

Automated Deduction Compendium SS2023

Ida Hönigmann

May 16, 2023

1 Introduction, SAT Solving

1.0.1 Proposition, Formulas

Def (Proposition). *Proposition is a statement that can be either true or false.*

Def (Propositional formula, Atom, Connective). *Atoms are boolean variables (e.g. p, q).*

- *Atoms are formulas.*
- *\top, \perp are formulas.*
- *If A is a formula, then $\neg A$ is a formula.*
- *If A_1, \dots, A_n are formulas, then $(A_1 \wedge \dots \wedge A_n)$ and $(A_1 \vee \dots \vee A_n)$ are formulas.*
- *If A and B are formulas, then $A \rightarrow B$ and $A \leftrightarrow B$ are formulas.*

The symbols $\top, \perp, \wedge, \vee, \neg, \rightarrow, \leftrightarrow$ are called logical connectives.

1.0.2 Precedence

Connective	Name	Precedence
\top	verum	
\perp	falsum	
\neg	negation	5
\wedge	conjunction	4
\vee	disjunction	3
\rightarrow	implication	2
\leftrightarrow	equivalence	1

1.0.3 Boolean Values, Interpretation

Def (Boolean values, Interpretation). *There are two boolean vales: true (1) and false (0).
An interpretation for a set P of boolean variables is a mapping $I : P \rightarrow \{0, 1\}$.*

1.0.4 Interpreting formulas

- $I(\top) = 1$ and $I(\perp) = 0$
- $I(A_1 \wedge \dots \wedge A_n) = 1$ iff $I(A_i) = 1$ for all i
- $I(A_1 \vee \dots \vee A_n) = 1$ iff $I(A_i) = 1$ for some i
- $I(\neg A) = 1$ iff $I(A) = 0$
- $I(A_1 \rightarrow A_2) = 1$ iff $I(A_1) = 0$ or $I(A_2) = 1$
- $I(A_1 \leftrightarrow A_2) = 1$ iff $I(A_1) = I(A_2)$

1.0.5 Satisfiable, Valid, Model

Def (Satisfiable, Model, Valid). *If $I(A) = 1$ then I satisfies A and I is a model of A , denoted by $I \models A$. A is satisfiable if some interpretation is a model of A . A is valid if every interpretation is a model of A . A and B are equivalent, denoted by $A \equiv B$, if they have the same models.*

1.0.6 Connection valid, satisfiable

- A is valid iff $\neg A$ is unsatisfiable.
- A is satisfiable iff $\neg A$ is not valid.

1.0.7 Equivalent replacement

Def (Equivalent replacement). *$A[B]$ is a formula A with a fixed occurrence of subformula B . $A[B']$ is the formula A where every occurrence of B is replaced by B' .*

Lemma 1 (Equivalent Replacement). *Let I be an interpretation and $I \models A_1 \leftrightarrow A_2$. Then $I \models B[A_1] \leftrightarrow B[A_2]$.
Let $A_1 \equiv A_2$. Then $B[A_1] \equiv B[A_2]$.*

1.0.8 Evaluating a formula

Algorithm 1. procedure evaluate(G, I)

input: formula G , interpretation I

output: the boolean value $I(G)$

begin

 forall atoms p occurring in G

 if I models p

 then replace all occurrences of p in G by 1;

 else replace all occurrences of p in G by 0;

 rewrite G into a normal form using the rewrite rules

 if $G = 1$ then return 1 else return 0

end

2 Splitting, Polarities

2.0.1 Soundness of Splitting

A_p^\perp and A_p^\top are obtained by replacing in A all occurrences of p by \perp and \top respectively.

Lemma 2. *Let p be an atom, A be a formula, and I be an interpretation.*

- *If $I \not\models p$, then A is equivalent to A_p^\perp in I .*
- *If $I \models p$, then A is equivalent to A_p^\top in I .*

Lemma 3. *Let A be a formula and p an atom.*

Then A is satisfiable iff at least one of the formulas A_p^\top and A_p^\perp is satisfiable.

