} .\left(M^{\prime} \geq N \longrightarrow D \in\right.\right.$ inferred-clause-sets $\left.R S M^{\prime}\right)$ ) by auto
from this obtain $M^{\prime}$ where $M^{\prime} \geq N$ and $D \notin$ inferred-clause-sets $R S M^{\prime}$ by auto
from $\left\langle M^{\prime} \geq N\right\rangle$ obtain $N^{\prime}::$ nat where $N^{\prime}=M^{\prime}-N$ by auto
have $D \in$ inferred-clause-sets $R S\left(N+\left(M^{\prime}-N\right)\right)$
by (simp add: < $\bigwedge M . D \in$ inferred-clause-sets $R S(N+M)\rangle)$
from this and $\left\langle D \notin\right.$ inferred-clause-sets $\left.R S M^{\prime}\right\rangle$ show False by (simp add: $\langle N$ $\left.\leq M^{\prime}>\right)$
qed
This entails that the clauses that are redundant in some set in the sequence are also redundant in the set of persistent clauses.

```
lemma persistent-clauses-subsume-redundant-clauses:
    assumes redundant \(C\) (inferred-clause-sets \(R S N\) )
    assumes all-fulfill finite \(S\)
    assumes derived-clauses-are-finite \(R\)
    assumes finite \(C\)
    shows redundant \(C\) (saturation \(R S\) )
proof -
    let ?nat-order \(=\{(x::\) nat \(, y:: n a t) . x<y\}\)
    \{
    fix \(I\) have \(\forall C N\). finite \(C \longrightarrow\) card \(C=I\)
            \(\longrightarrow(\) redundant \(C\) (inferred-clause-sets \(R S N)) \longrightarrow\) redundant \(C\) (saturation
\(R S)(\) is ? \(P I)\)
    proof ((rule wf-induct [of ?nat-order ?P I]),( simp add:wf))
    fix \(I\) assume hyp-induct: \(\forall J .(J, I) \in\) ?nat-order \(\longrightarrow(? P J)\)
    show ?P I
    proof \(((\) rule allI \()+,(\) rule impI \()+)\)
        fix \(C N\) assume finite \(C\) card \(C=I\) redundant \(C\) (inferred-clause-sets \(R S\)
N)
        show redundant \(C\) (saturation \(R S\) )
        proof (cases)
            assume tautology \(C\)
        then show redundant \(C\) (saturation \(R S\) ) unfolding redundant-def by auto
        next
            assume \(\neg\) tautology \(C\)
            from this and «redundant \(C\) (inferred-clause-sets \(R S N\) )〉 obtain \(D\)
                where subsumes \(D C\) and \(D \in\) inferred-clause-sets \(R S N\) unfolding
redundant-def by auto
```

```
    show redundant C (saturation R S)
    proof (cases)
    assume D\in saturation R S
    from this and <subsumes D C` show redundant C (saturation R S)
        unfolding redundant-def by auto
    next
        assume D\not\in saturation R S
        from assms(2) assms(3) and }\langleD\in\mathrm{ inferred-clause-sets R S N〉 and <D
& saturation R S>
    obtain M where strictly-redundant D (inferred-clause-sets R S M) using
        non-persistent-clauses-are-redundant [of D R S] by auto
        from «subsumes D C and «\negtautology C> have \negtautology D
        unfolding subsumes-def tautology-def by auto
    from «strictly-redundant D (inferred-clause-sets R S M)` and «\negtautology
D>
                obtain D' where D'\subsetD and D' }\mp@subsup{D}{}{\prime}\in\mathrm{ inferred-clause-sets R SM
                unfolding strictly-redundant-def by auto
    from }\langle\mp@subsup{D}{}{\prime}\subsetD\rangle\mathrm{ and «subsumes D C> have }\mp@subsup{D}{}{\prime}\subsetC\mathrm{ unfolding subsumes-def
by auto
            from \langle\mp@subsup{D}{}{\prime}\subsetC\rangle and <finite C> have finite D'
                by (meson psubset-imp-subset rev-finite-subset)
            from \langleD'\subsetC\rangle and <finite C\rangle have card D'< card C
                unfolding all-fulfill-def
                using psubset-card-mono by auto
            from this and <card C=I` have (card D',I) \in?nat-order by auto
    from }\langle\mp@subsup{D}{}{\prime}\in\mathrm{ inferred-clause-sets }RSM\rangle have redundant D'(inferred-clause-set
R S M)
            unfolding redundant-def subsumes-def by auto
            from hyp-induct and «(card D',I) \in?nat-order> have ?P (card D') by
force
            from this and <finite D'〉 and «redundant D' (inferred-clause-sets R S M)`
have
            redundant D' (saturation R S) by auto
            show redundant C (saturation R S)
                by (meson 〈D'\subsetC\rangle\langleredundant D' (saturation R S)\rangle
                    psubset-imp-subset subsumes-def subsumption-preserves-redundancy)
            qed
        qed
    qed
qed
}
then show redundant C (saturation R S) using assms(1) assms(4) by blast
qed
We deduce that the set of persistent clauses is saturated．
theorem persistent－clauses－are－saturated：
assumes derived－clauses－are－finite \(R\)
```

assumes all－fulfill finite $S$
shows saturated－binary－rule $R$（saturation $R S$ ）
proof（rule ccontr）
let $? S=$ saturation $R S$
assume $\neg$ saturated－binary－rule $R$ ？S
then obtain P1 P2 $C$ where $R$ P1 P2 $C$ and $P 1 \in ? S$ and $P 2 \in ? S$ and $\neg$ redundant $C$ ？$S$
unfolding saturated－binary－rule－def by blast
from $\langle P 1 \in ? S\rangle$ obtain $N 1$ where $i: \forall M .(M \geq N 1 \longrightarrow P 1 \in$（inferred－clause－sets RSM））
by auto
from $\langle P 2 \in$ ？$S\rangle$ obtain N2 where $i i: \forall M .(M \geq N 2 \longrightarrow P 2 \in$（inferred－clause－sets $R S M)$ ）
by auto
let ？$N=\max N 1 N 2$
have $? N \geq N 1$ and $? N \geq N 2$ by auto
from this and $i$ have $P 1 \in$ inferred－clause－sets $R S$ ？N by metis
from 〈？$N \geq N 2$ 2 and $i i$ have $P 2 \in$ inferred－clause－sets $R S$ ？N by metis
from $\langle R P 1 P 2 C\rangle$ and $\langle P 1 \in$ inferred－clause－sets $R S ? N\rangle$ and $\langle P 2 \in$ in－
ferred－clause－sets $R S$ ？$N$ 〉
have $C \in$ all－deducible－clauses $R$（ inferred－clause－sets $R S$ ？N）by auto
from this have $C \in$ add－all－deducible－clauses $R$（inferred－clause－sets $R S$ ？N）
by auto
from assms have all－fulfill finite（inferred－clause－sets $R S$ ？N）
using all－inferred－clause－sets－are－finite［of $R S$ ？$N$ ］by auto
from assms have all－fulfill finite（add－all－deducible－clauses $R$（inferred－clause－sets $R S$ ？N））
using add－all－deducible－clauses－finite by auto
from this and $\langle C \in$ add－all－deducible－clauses $R$（inferred－clause－sets $R S$ ？N）〉
have redundant $C$（inferred－clause－sets $R S$（Suc ？N））
using simplify－and－membership
［of add－all－deducible－clauses $R$（inferred－clause－sets $R S$ ？N）
inferred－clause－sets $R S$（Suc ？N）C］
by auto
have finite P1
using $\langle P 1 \in$ inferred－clause－sets $R S(\max N 1$ N2）$\rangle$
〈all－fulfill finite（inferred－clause－sets $R S(\max$ N1 N2））〉 all－fulfill－def by auto
have finite P2
using $\langle P 2 \in$ inferred－clause－sets $R S(\max N 1$ N2）$\rangle$
〈all－fulfill finite（inferred－clause－sets $R S(\max N 1$ N2）））all－fulfill－def by auto
from $\langle R$ P1 P2 $C\rangle$ and $\langle$ finite P1〉 and 〈finite P2 $\rangle$ and $\langle$ derived－clauses－are－finite $R$ 〉 have finite $C$
unfolding derived－clauses－are－finite－def by blast
from assms this and＜redundant $C$（inferred－clause－sets $R S(S u c$ ？N））〉
have redundant $C$（saturation $R S$ ）
using persistent－clauses－subsume－redundant－clauses［of C R S Suc ？N］
by auto

```
thus False using <\negredundant C ?S` by auto
qed
```