2.0.2 Splitting

Algorithm 2. procedure split(G)

parameters: function select

input: formula G

output: ''satisfiable'' or ''unsatisfiable''

begin

$G := \text{simplify}(G)$ # rewrite rules

 if $G = 1$ then return ''satisfiable''

 if $G = 0$ then return ''unsatisfiable''

```

(p,b) := select(G)
case b of
1 =>
  if split(replace(G,p,1)) = ''satisfiable''
    then return ''satisfiable''
    else return split(replace(G,p,0))
0 =>
  if split(replace(G,p,0)) = ''satisfiable''
    then return ''satisfiable''
    else return split(replace(G,p,1))
end

```

2.0.3 Polarities

- $A|_{\epsilon} = A$ and $pol(A, \epsilon) = 1$
- If $A|_{\pi} = B_1 \wedge \dots \wedge B_n$ or $A|_{\pi} = B_1 \vee \dots \vee B_n$ then $A|_{\pi.i} = B_i$ and $pol(A, \pi.i) = pol(A, \pi)$.
- If $A|_{\pi.i} = \neg B$ then $A|_{\pi.1} = B$ and $pol(A, \pi.1) = -pol(A, \pi)$.
- If $A|_{\pi} = B_1 \rightarrow B_2$ then $A|_{\pi.1} = B_1$, $A|_{\pi.2} = B_2$ and $pol(A, \pi.1) = -pol(A, \pi)$, $pol(A, \pi.2) = pol(A, \pi)$.
- If $A|_{\pi} = B_1 \leftrightarrow B_2$ then $A|_{\pi.1} = B_1$, $A|_{\pi.2} = B_2$ and $pol(A, \pi.1) = 0 = pol(A, \pi.2)$.

2.0.4 Monotonic replacement

Denote with $A[B]_{\pi}$ formula A with the subformula at the position π replaced by B .

Lemma 4 (Monotonic Replacement). *Let A, B, B' be formulas, I be an interpretation, and $I \models B \rightarrow B'$. If $pol(A, \pi) = 1$, then $I \models A[B]_{\pi} \rightarrow A[B']_{\pi}$. Likewise, if $pol(A, \pi) = -1$ then $I \models A[B']_{\pi} \rightarrow A[B]_{\pi}$.*

2.0.5 Pure Atom

Def. *Atom p is pure in a formula A , if either all occurrences of p in A are positive or all occurrences of p in A are negative.*

Lemma 5 (Pure Atom). *Let p have only positive occurrences in A and $I \models A$. Define $I' = I + (p \mapsto 1)$. Then $I' \models A$. Likewise, let p have only negative occurrences in A and $I \models A$. Define $I' = I + (p \mapsto 0)$. Then $I' \models A$.*

Lemma 6 (Pure Atom). *Let an atom p have only positive (respectively, only negative) occurrences in A . Then A is satisfiable iff A_p^{\top} (respectively, A_p^{\perp}) is satisfiable.*

2.0.6 Splitting with pure atom optimization

Algorithm 3. procedure split(G)
parameters: function select
input: formula G
output: ''satisfiable'' or ''unsatisfiable''
begin
 G := simplify_with_pure_atoms(G)
 if G = 1 then return ''satisfiable''
 if G = 0 then return ''unsatisfiable''
 (p,b) := select(G)
 case b of
 1 =>
 if split(replace(G,p,1)) = ''satisfiable''
 then return ''satisfiable''
 else return split(replace(G,p,0))
 0 =>

```

    if split(replace(G,p,0)) = ''satisfiable''
    then return ''satisfiable''
    else return split(replace(G,p,1))
end

```

3 CNF, DPLL, MiniSat

3.0.1 Clause

Def (Literal, Clause, Empty clause, Unit clause, Horn clause). *A literal is either an atom p or its negation $\neg p$.*

A clause is a disjunction of literals $L_1 \vee \dots \vee L_n$.

The empty clause \square is false in every interpretation.

If $n = 1$ then the clause is called unit clause.

A horn clause is a clause with at most one positive literal.