Finally, we show that the computed saturated set is equivalent to the initial formula.

```
theorem saturation-is-correct:
    assumes Sound R
    assumes derived-clauses-are-finite R
    assumes all-fulfill finite S
    shows equivalent S (saturation R S)
proof -
    have entails-formula S (saturation R S)
    proof (rule ccontr)
```

    assume \(\neg\) entails-formula \(S\) (saturation \(R S\) )
    then obtain \(C\) where \(C \in\) saturation \(R S\) and \(\neg\) entails \(S C\)
            unfolding entails-formula-def by auto
    from \(\langle C \in\) saturation \(R S\rangle\) obtain \(N\) where \(C \in\) inferred-clause-sets \(R\) S \(N\)
    by auto
\{
fix $N$
have entails-formula $S$ (inferred-clause-sets R S N)
proof (induction N)
show entails-formula $S$ (inferred-clause-sets R S 0)
using assms(3) simplify-preserves-semantic validity-implies-entailment by
auto
next
fix $N$ assume entails-formula $S$ (inferred-clause-sets R S N)
from assms(1) have entails-formula (inferred-clause-sets $R$ S N)
(add-all-deducible-clauses $R$ (inferred-clause-sets $R S N$ )
using add-all-deducible-sound by auto
from this and «entails-formula $S$ (inferred-clause-sets R S N) 〉
have entails-formula $S$ (add-all-deducible-clauses $R$ (inferred-clause-sets $R$
S N)
using entails-transitive
[of S inferred-clause-sets R S N add-all-deducible-clauses R (inferred-clause-sets
$R S N)]$
by auto
have inferred-clause-sets $R S(S u c N) \subseteq$ add-all-deducible-clauses $R$
(inferred-clause-sets R S N)
using simplify-def by auto
then have entails-formula (add-all-deducible-clauses $R$ (inferred-clause-sets
$R S N)$ )
(inferred-clause-sets $R S($ Suc $N)$ ) using entailment-subset by auto
from this and «entails-formula $S$ (add-all-deducible-clauses $R$ (inferred-clause-sets
$R S N)$ )
show entails-formula $S$ (inferred-clause-sets $R S($ Suc $N)$ )
using entails-transitive [of $S$ add-all-deducible-clauses $R$ (inferred-clause-sets
$R S N)]$
by auto

```
        qed
    }
    from this and «C \in inferred-clause-sets R S N\rangle and «\neg entails S C` show
False
    unfolding entails-formula-def by auto
    qed
    have entails-formula (saturation R S)S
    proof (rule ccontr)
    assume \neg entails-formula (saturation R S)S
    then obtain C where C\inS and \neg entails (saturation R S) C
        unfolding entails-formula-def by auto
    from }\langleC\inS\rangle\mathrm{ have redundant C S unfolding redundant-def subsumes-def by
auto
    from assms(3) and <redundant C S` have redundant C (inferred-clause-sets
RS 0)
            using simplify-preserves-redundancy by auto
    from assms(3) and \langleC\inS\rangle have finite C unfolding all-fulfill-def by auto
    from 〈redundant C (inferred-clause-sets R S 0)〉 assms(2) assms(3)<finite C〉
        have redundant C (saturation R S)
        using persistent-clauses-subsume-redundant-clauses [of C R S 0] by auto
    from this and «\neg entails (saturation R S) C> show False
        using entails-formula-def redundancy-implies-entailment by auto
qed
from <entails-formula S (saturation R S)` and <entails-formula (saturation R S)
S>
    show ?thesis
    unfolding equivalent-def by auto
qed
end
end
```


## 9 Prime Implicates Generation

We show that the unrestricted resolution rule is deductive complete，i．e． that it is able to generate all（prime）implicates of any given clause set．
theory Prime－Implicates
imports Propositional－Resolution
begin
context propositional－atoms
begin

### 9.1 Implicates and Prime Implicates

We first introduce the definitions of implicates and prime implicates.

```
definition implicates :: 'at Formula }=>\mathrm{ 'at Formula
    where implicates S={C. entails S C }
definition prime-implicates :: 'at Formula }=>\mathrm{ 'at Formula
    where prime-implicates S = simplify (implicates S)
```