3.0.2 CNF

Def (CNF). *A formula A is in conjenctive normal form if it is \top , \perp or a conjunction of disjunctions of literals $\bigwedge_i \bigvee_j L_{i,j}$.*

3.0.3 Naming

If A is a non-trivial subformula A . Introduce a new name n for it. Add formula $n \leftrightarrow A$ and replace subformula by its name in the original formula.

Lemma 7 (Naming). *Let S be a set of formulas and A a formula. Let n be a boolean variable not occurring in S , nor in A .*

Then S is satisfiable iff the set of formulas $S \cup \{n \leftrightarrow A\}$ is satisfiable.

3.0.4 Optimized CNF Transformation

Introduce a new name n every subformula B and replace it with the name. If the subformula occurs only positively then add $n \rightarrow B$. If it occurs only negatively then add $B \rightarrow n$ and if it does not occur only positively or negatively than add $n \leftrightarrow B$.

Lemma 8. *A set of formulas is satisfiable iff the optimized CNF transformation of these formulas is satisfiable.*

3.0.5 Unit propagation

Let S be a set of clauses. If S contains a unit clause L then remove from S every clause of the form $L \vee C$ and replace in S every clause of the form $\bar{L} \vee C$ by the clause C .

3.0.6 DPLL = splitting + unit propagation

Algorithm 4. procedure DPLL(S)

input: set of clauses S

output: satisfiable or unsatisfiable

parameters: function select_literal

begin

$S := \text{propagate}(S)$ # unit propagation

 if S is empty then return satisfiable

 if S contains 0 then return unsatisfiable

$L := \text{select_literals}(S)$ # splitting

 if DPLL($S \cup \{L\}$) = satisfiable

 then return satisfiable

 else return DPLL($S \cup \{\text{not } L\}$)

end

Tautologies (e.g. $p \vee \neg p \vee C$) can be removed.

3.0.7 Pure literals

Def (Pure literal). *A literal L in S is called pure if S contains no clauses of the form $\bar{L} \vee C$.*

If L is a pure literal in S then all clauses containing this literal can be removed.

4 Random SAT, Horn clauses

4.0.1 Random Clause Generation

Fix a number n of boolean variables. Fix the length k of the clause. Choose k times a random literal $p_1, \dots, p_n, \neg p_1, \dots, \neg p_n$ with equal probability.

4.0.2 k-SAT

We can reduce SAT to 3-SAT by naming: Let $L_1 \vee L_2 \vee L_3 \vee L_4 \vee \dots$ be a clause with more than 3 literals. Then we can replace it with $L_1 \vee L_2 \vee n$ and $\neg n \vee L_3 \vee L_4 \vee \dots$ where n is a new variable.

SAT is NP-complete. 2-SAT is decidable in linear time. 3-SAT is NP-complete.

4.0.3 Chaos Algorithm, GSAT, WSAT

Algorithm 5. procedure chaos(S)

input: set of clauses S

output: interpretation I such that I models S or don't know

parameters: positive interger max_tries

begin

 repeat max_tries times

 I := random interpretation

 if I models S then return I

 return don't know

end

Algorithm 6. procedure GSAT(S)

input: set of clauses S

output: interpretation I such that I models S or don't know

parameters: positive intergers max_tries, max_flips

begin

 repeat max_tries times

 I := random interpretation

 if I models S then return I

 repeat max_flips times

 p := a variable such that flip(I, p) satisfies the maximal number of clauses in S

 I = flip(I, p)

 if I models S then return I

 return don't know

end

Algorithm 7. procedure WSAT(S)

input: set of clauses S

output: interpretation I such that I models S or don't know

parameters: positive intergers max_tries, max_flips

begin

 repeat max_tries times

 I := random interpretation

 if I models S then return I

 repeat max_flips times

 randomly select a clause C in S such that I does not model C

 randomly select a variable p in C

 I = flip(I, p)

 if I models S then return I

```

return don't know
end

```

4.0.4 SAT of Horn clauses

Satisfiability of horn clauses can be decided using unit propagation.

5 First-Order Logic, Theories

5.0.1 Syntax

Def (Signature). A signature consists of

- a set of sorts (e.g. integers, arrays of rationals) denoted by α, β .
- constants, denoted by a, b, c . Each constant c has a sort α , written $c : \alpha$.
- function symbols, denoted by f, g . Each function symbol f has a type $\alpha_1 \times \dots \times \alpha_n \rightarrow \alpha$.
- Predicate symbols, denoted by p, q . Each predicate symbol p has a type $\alpha_1 \times \dots \times \alpha_n$.

Variables are not part of the signature, but do have sorts.

Def (Interpretation). An interpretation I maps

- each sort to a non-empty set, called the domain of this sort.
- each constant $c : \alpha$ to an element $c' \in I(\alpha)$.
- each variable $x : \alpha$ to an element $x' \in I(\alpha)$.
- each function symbol $f : \alpha_1 \times \dots \times \alpha_n \rightarrow \alpha$ to a function $f' : I(\alpha_1) \times \dots \times I(\alpha_n) \rightarrow I(\alpha)$.
- each predicate symbol $p : \alpha_1 \times \dots \times \alpha_n$ to a relation p' on $I(\alpha_1) \times \dots \times I(\alpha_n)$.

Def (Term, Atomic Formula). Terms of the sort α are constants $c : \alpha$ or variables $x : \alpha$. If t_1, \dots, t_n are terms of the sorts $\alpha_1, \dots, \alpha_n$ and $f : \alpha_1 \times \dots \times \alpha_n \rightarrow \alpha$, then $f(t_1, \dots, t_n)$ is a term of the sort α . An atomic formula is an expression $p(t_1, \dots, t_n)$ where $p : \alpha_1 \times \dots \times \alpha_n$ and t_1, \dots, t_n are terms of sorts $\alpha_1, \dots, \alpha_n$.

Note that $=$ and $>$ are interpreted, but other symbols are uninterpreted.

Def (Formula, Quantifier, Bound, Free, Ground). Let A be a formula and x a variable, then $\forall xA$ and $\exists xA$ are formulas.

The symbols \forall, \exists are called quantifiers.

A variable occurring in a formula A is called bound, if it is in the scope of a quantifier, otherwise it is called free.

A formula is called ground or quantifier-free if it contains no occurrences of quantifiers.

Def (x-variants). Let \bar{x} be a sequence of variables. We say that two interpretations of the same signature Σ are \bar{x} -variants if they coincide on all symbols and all variables not occurring in \bar{x} .

Def (Extension of interpretation). Let I be an interpretation and t a term of sort α . Define an element $t^I \in I(\alpha)$ as follows.

- for constants $c : \alpha$ and variables $x : \alpha$ we have $c^I \iff I(c)$ and $x^I \iff I(x)$.
- $f(t_1, \dots, t_n)^I \iff f'(t_1^I, \dots, t_n^I)$.
- $p(t_1, \dots, t_n)^I = 1 \iff (t_1^I, \dots, t_n^I) \in p^I$.
- for connectives as before, e.g. $(A \rightarrow B)^I \iff (A^I = 0 \vee B^I = 1)$
- $(\forall xA)^I = 1$ iff for all \bar{x} -variants I' of I we have $(A)^{I'} = 1$.
- $(\exists xA)^I = 1$ iff for some \bar{x} -variants I' of I we have $(A)^{I'} = 1$.

Def (Satisfiable, Valid). A formula A with free variables \bar{x} is said to be satisfiable in an interpretation I if for some \bar{x} -variant I' of A we have $I' \models A$.

A is satisfiable iff it is satisfiable in some interpretation.

A formula A with free variables \bar{x} is said to be valid in an interpretation I if for every \bar{x} -variant I' of A we have $I' \models A$.

A is valid iff it is valid in every interpretation.

A is valid iff $\neg A$ is unsatisfiable.