### 9.2 Generation of Prime Implicates

We introduce a function simplifying a given clause set by evaluating some literals to false. We show that this partial evaluation operation preserves saturatedness and that if the considered set of literals is an implicate of the initial clause set then the partial evaluation yields a clause set that is unsatisfiable. Then the proof follows from refutational completeness: since the partially evaluated set is unsatisfiable and saturated it must contain the empty clause, and therefore the initial clause set necessarily contains a clause subsuming the implicate.
fun partial-evaluation :: ' $a$ Formula $\Rightarrow$ 'a Literal set $\Rightarrow{ }^{\prime} a$ Formula
where
(partial-evaluation $S C)=\{E . \exists D . D \in S \wedge E=D-C \wedge \neg(\exists L .(L \in C) \wedge$ $($ complement $L) \in D)\}$

```
lemma partial-evaluation-is-saturated :
    assumes saturated-binary-rule resolvent S
    shows saturated-binary-rule ordered-resolvent (partial-evaluation S C)
proof (rule ccontr)
    let ?peval = partial-evaluation S C
    assume \negsaturated-binary-rule ordered-resolvent ?peval
    from this obtain P1 and P2 and R where P1 \in?peval and P2 \in?peval
            and ordered-resolvent P1 P2 R and }\neg\mathrm{ (tautology R)
            and not-subsumed: }\neg(\existsD.((D\in(\mathrm{ partial-evaluation S C))}\wedge(\mathrm{ subsumes D
R)))
    unfolding saturated-binary-rule-def and redundant-def by auto
    from }\langleP1\in\mathrm{ ?peval }>\mathrm{ obtain PP1 where PP1 GS and P1 = PP1 - C
        and i:\neg(\existsL. (L\inC)\wedge (complement L) \inPP1) by auto
    from <P2 G ?peval> obtain PP2 where PP2 G S and P2 = PP2 - C
        and ii: \neg(\existsL. (L\inC)\wedge(complement L) \inPP2) by auto
    from <(ordered-resolvent P1 P2 R)> obtain A where
        r-def: R = (P1 - {Pos A }) \cup (P2 - {Neg A }) and (Pos A) \inP1 and
(Neg A) \inP2
    unfolding ordered-resolvent-def strictly-maximal-literal-def by auto
    let ?RR = (PP1 - {Pos A }) \cup(PP2 - { Neg A })
    from }\langleP1=PP1 - C> and <(Pos A) \inP1\rangle have (Pos A) \inPP1 by aut
    from <P2 = PP2 - C` and }\langle(NegA)\inP2\rangle have (Neg A) \inPP2 by aut
    from r-def and }\langleP1=PP1-C\rangle and <P2 = PP2 - C> have R = ?RR
C by auto
```

```
    from < (Pos A) \inPP1> and < (Neg A) \inPP2\rangle
            have resolvent PP1 PP2 ?RR unfolding resolvent-def by auto
    with }\langlePP1\inS\rangle\mathrm{ and }\langlePP2\mathcal{L}\inS\rangle\mathrm{ and <saturated-binary-rule resolvent S〉
    have tautology ?RR}\vee(\existsD.(D\inS\wedge(subsumes D ?RR))
    unfolding saturated-binary-rule-def redundant-def by auto
    thus False
    proof
    assume tautology ?RR
    with \langleR = ?RR - C> and «\negtautology R〉
        obtain X where X \inC and complement X \inPP1\cupPP2
        unfolding tautology-def by auto
    from }\langleX\inC\rangle\mathrm{ and <complement X PPP1 }\cupPP2\rangle and i and ii
        show False by auto
    next
    assume }\exists\textrm{D}.((D\inS)\wedge(\mathrm{ subsumes D ?RR))
    from this obtain D where D\inS and subsumes D ?RR
    by auto
    from <subsumes D ?RR` and <R =? RR - C`
        have subsumes ( D - C) R unfolding subsumes-def by auto
    from }\langleD\inS\rangle\mathrm{ and ii and }i\mathrm{ and «(subsumes D ?RR)〉 have D - C E?peval
        unfolding subsumes-def by auto
        with <subsumes (D - C) R` and not-subsumed show False by auto
    qed
qed
lemma evaluation-wrt-implicate-is-unsat :
    assumes entails S C
    assumes \negtautology C
    shows \negsatisfiable (partial-evaluation S C)
proof
    let ?peval = partial-evaluation S C
    assume satisfiable ?peval
    then obtain I where validate-formula I ?peval unfolding satisfiable-def by
auto
    let ?J = (I - {X. (Pos X) \inC }) \cup{Y. (Neg Y) \inC }
    have \negvalidate-clause ?J C
    proof
            assume validate-clause ?J C
            then obtain L where L\inC and validate-literal ?J L by auto
            obtain X where L=(Pos X)\veeL=(NegX) using Literal.exhaust [of L]
by auto
            from}\langleL=(Pos X)\veeL=(Neg X)\rangle and \langleL\inC\rangle and <\negtautology C> and
<validate-literal ?J L`
            show False unfolding tautology-def by auto
    qed
    have validate-formula ?J S
    proof (rule ccontr)
        assume \neg (validate-formula ?J S)
        then obtain D where D\inS and }\neg\mathrm{ (validate-clause ?J D) by auto
```

```
    from}\langleD\inS\rangle\mathrm{ have }D-C\in\mathrm{ ?peval }\vee (\existsL. (L\inC)\wedge (complement L)
D)
    by auto
    thus False
    proof
        assume }\existsL.(L\inC)\wedge(\mathrm{ complement L) 
        then obtain L where L\inC and complement L\inD by auto
        obtain X where L= (Pos X)\veeL=(Neg X) using Literal.exhaust [of L]
by auto
            from this and }\langleL\inC\rangle\mathrm{ and }\langle\neg(\mathrm{ tautology }C)\rangle\mathrm{ have validate-literal ?J
(complement L)
            unfolding tautology-def by auto
            from «(validate-literal ?J (complement L))\rangle and «(complement L) \inD>
            and <\neg(validate-clause ?J D)\rangle
        show False by auto
    next
        assume D-C\in ?peval
        from }\langleD-C\in\mathrm{ ?peval〉 and «(validate-formula I ?peval)>
        have validate-clause I (D-C) using validate-formula.simps by blast
        from this obtain L where L\inD and L\not\inC and validate-literal I L by
auto
    obtain X where L= (Pos X)\veeL=(Neg X) using Literal.exhaust [of L]
by auto
            from}\langleL=(\mathrm{ Pos X)}\veeL=(Neg X)\rangle and <validate-literal I L> and <L\not
C>
            have validate-literal ?J L unfolding tautology-def by auto
            from 〈validate-literal ?J L> and <L < D> and «\neg(validate-clause ?J D)〉
            show False by auto
        qed
    qed
    from〈\negvalidate-clause ?J C〉 and «validate-formula ?J S` and «entails S C`
show False
    unfolding entails-def by auto
qed
lemma entailment-and-implicates:
    assumes entails-formula S1 S2
    shows implicates S2 \subseteq implicates S1
using assms entailed-formula-entails-implicates implicates-def by auto
lemma equivalence-and-implicates:
    assumes equivalent S1 S2
    shows implicates S1 = implicates S2
using assms entailment-and-implicates equivalent-def by blast
lemma equivalence-and-prime-implicates:
    assumes equivalent S1 S2
    shows prime-implicates S1 = prime-implicates S2
using assms equivalence-and-implicates prime-implicates-def by auto
```

```
lemma unrestricted-resolution-is-deductive-complete :
    assumes saturated-binary-rule resolvent S
    assumes all-fulfill finite S
    assumes C\in implicates S
    shows redundant C S
proof ((cases tautology C),(simp add: redundant-def))
next
    assume \neg tautology C
    have }\existsD.(D\inS)^(\mathrm{ subsumes D C)
    proof -
        let ?peval = partial-evaluation S C
        from 〈saturated-binary-rule resolvent S`
            have saturated-binary-rule ordered-resolvent ?peval
            using partial-evaluation-is-saturated by auto
    from <C\in implicates S〉 have entails S C unfolding implicates-def by auto
    from <entails S C` and «\negtautology C` have \negsatisfiable ?peval
    using evaluation-wrt-implicate-is-unsat by auto
    from〈all-fulfill finite S` have all-fulfill finite ?peval unfolding all-fulfil-def
by auto
    from «\negsatisfiable ?peval` and «saturated-binary-rule ordered-resolvent ?peval`
            and <all-fulfill finite ?peval>
    have {} \in ?peval using Complete-def ordered-resolution-is-complete by blast
    then show ?thesis unfolding subsumes-def by auto
    qed
    then show ?thesis unfolding redundant-def by auto
qed
lemma prime-implicates-generation-correct :
    assumes saturated-binary-rule resolvent S
    assumes non-redundant S
    assumes all-fulfill finite S
    shows S\subseteqprime-implicates S
proof
    fix x assume }x\in
    show x }\in\mathrm{ prime-implicates S
    proof (rule ccontr)
        assume \negx\in prime-implicates S
        from}\langlex\inS\rangle have entails Sx unfolding entails-def implicates-def by aut
        then have x\in implicates S unfolding implicates-def by auto
        with }\neg~x\in(\mathrm{ prime-implicates S)}>\mathrm{ have strictly-redundant x (implicates S)
            unfolding prime-implicates-def simplify-def by auto
    from this have tautology }x\vee(\existsy.(y\in(\mathrm{ implicates S))}\wedge(y\subsetx)
            unfolding strictly-redundant-def by auto
    then have strictly-redundant x S
    proof ((cases tautology x),(simp add: strictly-redundant-def))
    next
            assume \negtautology x
```

with <tautology $x \vee(\exists y .(y \in($ implicates $S)) \wedge(y \subset x))\rangle$
obtain $y$ where $y \in$ implicates $S$ and $y \subset x$ by auto
from $\langle y \in$ implicates $S\rangle$ and $\langle$ saturated-binary-rule resolvent $S\rangle$ and $\langle$ all-fulfill finite $S$ >
have redundant y $S$ using unrestricted-resolution-is-deductive-complete by auto
from $\langle y \subset x\rangle$ and $\langle\neg$ tautology $x\rangle$ have $\neg$ tautology $y$ unfolding tautology-def by auto
with 〈redundant $y S\rangle$ obtain $z$ where $z \in S$ and $z \subseteq y$
unfolding redundant-def subsumes-def by auto
with $\langle y \subset x\rangle$ have $z \subset x$ by auto
with $\langle z \in S\rangle$ show strictly-redundant $x S$ using strictly-redundant-def by auto
qed
with $\langle$ non-redundant $S\rangle$ and $\langle x \in S\rangle$ show False unfolding non-redundant-def by auto
qed
qed
theorem prime-implicates-of-saturated-sets:
assumes saturated-binary-rule resolvent $S$
assumes all-fulfill finite $S$
assumes non-redundant $S$
shows $S=$ prime-implicates $S$
proof
from assms show $S \subseteq$ prime-implicates $S$ using prime-implicates-generation-correct by auto
show prime-implicates $S \subseteq S$
proof
fix $x$ assume $x \in$ prime-implicates $S$
from this have $x \in$ implicates $S$ unfolding prime-implicates-def simplify-def
by auto
with assms have redundant $x S$
using unrestricted-resolution-is-deductive-complete by auto
show $x \in S$
proof (rule ccontr)
assume $x \notin S$
with «redundant $x S$ have strictly-redundant $x S$
unfolding redundant-def strictly-redundant-def subsumes-def by auto
with $\langle S \subseteq$ prime-implicates $S\rangle$ have strictly-redundant $x$ (prime-implicates
S)
unfolding strictly-redundant-def by auto
then have strictly-redundant $x$ (implicates $S$ )
unfolding strictly-redundant-def prime-implicates-def simplify-def by auto
with $\langle x \in$ prime-implicates $S\rangle$ show False
unfolding prime-implicates-def simplify-def by auto
qed
qed
qed

### 9.3 Incremental Prime Implicates Computation

We show that it is possible to compute the set of prime implicates incrementally i.e., to fix an ordering among atoms, and to compute the set of resolvents upon each atom one by one, without backtracking (in the sense that if the resolvents upon a given atom are generated at some step $i$ then no resolvents upon the same atom are generated at step $i<j$. This feature is critical in practice for the efficiency of prime implicates generation algorithms.

We first introduce a function computing all resolvents upon a given atom.

```
definition all-resolvents-upon :: 'at Formula }=>\mathrm{ 'at }=>\mathrm{ ' 'at Formula
    where (all-resolvents-upon S A) ={ C.\existsP1 P2. P1 GS^P2 G S^C=
(resolvent-upon P1 P2 A) }
lemma resolvent-upon-correct:
    assumes P1 \inS
    assumes P2 }\in
    assumes }C=\mathrm{ resolvent-upon P1 P2 A
    shows entails S C
proof cases
    assume Pos A\inP1 ^Neg A \inP2
    with <C = resolvent-upon P1 P2 A> have resolvent P1 P2 C
        unfolding resolvent-def by auto
    with }\langleP1\inS\rangle\mathrm{ and }\langleP2\inS\rangle\mathrm{ show ?thesis
        using soundness-and-entailment resolution-is-correct by auto
next
    assume }\neg(Pos A\inP1\wedge Neg A\inP2)
    with <C = resolvent-upon P1 P2 A have P1 \subseteqC\vee P2 \subseteqC by auto
    with }\langleP1\inS\rangle\mathrm{ and }\langleP2\inS\rangle\mathrm{ have redundant C S
        unfolding redundant-def subsumes-def by auto
    then show ?thesis using redundancy-implies-entailment by auto
qed
lemma all-resolvents-upon-is-finite:
    assumes all-fulfill finite S
    shows all-fulfill finite (S\cup (all-resolvents-upon S A))
using assms unfolding all-fulfill-def all-resolvents-upon-def by auto
lemma atoms-formula-resolvents:
    shows atoms-formula (all-resolvents-upon S A) \subseteq atoms-formula S
unfolding all-resolvents-upon-def by auto
```

We define a partial saturation predicate that is restricted to a specific atom.

```
definition partial-saturation :: 'at Formula \(\Rightarrow\) 'at \(\Rightarrow\) 'at Formula \(\Rightarrow\) bool
where
    (partial-saturation \(S A R)=(\forall P 1\) P2. \((P 1 \in S \longrightarrow P 2 \in S\)
        \(\longrightarrow(r e d u n d a n t(r e s o l v e n t-u p o n ~ P 1 ~ P 2 ~ A) ~ R))) ~\)
```

We show that the resolvent of two redundant clauses in a partially saturated set is itself redundant．
lemma resolvent－upon－and－partial－saturation ：
assumes redundant P1 S
assumes redundant P2 $S$
assumes partial－saturation $S A(S \cup R)$
assumes $C=$ resolvent－upon P1 P2 A
shows redundant $C(S \cup R)$
proof（rule ccontr）
assume $\neg$ redundant $C(S \cup R)$
from $\langle C=$ resolvent－upon P1 P2 $A\rangle$ have $C=(P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{$
Neg $A$ \}) by auto
from $\langle\neg$ redundant $C(S \cup R)\rangle$ have $\neg$ tautology $C$ unfolding redundant－def by auto
have $\neg$（tautology P1）
proof
assume tautology P1
then obtain $B$ where Pos $B \in P 1$ and Neg $B \in P 1$ unfolding tautology－def
by auto
show False
proof cases
assume $A=B$
with $\langle$ Neg $B \in P 1\rangle$ and $\langle C=(P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{\operatorname{Neg} A\})\rangle$ have
subsumes P2 C
unfolding subsumes－def using Literal．distinct by blast
with 〈redundant P2 $S$ 〉 have redundant $C S$
using subsumption－preserves－redundancy by auto
with $\langle\neg$ redundant $C(S \cup R)$ show False unfolding redundant－def by auto
next
assume $A \neq B$
with $\langle C=(P 1-\{P o s A\}) \cup(P 2-\{N e g A\})\rangle$ and $\langle P o s B \in P 1\rangle$ and
$\langle N e g B \in P 1\rangle$
have Pos $B \in C$ and Neg $B \in C$ by auto
with $\neg \neg$ redundant $C(S \cup R)\rangle$ show False
unfolding tautology－def redundant－def by auto
qed
qed
with 〈redundant P1 S〉 obtain $Q 1$ where $Q 1 \in S$ and subsumes $Q 1$ P1
unfolding redundant－def by auto
have $\neg$（tautology P2）
proof
assume tautology P2
then obtain $B$ where Pos $B \in P 2$ and Neg $B \in P 2$ unfolding tautology－def by auto
show False
proof cases
assume $A=B$
with $\langle\operatorname{Pos} B \in P 2\rangle$ and $\langle C=(P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{\operatorname{Neg} A\})\rangle$ have
subsumes P1 C
unfolding subsumes－def using Literal．distinct by blast with 〈redundant P1 $S$ 〉 have redundant $C S$
using subsumption－preserves－redundancy by auto
with $\langle\neg$ redundant $C(S \cup R)$ 〉 show False unfolding redundant－def by auto
next
assume $A \neq B$
with $\langle C=(P 1-\{\operatorname{Pos} A\}) \cup(P 2-\{N e g A\})\rangle$ and $\langle P o s B \in P 2\rangle$ and $\langle N e g B \in P 2\rangle$
have Pos $B \in C$ and $N e g B \in C$ by auto
with $\langle\neg$ redundant $C(S \cup R)$ s show False
unfolding tautology－def redundant－def by auto
qed
qed
with 〈redundant P2 $S$ 〉 obtain $Q 2$ where $Q 2 \in S$ and subsumes $Q 2$ P2 unfolding redundant－def by auto
let ？res $=(Q 1-\{\operatorname{Pos} A\}) \cup(Q 2-\{\operatorname{Neg} A\})$
have $?$ res $=$ resolvent－upon Q1 Q2 A by auto
from this and $\langle$ partial－saturation $S A(S \cup R)\rangle$ and $\langle Q 1 \in S\rangle$ and $\langle Q 2 \in S\rangle$
have redundant ？res $(S \cup R)$
unfolding partial－saturation－def by auto
from 〈subsumes Q1 P1〉 and 〈subsumes Q2 P2〉 and $\langle C=(P 1-\{\operatorname{Pos} A\})$
$\cup(P 2-\{\operatorname{Neg} A\})\rangle$
have subsumes ？res $C$ unfolding subsumes－def by auto
with «redundant ？res $(S \cup R)$ 〉 and $\langle\neg$ redundant $C(S \cup R)$ 〉 show False
using subsumption－preserves－redundancy by auto
qed
We show that if $R$ is a set of resolvents of a set of clauses $S$ then the same holds for $S \cup R$ ．For the clauses in $S$ ，the premises are identical to the resolvent and the inference is thus redundant（this trick is useful to simplify proofs）．
definition in－all－resolvents－upon：：＇at Formula $\Rightarrow{ }^{\prime}$＇at $\Rightarrow$＇at Clause $\Rightarrow$ bool where
in－all－resolvents－upon $S A C=(\exists P 1 P 2 .(P 1 \in S \wedge P 2 \in S \wedge C=$ resol－ vent－upon P1 P2 A））
lemma every－clause－is－a－resolvent：
assumes all－fulfill（in－all－resolvents－upon $S A$ ）$R$
assumes all－fulfill $(\lambda x$ ．$\neg($ tautology $x)) S$
assumes $P 1 \in S \cup R$
shows in－all－resolvents－upon S A P1
proof $(($ cases P1 $\in R),($ metis all－fulfill－def assms（1）））
next
assume $P 1 \notin R$
with $\langle P 1 \in S \cup R\rangle$ have $P 1 \in S$ by auto
with 〈（all－fulfill $(\lambda x$ ．$\neg($ tautology $x)) S)$ have $\neg$ tautology P1
unfolding all－fulfill－def by auto
from $\langle\neg$ tautology P1 have Neg $A \notin P 1 \vee \operatorname{Pos} A \notin P 1$ unfolding tautology－def

```
by auto
    from this have P1 = (P1 - {Pos A }) \cup (P1 - {Neg A }) by auto
    with }\langleP1\inS\rangle\mathrm{ show ?thesis unfolding resolvent-def
        unfolding in-all-resolvents-upon-def by auto
qed
```