5.0.2 Theory of Equality

The theory of equality is axiomatized by

- reflexivity, symmetry and transitivity:

$$x = x, x = y \rightarrow y = x, x = y \wedge y = z \rightarrow x = z$$

- congruence axioms for each function symbol f in Σ :

$$x_1 = y_1 \wedge \dots \wedge x_n = y_n \rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$$

- congruence axioms for each predicate symbol p of Σ :

$$x_1 = y_1 \wedge \dots \wedge x_n = y_n \wedge p(x_1, \dots, x_n) \rightarrow p(y_1, \dots, y_n)$$

6 SMT, Theory of Equality, DPLL(T)

6.0.1 Sat modulo theory

Def (T-interpretation, satisfiable modulo theory). Let \mathcal{T} be a theory axiomatized by $A_{\mathcal{T}}$. An interpretation I with $I \models A_{\mathcal{T}}$ then I is called a \mathcal{T} -interpretation.

A formula F is valid in \mathcal{T} if F is valid in every \mathcal{T} -interpretation.

A formula F is satisfiable in \mathcal{T} if there exists a \mathcal{T} -interpretation which satisfies F .

6.0.2 Congruence Closure

We can rewrite predicates into formulas (e.g. $p(x) \wedge \neg q(x, y)$ gets rewritten to $f_p(x) = t \wedge f_q(x, y) \neq t$).

Def (congruence class). The congruence class of $t \in S$ under the congruence relation R is $[t]_R = \{t' \in S \mid tRt'\}$.

Algorithm 8. procedure CongruenceClosure(F)
input: F is $s_1=t_1 \ \& \ \dots \ \& \ s_n = t_n \ \& \ \sim s_{(n+1)}=t_{(n+1)} \ \& \ \dots \ \& \ \sim s_m=t_m$
output: satisfiable or unsatisfiable
parameters: function subterm_set
begin
 SF := subterm_set(F)
 R := {sRs | s in SF} union {{s} | s in SF}
 for every $s_i = t_i$ in F
 union s_i and t_i in R
 propagate by function congruence
 if $s_j R t_j$ for any $j=n+1, \dots, m$ then return unsatisfiable
 else return satisfiable
end

6.0.3 DPLL

For non-unit clauses in any theory:

1. Abstract the problem by renaming every literal in the formulas.
2. Use a SAT solver to find a model.
3. If no model was found return unsat.

4. If a model was found use it to get a unit clause version of the original problem and solve it.
5. If it has a solution return sat.
6. If it does not have a solution rule out the sat model and jump to step 2.

7 Theory of Arrays, Theory Combination, Nelson-Oppen, Z3

7.0.1 Theory of arrays

The theory of arrays \mathcal{T}_A is defined by the signature $\{read, write\}$ and the axioms

$$x = y \rightarrow read(write(A, x, v), y) = v \text{ and } x \neq y \rightarrow read(write(A, x, v), y) = read(A, y).$$

Reduce \mathcal{T}_A to \mathcal{T}_E by

- If F contains no *write* terms, replace $read(A, x)$ with $f_A(x)$ in F .
- If F contains *write* terms ($read(write(A, x, v), y)$)
 1. replace F by $x = y \wedge F[v]$ where $F[v]$ is the formula obtained by replacing $read(write(A, x, v), y)$ by v in F . If this is satisfiable return sat. Else try:
 2. replace F by $x \neq y \wedge F[read(A, y)]$ where $F[read(A, y)]$ is the formula obtained by replacing $read(write(A, x, v), y)$ by $read(A, y)$ in F . If this is satisfiable return sat. Else return unsat.

7.0.2 Combining theories

Algorithm 9. procedure SeparatingReasoning(F)

input: formula F in T_1 union ... union T_n

output: equisatisfiable formulas F_1 in T_1 , ..., T_n in T_n

assumptions: theory signatures S_1 , ..., S_n are disjoint

parameters: function head(t) returning the root symbol of a term t

begin

 repeat as long as possible

 if f in S_i and head(t) in S_j with $i \neq j$:

 rewrite $F[f(t_1, \dots, t_n)]$ into $F[f(t_1, \dots, c, \dots, t_n)]$ & $c=t$ where c is a new variable

 if p in S_i and head(t) in S_j with $i \neq j$:

 rewrite $F[p(t_1, \dots, t_n)]$ into $F[p(t_1, \dots, c, \dots, t_n)]$ & $c=t$ where c is a new variable

 if head(s) in S_i and head(t) in S_j with $i \neq j$:

 rewrite $F[s=t]$ into $F[s=c]$ & $c=t$ where c is a new variable

 end repeat

 return modified F as F_1 & ... & F_n with each F_i in T_i

This only works if the theories have disjoint signatures and if each theory is stably infinite.

Def (stably infinite, convex). A theory \mathcal{T} with signature Σ is stably infinite if for every satisfiable formula $F \in \mathcal{T}$ there exists some \mathcal{T} interpretation such that $I \models F$ and I has a domain of infinite cardinality. A theory \mathcal{T} is convex if for every formula $F \in \mathcal{T}$ such that F is a conjunction of \mathcal{T} -literals: if $F \rightarrow \bigvee_{j=1}^k (u_j = v_j)$ then $F \rightarrow u_j = v_j$ for some $j \in \{1, \dots, k\}$.

Both $\mathcal{T}_{\mathbb{Z}}$ and \mathcal{T}_A are not convex, but \mathcal{T}_E and \mathcal{T}_Q are.

7.0.3 Exchange of equality between theories

For a convex theory its decision procedure discovers a new equality $u = v$ for shared u, v . Pass this new equality to the decision procedure of the other theories.

For a non-convex theory its decision procedure discovers a disjunction of new equalities $\bigvee_k u_k = v_k$ for shared u_k, v_k . Split the disjunction and exchange equalities along multiple branches.

8 First-Order Theorem Proving, TPTP, Inference Systems

8.0.1 TPTP Syntax

fof(name, axiom/hypothesis/conjecture,
formula).

FOL	TPTP
\top, \perp	<code>\$false, \$true</code>
$\neg a$	<code>~a</code>
$a_1 \wedge \dots \wedge a_n$	<code>a1&...&an</code>
$a_1 \vee \dots \vee a_n$	<code>a1 ... an</code>
$a_1 \rightarrow a_2$	<code>a1=>a2</code>
$(\forall x_1) \dots (\forall x_n) a$	<code>![X1,...,Xn]:a</code>
$(\exists x_1) \dots (\exists x_n) a$	<code>?[X1,...,Xn]:a</code>

8.0.2 Inference System

Def (Inference, Inference rule, Inference system, Derivation, Proof). *An inference has the form*

$$\frac{F_1 \dots F_n}{G}$$

where $n \geq 0$ and F_1, \dots, F_n, G are formulas. G is called the conclusion of the inference. F_1, \dots, F_n are called the premises.

An inference rule R is a set of inferences. Every inference $I \in R$ is called an instance of R .

An Inference system \mathbb{I} is a set of inference rules.

Axiom is a inference rule with no premises.

A derivation in an inference system \mathbb{I} is a tree built from inferences in \mathbb{I} . If the root of this derivation is E , we say it is a derivation of E . A derivation of E from E_1, \dots, E_m is a finite derivation of E whose every leaf is either an axiom or one of the expressions E_1, \dots, E_m .

A proof of E is a finite derivation whose leaves are axioms.

Def (Soundness, Completeness). *An inference is sound if the conclusion is a logical consequence of its premises.*

An inference system is sound if every inference rule in this system is sound.

An inference system is complete, if for every unsatisfiable S there exists a derivation of \square from S in the inference system.

8.0.3 Ground Binary Resolution Inference System

\mathbb{BR} consists of two inference rules:

$$\frac{p \vee C_1 \quad \neg p \vee C_2}{C_1 \vee C_2} (BR) \qquad \frac{L \vee L \vee C}{L \vee C} (Fact)$$

\mathbb{BR} is sound.