We show that if a formula is partially saturated then it stays so when new resolvents are added in the set．
lemma partial－saturation－is－preserved ：
assumes partial－saturation S E1 S
assumes partial－saturation $S$ E2 $(S \cup R)$
assumes all－fulfill $(\lambda x$ ．$\neg($ tautology $x)) S$
assumes all－fulfill（in－all－resolvents－upon $S$ E2）$R$
shows partial－saturation $(S \cup R) E 1(S \cup R)$
proof（rule ccontr）
assume $\neg$ partial－saturation $(S \cup R) E 1(S \cup R)$
from this obtain P1 P2 $C$ where $P 1 \in S \cup R$ and $P 2 \in S \cup R$ and $C=$ resolvent－upon P1 P2 E1
and $\neg$ redundant $C(S \cup R)$
unfolding partial－saturation－def by auto
from $\langle C=$ resolvent－upon P1 P2 E1 $\downarrow$ have $C=(P 1-\{$ Pos E1 $\}) \cup($ P2－
\｛ Neg E1 \}) by auto
from $\langle P 1 \in S \cup R\rangle$ and $\operatorname{assms}(4)$ and $\langle($ all－fulfill $(\lambda x . \neg($ tautology $x)) S)\rangle$
have in－all－resolvents－upon S E2 P1 using every－clause－is－a－resolvent by auto
then obtain $P 1-1 P 1-2$ where $P 1-1 \in S$ and $P 1-2 \in S$ and $P 1=$ resol－ vent－upon P1－1 P1－2 E2
using every－clause－is－a－resolvent unfolding in－all－resolvents－upon－def by blast from $\langle P 2 \in S \cup R\rangle$ and $\operatorname{assms}(4)$ and $\langle($ all－fulfill $(\lambda x . \neg($ tautology $x)) S$ ）〉
have in－all－resolvents－upon S E2 P2 using every－clause－is－a－resolvent by auto
then obtain P2－1 P2－2 where $P 2-1 \in S$ and $P 2-2 \in S$ and $P 2=$ resol－ vent－upon P2－1 P2－2 E2
using every－clause－is－a－resolvent unfolding in－all－resolvents－upon－def by blast let ？R1＝resolvent－upon P1－1 P2－1 E1
from $\langle$ partial－saturation $S E 1 S\rangle$ and $\langle P 1-1 \in S\rangle$ and $\langle P 2-1 \in S\rangle$ have redun－ dant ？R1 S
unfolding partial－saturation－def by auto
let ？R2＝resolvent－upon P1－2 P2－2 E1
from $\langle$ partial－saturation $S E 1 S\rangle$ and $\langle P 1-2 \in S\rangle$ and $\langle P 2-2 \in S\rangle$ have redun－ dant ？R2 $S$
unfolding partial－saturation－def by auto
let ？$C=$ resolvent－upon ？R1 ？R2 E2
from $\langle C=$ resolvent－upon P1 P2 E1 $\rangle$ and $\langle P 2=$ resolvent－upon P2－1 P2－2 E2 $\rangle$
and $\langle P 1=$ resolvent－upon P1－1 P1－2 E2 $\rangle$
have ？$C=C$ by auto
with 〈redundant ？R1 $S$ 〉 and 〈redundant ？R2 $S\rangle$ and 〈partial－saturation $S$ E2 $(S \cup R)$＞
and «ᄀ redundant $C(S \cup R)$ 〉
show False using resolvent－upon－and－partial－saturation by auto

## qed

The next lemma shows that the clauses inferred by applying the resolu－ tion rule upon a given atom contain no occurrence of this atom，unless the inference is redundant．

```
lemma resolvents-do-not-contain-atom :
    assumes \(\neg\) tautology P1
    assumes \(\neg\) tautology P2
    assumes \(C=\) resolvent-upon P1 P2 E2
    assumes \(\neg\) subsumes P1 C
    assumes \(\neg\) subsumes P2 \(C\)
    shows (Neg E2) \(\notin C \wedge(\) Pos E2 \() \notin C\)
proof
    from 〈C = resolvent-upon P1 P2 E2 \(\langle\) have \(C=(P 1-\{\) Pos E2 \(\}) \cup(\) P2 -
\{ Neg E2 \})
        by auto
    show (Neg E2) \(\notin C\)
    proof
        assume Neg E2 \(\in C\)
        from \(\langle C=\) resolvent-upon P1 P2 E2 \(\rangle\) have \(C=(\) P1 - \{Pos E2 \(\}) \cup(P 2-\)
\{ Neg E2 \})
            by auto
        with \(\langle N e g E 2 \in C\rangle\) have \(N e g E 2 \in P 1\) by auto
        from \(\langle\neg\) subsumes \(P 1 C\rangle\) and \(\langle C=(P 1-\{\) Pos E2 \(\}) \cup(P 2-\{N e g E 2\)
\})> have Pos E2 \(\in P 1\)
            unfolding subsumes-def by auto
        from \(\langle\) Neg E2 \(\in P 1\rangle\) and \(\langle P o s E 2 \in P 1\rangle\) and \(\langle\neg\) tautology \(P 1\rangle\) show False
            unfolding tautology-def by auto
        qed
        next show (Pos E2) \(\notin C\)
        proof
        assume Pos \(E 2 \in C\)
        from 〈C = resolvent-upon P1 P2 E2〉 have \(C=(\) P1 - \{Pos E2 \} \() \cup(\) P2 -
\{ Neg E2 \})
            by auto
        with \(\langle\) Pos E2 \(\in C\) 〉 have Pos E2 \(\in P 2\) by auto
        from \(\langle\neg\) subsumes P2 \(C\rangle\) and \(\langle C=(P 1-\{\) Pos E2 \(\}) \cup(P 2-\{N e g E 2\)
\})> have Neg E2 \(\in P\) 2
            unfolding subsumes-def by auto
        from \(\langle N e g E 2 \in P 2\rangle\) and \(\langle P o s E 2 \in P 2\rangle\) and \(\langle\neg\) tautology P2〉 show False
            unfolding tautology-def by auto
    qed
qed
```

The next lemma shows that partial saturation can be ensured by computing all（non－redundant）resolvents upon the considered atom．
lemma ensures－partial－saturation ：
assumes partial－saturation $S$ E2 $(S \cup R)$
assumes all－fulfill $(\lambda x$ ．$\neg($ tautology $x)) S$
assumes all－fulfill（in－all－resolvents－upon S E2）$R$
assumes all－fulfill $(\lambda x$ ．$(\neg$ redundant $x S)) R$
shows partial－saturation $(S \cup R)$ E2 $(S \cup R)$
proof（rule ccontr）
assume $\neg$ partial－saturation $(S \cup R)$ E2 $(S \cup R)$
from this obtain P1 P2 $C$ where $P 1 \in S \cup R$ and $P 2 \in S \cup R$ and $C=$ resolvent－upon P1 P2 E2
and $\neg$ redundant $C(S \cup R)$
unfolding partial－saturation－def by auto
have $P 1 \in S$
proof（rule ccontr）
assume P1 $\notin S$
with $\langle P 1 \in S \cup R\rangle$ have $P 1 \in R$ by auto
with $\operatorname{assms}(3)$ obtain $P 1-1$ and $P 1-2$ where $P 1-1 \in S$ and P1－2 $\in S$
and P1 $=$ resolvent－upon P1－1 P1－2 E2
unfolding all－fulfill－def in－all－resolvents－upon－def by auto
from 〈all－fulfill $(\lambda x . \neg($ tautology $x)) S\rangle$ and $\langle P 1-1 \in S\rangle$ and $\langle P 1-2 \in S\rangle$
have $\neg$ tautology P1－1 and $\neg$ tautology P1－2
unfolding all－fulfill－def by auto
from $\langle$ all－fulfill $(\lambda x$ ．$(\neg$ redundant $x S)) R\rangle$ and $\langle P 1 \in R\rangle$ and $\langle P 1-1 \in S\rangle$ and $\langle P 1-2 \in S\rangle$
have $\neg$ subsumes P1－1 P1 and $\neg$ subsumes P1－2 P1
unfolding redundant－def all－fulfill－def by auto
from 〈 $\neg$ tautology P1－1〉 $\neg$ tautology P1－2〉 $\neg \neg$ subsumes P1－1 P1〉 and $\langle\neg$ subsumes P1－2 P1＞
and $\langle P 1=$ resolvent－upon P1－1 P1－2 E2〉
have（Neg E2）$\notin P 1 \wedge($ Pos E2 $) \notin P 1$
using resolvents－do－not－contain－atom［of P1－1 P1－2 P1 E2］by auto
with $\langle C=$ resolvent－upon P1 P2 E2〉 have subsumes P1 $C$ unfolding sub－ sumes－def by auto
with $\langle\neg$ redundant $C(S \cup R)\rangle$ and $\langle P 1 \in S \cup R\rangle$ show False unfolding redundant－def
by auto
qed
have $P 2 \in S$
proof（rule ccontr）
assume $P 2 \notin S$
with $\langle P 2 \in S \cup R\rangle$ have $P 2 \in R$ by auto
with $\operatorname{assms}(3)$ obtain P2－1 and P2－2 where P2－1 $\in S$ and P2－2 $\in S$ and P2 $=$ resolvent－upon P2－1 P2－2 E2 unfolding all－fulfill－def in－all－resolvents－upon－def by auto
from 〈（all－fulfill $(\lambda x . \neg($ tautology $x)) S)\rangle$ and $\langle P 2-1 \in S\rangle$ and $\langle P 2-2 \in S\rangle$ have $\neg$ tautology P2－1 and $\neg$ tautology P2－2 unfolding all－fulfill－def by auto
from $\langle$ all－fulfill $(\lambda x$ ．$(\neg$ redundant $x S)) R\rangle$ and $\langle P 2 \in R\rangle$ and $\langle P 2-1 \in S\rangle$ and $\langle P 2-2 \in S\rangle$
have $\neg$ subsumes P2－1 P2 and $\neg$ subsumes P2－2 P2
unfolding redundant－def all－fulfill－def by auto
from $\langle\neg$ tautology P2－1〉 $\neg ~ t a u t o l o g y ~ P 2-2 〉\langle\neg ~ s u b s u m e s ~ P 2-1 ~ P 2 〉 ~ a n d ~ 〈 \neg ~$

```
subsumes P2-2 P2`
```



```
    have (Neg E2) &P2 ^(Pos E2) & P2
    using resolvents-do-not-contain-atom [of P2-1 P2-2 P2 E2] by auto
    with 〈C= resolvent-upon P1 P2 E2` have subsumes P2 C unfolding sub-
sumes-def by auto
    with }\neg\mathrm{ redundant }C(S\cupR)\rangle\mathrm{ and }\langleP2\mathcal{L}\inS\cupR
            show False unfolding redundant-def by auto
    qed
```



```
    and <C = resolvent-upon P1 P2 E2\rangle and <\neg redundant C (S\cupR)>
    show False unfolding redundant-def partial-saturation-def by auto
qed
lemma resolvents-preserve-equivalence:
    shows equivalent S(S\cup(all-resolvents-upon S A))
proof -
    have S\subseteq(S\cup(all-resolvents-upon S A)) by auto
    then have entails-formula (S\cup(all-resolvents-upon S A)) S using entail-
ment-subset by auto
    have entails-formula S(S\cup(all-resolvents-upon SA))
    proof (rule ccontr)
        assume \negentails-formula S(S\cup(all-resolvents-upon S A))
        from this obtain C where C ( all-resolvents-upon SA) and \negentails S C
            unfolding entails-formula-def using entails-member by auto
        from <C\in(all-resolvents-upon S A)\rangle obtain P1 P2
            where C= resolvent-upon P1 P2 A and P1 \inS and P2 G S
            unfolding all-resolvents-upon-def by auto
    from }\langleC=\mathrm{ resolvent-upon P1 P2 A> and }\langleP1\inS\rangle\mathrm{ and }\langleP2~S\rangle\mathrm{ have entails
SC
            using resolvent-upon-correct by auto
            with «\negentails S C` show False by auto
    qed
    from <entails-formula (S\cup(all-resolvents-upon S A)) S`
        and <entails-formula S(S\cup(all-resolvents-upon S A))>
        show ?thesis unfolding equivalent-def by auto
qed
```

Given a sequence of atoms, we define a sequence of clauses obtained by resolving upon each atom successively. Simplification rules are applied at each iteration step.

```
fun resolvents-sequence :: (nat \(\Rightarrow{ }^{\prime}\) at \() \Rightarrow{ }^{\prime}\) 'at Formula \(\Rightarrow\) nat \(\Rightarrow\) 'at Formula
where
    (resolvents-sequence A S 0) \(=(\) simplify \(S) \mid\)
    (resolvents-sequence A \(S(\) Suc \(N)\) ) \(=\)
    (simplify ((resolvents-sequence ASN)
        \(\cup(\) all-resolvents-upon (resolvents-sequence \(A S N)(A N)))\)
```

The following lemma states that partial saturation is preserved by simplifi-
cation．

```
lemma redundancy-implies-partial-saturation:
    assumes partial-saturation S1 A S1
    assumes \(S 2 \subseteq S 1\)
    assumes all-fulfill ( \(\lambda x\). redundant \(x\) S2) S1
    shows partial-saturation S2 A S2
proof (rule ccontr)
    assume \(\neg\) partial-saturation S2 A S2
    then obtain P1 P2 \(C\) where \(P 1 \in S 2 P 2 \in S 2\) and \(C=(\) resolvent-upon P1
P2 A)
        and \(\neg\) redundant \(C\) S2
        unfolding partial-saturation-def by auto
    from \(\langle P 1 \in S 2\rangle\) and \(\langle S 2 \subseteq S 1\rangle\) have \(P 1 \in S 1\) by auto
    from \(\langle P 2 \in S 2\rangle\) and \(\langle S 2 \subseteq S 1\rangle\) have \(P 2 \in S 1\) by auto
    from \(\langle P 1 \in S 1\rangle\) and \(\langle P 2 \in S 1\rangle\) and \(\langle\) partial-saturation S1 A S1 \(\rangle\) and \(\langle C=\)
resolvent-upon P1 P2 A)
    have redundant C S1 unfolding partial-saturation-def by auto
    from \(« \neg\) redundant \(C\) S 2\(\rangle\) have \(\neg\) tautology \(C\) unfolding redundant-def by auto
    with «redundant \(C\) S1〉 obtain \(D\) where \(D \in S 1\) and \(D \subseteq C\)
        unfolding redundant-def subsumes-def by auto
    from \(\langle D \in\) S1〉 and «all-fulfill ( \(\lambda x\). redundant \(x\) S2) S1〉 have redundant D S2
        unfolding all-fulfill-def by auto
    from \(\langle\neg\) tautology \(C\rangle\) and \(\langle D \subseteq C\rangle\) have \(\neg\) tautology \(D\) unfolding tautology-def
by auto
    with 〈redundant D S2〉 obtain \(E\) where \(E \in S 2\) and \(E \subseteq D\)
        unfolding redundant-def subsumes-def by auto
    from \(\langle E \subseteq D\rangle\) and \(\langle D \subseteq C\rangle\) have \(E \subseteq C\) by auto
    from \(\langle E \in S 2\rangle\) and \(\langle E \subseteq C\rangle\) and \(\langle\neg\) redundant \(C\) S2〉 show False
        unfolding redundant-def subsumes-def by auto
qed
```

The next theorem finally states that the implicate generation algorithm is sound and complete in the sense that the final clause set in the sequence is exactly the set of prime implicates of the considered clause set．

```
theorem incremental-prime-implication-generation:
    assumes atoms-formula \(S=\{X . \exists I::\) nat. \(I<N \wedge X=(A I)\}\)
    assumes all-fulfill finite \(S\)
    shows \((\) prime-implicates \(S)=(\) resolvents-sequence \(A S N)\)
proof -
```

We define a set of invariants and show that they are satisfied by all sets in the above sequence．For the last set in the sequence，the invariants ensure that the clause set is saturated，which entails the desired property．

```
let ?Final = resolvents-sequence A SN
```

We define some properties and show by induction that they are satisfied by all the clause sets in the constructed sequence

```
let ?equiv-init = \lambdaI.(equivalent S(resolvents-sequence ASI))
```

```
    let ?partial-saturation = \lambdaI. ( }\forall\mathrm{ J::nat. ( }J<
    \longrightarrow \text { (partial-saturation (resolvents-sequence A S I) (A J) (resolvents-sequence}
A SI))))
    let ?no-tautologies = \lambdaI.(all-fulfill }(\lambdax.\neg(tautology x))(resolvents-sequence A S
I) )
    let ?atoms-init = \lambdaI.(atoms-formula (resolvents-sequence A S I)
                            \subseteq \{ X . \exists I : : n a t . ~ I < N \wedge X = ( A I ) \} )
    let ?non-redundant = \lambdaI.(non-redundant (resolvents-sequence A SI))
    let ?finite = \I.(all-fulfill finite (resolvents-sequence A S I))
    have }\forallI.(I\leqN\longrightarrow(?equiv-init I) ^(?partial-saturation I)^(?no-tautologies
I)
    \wedge(?atoms-init I) ^(?non-redundant I)}\wedge(?finite I) )
    proof (rule allI)
    fix I
    show (I\leqN
    \longrightarrow ( ? e q u i v - i n i t ~ I ) \wedge ( ? p a r t i a l - s a t u r a t i o n ~ I ) ~ \wedge ( ? n o - t a u t o l o g i e s ~ I ) ~ \wedge ( ? a t o m s - i n i t ~
I)
            \wedge(?non-redundant I)^(?finite I) )(is I\leqN\longrightarrow?P I)
    proof (induction I)
```