\mathbb{BR} with selection:

$$\frac{\underline{p \vee C_1} \quad \underline{\neg p \vee C_2}}{C_1 \vee C_2} (BR) \qquad \frac{\underline{p \vee p \vee C}}{p \vee C} (Fact) \qquad \frac{\underline{\neg p \vee \neg p \vee C}}{\neg p \vee C} (Fact)$$

8.0.4 Literal Orderings

Def (well-founded, well-behaved). $>$ is called a well-founded ordering on atoms, if there exists no infinite decreasing chain of atoms.

For a fixed ordering $>$ a well-behaved selection function selects either a negative literal or all maximal literals w.r.t. $>$ must be selected.

We can extend an ordering on atoms to one on literals by $p > q \implies \neg p > p > \neg q > q$.

\mathbb{BR} with selection is complete for every well-behaved selection function.

9 Selection functions, Saturation, Fairness and Redundancy

9.0.1 Idea of a saturation algorithm

Initially $S = S_0$. Repeatedly apply inferences in \mathbb{I} to clauses in S and add their conclusions to S . Repeat until we either obtain \square and know S_0 is unsat, or until every conclusion already is in S and we know S_0 is sat.

Def (Saturated, Closure, Inference process, Limit). S is called saturated with respect to \mathbb{I} , if for every inference of \mathbb{I} with premises in S , the conclusion of this inference also belongs to S .

The closure of S with respect to \mathbb{I} is the smallest set S' containing S and saturated with respect to \mathbb{I} .

An \mathbb{I} -inference process is a sequence of sets of formulas $S_0 \Rightarrow S_1 \Rightarrow \dots$ where every step is an \mathbb{I} -step. $S_i \Rightarrow S_{i+1}$ is an \mathbb{I} -step if there exists an inference $\frac{F_1 \dots F_n}{F}$ in \mathbb{I} such that $\{F_1, \dots, F_n\} \subseteq S_i$ and $S_{i+1} = S_i \cup \{F\}$.

The limit of an inference process $S_0 \Rightarrow S_1 \Rightarrow \dots$ is $\bigcup_i S_i$.

The limit of an inference process is the set of all derived formulas.

Def (Fairness). Let $S_0 \Rightarrow S_1 \Rightarrow \dots$ be an inference process with the limit S_ω . The process is called fair if for every \mathbb{I} -inference $\frac{F_1 \dots F_n}{F}$ where $\{F_1, \dots, F_n\} \subseteq S_\omega$ then there exists an i such that $F \in S_i$.

Lemma 9. \mathbb{I} is complete iff for every unsatisfiable set of formulas S_0 and any fair \mathbb{I} -inference process with the initial set S_0 , the limit of this inference process contains \square .

There are three possible scenarios when performing a saturation algorithm:

- The empty clause is generated at some point. In this case the input set of clauses is unsat.
- Saturation terminates without generating the empty clause. In this case the input set of clauses is sat.
- The saturation runs forever without generating the empty clause. In this case the input set of clauses is sat, but in practice we will run out of resources and the satisfiability is unknown.

9.0.2 Redundancy

Tautologies (e.g. $p \vee \neg p \vee C$) and subsumed clauses (C and $C \vee D$) are redundant.

Def (Bag extension). Let $>$ be any (strict) ordering on a set X . The bag extension of $>$ is a binary relation $>^{bag}$ on bags over X defined as the smallest transitive relation on bags such that

$$\{x, y_1, \dots, y_n\} >^{bag} \{x_1, \dots, x_m, y_1, \dots, y_n\} \text{ if } \forall i \in \{1, \dots, m\} : x > x_i$$

$>^{bag}$ is an ordering. If $>$ is total then so is $>^{bag}$. If $>$ is well-founded then so is $>^{bag}$.

We can now order clauses by interpreting them as bags of literals.

Def (Redundancy). A clause $C \in S$ is called redundant in S if it is a logical consequence of clauses in S strictly smaller than C .

10 Redundancy, First-Order Reasoning with Equality

11 Ground Superposition, Term Orderings

12 Unification and Lifting

13 Non-Ground Superposition