We show that the properties are all satisfied by the initial clause set（after simplification）．

```
show \(0 \leq N \longrightarrow\) ? P 0
proof (rule impI)+
    assume \(0 \leq N\)
    let \(? R=\) resolvents-sequence \(A S 0\)
    from 〈all-fulfill finite \(S\) 〉
    have ?equiv-init 0 using simplify-preserves-equivalence by auto
    moreover have ?no-tautologies 0
            using simplify-def strictly-redundant-def all-fulfill-def by auto
    moreover have ?partial-saturation 0 by auto
    moreover from 〈all-fulfill finite \(S\) 〉 have ?finite 0 using simplify-finite
    moreover have atoms-formula ? \(R \subseteq\) atoms-formula \(S\) using atoms-formula-simplify
    moreover with «atoms-formula \(S=\{X . \exists I::\) nat. \(I<N \wedge X=(A I)\)
            have \(v\) : ?atoms-init 0 unfolding simplify-def by auto
            moreover have ?non-redundant 0 using simplify-non-redundant by auto
            ultimately show ?P 0 by auto
    qed
```

by auto
by auto
\})

We then show that the properties are preserved by induction．

```
next
fix I assume I\leqN\longrightarrow?P I
show (Suc I) \leqN \longrightarrow(?P (Suc I))
proof (rule impI)+
```

assume $($ Suc $I) \leq N$
let ？Prec $=$ resolvents－sequence $A S I$
let $? R=$ resolvents－sequence $A S(S u c I)$
from $\langle S u c I \leq N\rangle$ and $\langle I \leq N \longrightarrow ? P I\rangle$
have ？equiv－init $I$ and ？partial－saturation $I$ and ？no－tautologies $I$ and
？finite I
and ？atoms－init I and ？non－redundant I by auto
have equivalent ？Prec（？Prec $\cup$（all－resolvents－upon ？Prec（AI）））
using resolvents－preserve－equivalence by auto
from 〈？finite $I\rangle$ have all－fulfill finite（？Prec $\cup$（all－resolvents－upon ？Prec （A $I)$ ））
using all－resolvents－upon－is－finite by auto
then have all－fulfill finite（simplify（？Prec $\cup$（all－resolvents－upon ？Prec（ $A$ I））））
using simplify－finite by auto
then have ？finite（Suc I）by auto
from «all－fulfill finite（？Prec $\cup$（all－resolvents－upon ？Prec（A I）））＞
have equivalent（？Prec $\cup($ all－resolvents－upon ？Prec（AI）））？R
using simplify－preserves－equivalence by auto
from «equivalent ？Prec（？Prec $\cup($ all－resolvents－upon ？Prec $(A I)))$ 〉
and «equivalent（？Prec $\cup($ all－resolvents－upon ？Prec $(A I)))$ ？R〉
have equivalent ？Prec ？R by（rule equivalent－transitive）
from〈？equiv－init $I$ 〉 and this have ？equiv－init（Suc I）by（rule equiva－ lent－transitive）
have ？no－tautologies（Suc I）using simplify－def strictly－redundant－def all－fulfill－def
by auto
let ？Delta $=$ ？$R-$ ？Prec
have ？$R \subseteq$ ？Prec $\cup$ ？Delta by auto
have all－fulfill $(\lambda x$ ．（redundant $x$ ？$R$ ））（？Prec $\cup$ ？Delta）
proof（rule ccontr）
assume $\neg$ all－fulfill $(\lambda x$ ．（redundant $x$ ？$R$ ））（？Prec $\cup$ ？Delta）
then obtain $x$ where $\neg$ redundant $x ? R$ and $x \in$ ？Prec $\cup$ ？Delta unfolding all－fulfill－def
by auto
from $\neg$ redundant $x ? R\rangle$ have $\neg x \in ? R$ unfolding redundant－def sub－ sumes－def by auto
with $\langle x \in$ ？Prec $\cup$ ？Delta have $x \in$（？Prec $\cup$（all－resolvents－upon ？Prec
（ $A \quad I)$ ））
by auto
with＜all－fulfill finite（？Prec $\cup$（all－resolvents－upon ？Prec $(A I))$ ）＞
have redundant $x($ simplify $(? P r e c \cup($ all－resolvents－upon ？Prec $(A I))))$ using simplify－and－membership by blast
with $\neg$ redundant $x$ ？R〉 show False by auto
qed
have all－fulfill（in－all－resolvents－upon ？Prec（A I））？Delta
proof（rule ccontr）
assume $\neg($ all－fulfill（in－all－resolvents－upon ？Prec（A I））？Delta）
then obtain $C$ where $C \in$ ？Delta
and $\neg$ in－all－resolvents－upon ？Prec $(A I) C$
unfolding all－fulfill－def by auto
then obtain $C$ where $C \in$ ？Delta
and not－res：$\forall$ P1 P2．$\neg(P 1 \in$ ？Prec $\wedge P 2 \in$ ？Prec $\wedge C=$ resolvent－upon
P1 P2（A I））
unfolding all－fulfill－def in－all－resolvents－upon－def by blast
from $\langle C \in ?$ Delta〉 have $C \in ? R$ and $C \notin ?$ Prec by auto
then have $C \in$ simplify（？Prec $\cup$（all－resolvents－upon ？Prec $(A I))$ ）by
auto
then have $C \in$ ？Prec $\cup$（all－resolvents－upon ？Prec（ $A$ I ））unfolding
simplify－def by auto
with $\langle C \notin$ ？Prec〉 have $C \in$（all－resolvents－upon ？Prec $(A I))$ by auto
with not－res show False unfolding all－resolvents－upon－def by auto
qed
have all－fulfill（ $\lambda x$ ．（ $\neg$ redundant $x$ ？Prec））？Delta
proof（rule ccontr）
assume $\neg$ all－fulfill $(\lambda x$ ．$(\neg$ redundant $x$ ？Prec $)$ ）？Delta
then obtain $C$ where $C \in$ ？Delta and redundant：redundant $C$ ？Prec unfolding all－fulfill－def by auto
from $\langle C \in ?$ Delta〉 have $C \in ? R$ and $C \notin$ ？Prec by auto show False
proof cases
assume strictly－redundant $C$ ？Prec
then have strictly－redundant $C$（？Prec $\cup$（all－resolvents－upon ？Prec（ $A$
I）））
unfolding strictly－redundant－def by auto
then have $C \notin$ simplify（？Prec $\cup($ all－resolvents－upon ？Prec $(A I)))$
unfolding simplify－def by auto
then have $C \notin ? R$ by auto
with $\langle C \in ? R\rangle$ show False by auto
next assume $\neg$ strictly－redundant $C$ ？Prec
with redundant have $C \in$ ？Prec
unfolding strictly－redundant－def redundant－def subsumes－def by auto
with $\langle C \notin$ ？Prec〉 show False by auto
qed
qed
have $\forall J$ ：：nat．$(J<($ Suc $I)) \longrightarrow($ partial－saturation ？R $(A J)$ ？R）
proof（rule ccontr）
assume $\neg(\forall J$ ：：nat．$(J<($ Suc $I)) \longrightarrow($ partial－saturation ？R $(A J)$ ？R）$)$
then obtain $J$ where $J<($ Suc $I)$ and $\neg($ partial－saturation ？R $(A J)$
？R）by auto
from $\langle\neg($ partial－saturation ？$R(A J)$ ？$R)\rangle$ obtain P1 P2 $C$
where $P 1 \in ? R$ and $P 2 \in ? R$ and $C=$ resolvent－upon P1 P2 $(A J)$ and

## $\neg$ redundant $C$ ？$R$

unfolding partial－saturation－def by auto
have partial－saturation ？Prec（A I）（？Prec $\cup$ ？Delta）
proof（rule ccontr）
assume $\neg$ partial－saturation ？Prec（A I）（？Prec $\cup$ ？Delta）
then obtain P1 P2 $C$ where P1 $\in$ ？Prec and P2 $\in$ ？Prec
and $C=$ resolvent－upon P1 P2（ $A$ I）and
$\neg$ redundant $C$（？Prec $\cup$ ？Delta）unfolding partial－saturation－def by
auto
from $\langle C=$ resolvent－upon P1 P2 $(A I)\rangle$ and $\langle P 1 \in$ ？Prec $\rangle$ and $\langle P 2 \in$ ？Prec＞
have $C \in$ ？Prec $\cup$（all－resolvents－upon ？Prec（ $A$ I ））
unfolding all－resolvents－upon－def by auto
from＜all－fulfill finite（？Prec $\cup$（all－resolvents－upon ？Prec（AI）））＞ and this have redundant $C$ ？$R$
using simplify－and－membership［of ？Prec $\cup$（all－resolvents－upon ？Prec
$(A I)) ? R C]$ by auto
with $\langle ? R \subseteq$ ？Prec $\cup$ ？Delta have redundant $C$（？Prec $\cup$ ？Delta）
using superset－preserves－redundancy $[$ of $C$ ？$R$（？Prec $\cup$ ？Delta $)$ ］by auto with $\neg$ redundant $C$（？Prec $\cup$ ？Delta） show False by auto
qed
show False
proof cases
assume $J=I$
from 〈partial－saturation ？Prec（A I）（？Prec $\cup$ ？Delta）〉 and «？no－tautologies
and «（all－fulfill（in－all－resolvents－upon ？Prec（A I））？Delta）〉
and 〈all－fulfill $(\lambda x$ ．$\neg$ redundant $x$ ？Prec $)$ ）？Delta〉
have partial－saturation（？Prec $\cup$ ？Delta）$(A I)(? P r e c \cup$ ？Delta）
using ensures－partial－saturation［of ？Prec（A I）？Delta］by auto
with $\langle ? R \subseteq$ ？Prec $\cup$ ？Delta
and «all－fulfill $(\lambda x$ ．（redundant $x ? R)$ ）（？Prec $\cup$ ？Delta）＞
have partial－saturation ？R（AI）？R using redundancy－implies－partial－saturation
by auto
with $\langle J=I\rangle$ and $\langle\neg($ partial－saturation ？$R(A J)$ ？$R$ ）〉 show False by
auto
next
assume $J \neq I$
with $\langle J<($ Suc $I)\rangle$ have $J<I$ by auto
with 〈？partial－saturation $I$ 〉
have partial－saturation ？Prec $(A J)$ ？Prec by auto
with «partial－saturation ？Prec（AI）（？Prec $\cup$ ？Delta）〉 and 〈？no－tautologies
I $>$
and «（all－fulfill（in－all－resolvents－upon ？Prec（A I））？Delta）〉
and 〈all－fulfill $(\lambda x$ ．$\neg$ redundant $x$ ？Prec $)$ ）？Delta $\rangle$
have partial－saturation（？Prec $\cup$ ？Delta）$(A \quad J)(? P r e c \cup ? D e l t a)$
using partial－saturation－is－preserved［of ？Prec A J A I ？Delta］by auto with «？$R \subseteq$ ？Prec $\cup$ ？Delta
and＜all－fulfill $(\lambda x$ ．（redundant $x ? R))(? P r e c \cup$ ？Delta $)$ 〉
have partial－saturation ？$R(A J) ? R$
using redundancy－implies－partial－saturation by auto
with $\neg \neg($ partial－saturation ？$R(A J)$ ？$R)$ 〉show False by auto
qed

```
    qed
    have non-redundant ?R using simplify-non-redundant by auto
    from <?atoms-init I` have atoms-formula (all-resolvents-upon ?Prec (A I))
                \subseteq \{ X . \exists I : : n a t . ~ I < N \wedge ~ X = ( A I ) \}
    using atoms-formula-resolvents [of ?Prec A I] by auto
    with <?atoms-init I`
        have atoms-formula (?Prec \cup (all-resolvents-upon ?Prec (A I)))
                \subseteq \{ X . \exists I : : n a t . ~ I < N \wedge X = ( A I ) \}
        using atoms-formula-union [of ?Prec all-resolvents-upon ?Prec (A I)] by
auto
    from this have atoms-formula ?R \subseteq{X. \existsI::nat. I<N\wedge X=(AI)}
    using atoms-formula-simplify [of ?Prec \cup (all-resolvents-upon ?Prec (A I))]
by auto
    from <equivalent S (resolvents-sequence A S (Suc I))>
        and < (\forallJ::nat. ( }J<(Suc I
            \longrightarrow(partial-saturation (resolvents-sequence A S (Suc I)) (A J)
                (resolvents-sequence A S (Suc I)))))>
            and <(all-fulfill ( }\lambdax.\neg(\mathrm{ tautology x)) (resolvents-sequence A S (Suc I)))>
            and<(all-fulfill finite (resolvents-sequence A S (Suc I)))>
            and <non-redundant ?R`
            and <atoms-formula (resolvents-sequence A S (Suc I)) \subseteq{X.\existsI::nat.
I<N\wedge X=(AI)}>
            show ?P (Suc I) by auto
        qed
    qed
qed
```

Using the above invariants，we show that the final clause set is saturated．

```
    from this have \(\forall J .(J<N \longrightarrow\) partial-saturation ?Final \((A J)\) ?Final \()\)
    and atoms-formula (resolvents-sequence \(A S N\) ) \(\subseteq\{X . \exists I::\) nat. \(I<N \wedge X\)
\(=(A I)\}\)
    and equivalent \(S\) ?Final
    and non-redundant? Final
    and all-fulfill finite? Final
by auto
have saturated-binary-rule resolvent ?Final
proof (rule ccontr)
    assume \(\neg\) saturated-binary-rule resolvent ?Final
    then obtain P1 P2 \(C\) where P1 \(\in\) ?Final and P2 \(\in\) ?Final and resolvent
P1 P2 C
        and \(\neg\) redundant \(C\) ?Final
        unfolding saturated-binary-rule-def by auto
    from 〈resolvent P1 P2 \(C\) > obtain \(B\) where \(C=\) resolvent-upon P1 P2 B
        unfolding resolvent-def by auto
        show False
    proof cases
        assume \(B \in\) (atoms-formula ?Final)
        with 〔atoms-formula ? Final \(\subseteq\{X . \exists I::\) nat. \(I<N \wedge X=(A I)\}\) 〉
            obtain \(I\) where \(B=(A I)\) and \(I<N\)
```

by auto
from $\langle B=(A I)\rangle$ and $\langle C=$ resolvent－upon P1 P2 $B\rangle$ have $C=$ resolvent－upon P1 P2（A I）
by auto
from $\langle\forall J .(J<N \longrightarrow$ partial－saturation ？Final $(A J)$ ？Final $)\rangle$ and $\langle B=$ $(A I)\rangle$ and $\langle I<N\rangle$
have partial－saturation ？Final（ $A$ I）？Final by auto
with $\langle C=$ resolvent－upon P1 P2 $(A I)\rangle$ and $\langle P 1 \in$ ？Final $\rangle$ and $\langle P 2 \in$ ？Final＞
have redundant $C$ ？Final unfolding partial－saturation－def by auto with «ᄀredundant $C$ ？Final〉 show False by auto
next
assume $B \notin$ atoms－formula ？Final
with $\langle P 1 \in$ ？Final $\rangle$ have $B \notin$ atoms－clause P1 by auto
then have $\operatorname{Pos} B \notin P 1$ by auto
with $\langle C=$ resolvent－upon P1 P2 B $\rangle$ have P1 $\subseteq C$ by auto
with $\langle P 1 \in$ ？Final〉 and $\langle\neg$ redundant $C$ ？Final〉 show False
unfolding redundant－def subsumes－def by auto
qed
qed
with〈all－fulfill finite ？Final〉 and 〈non－redundant ？Final〉
have prime－implicates ？Final $=$ ？Final
using prime－implicates－of－saturated－sets［of ？Final］by auto
with 〈equivalent $S$ ？Final〉 show ？thesis using equivalence－and－prime－implicates by auto
qed
end
end

